HITACHI
Inspire the Next

uCosminexus Application Server

Web Service Development Guide

3020-3-Y23-10(E)

■ Relevant program products

See the manual *uCosminexus Application Server Overview*.

■ Export restrictions

If you export this product, please check all restrictions (for example, Japan's Foreign Exchange and Foreign Trade Law, and USA export control laws and regulations), and carry out all required procedures.

If you require more information or clarification, please contact your Hitachi sales representative.

■ Trademarks

GIF is a format name developed by the CompuServe Inc. in the United States.

gzip is a software provided by the Free Software Foundation in the United States.

Microsoft is either a registered trademark or trademark of Microsoft Corporation in the United States and/or other countries.

Microsoft Office and Excel are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

Oracle and Java are registered trademarks of Oracle and/or its affiliates in the United States and/or other countries.

SOAP (Simple Object Access Protocol) is an XML-based communication protocol for exchanging information in a distributed environment.

W3C is a trademark (registered in numerous countries) of the World Wide Web Consortium.

Windows is either a registered trademark or trademark of Microsoft Corporation in the United States and/or other countries.

Windows Server is either a registered trademark or trademark of Microsoft Corporation in the United States and/or other countries.

Windows Vista is either a registered trademark or trademark of Microsoft Corporation in the United States and/or other countries.

Other product and company names mentioned in this document may be the trademarks of their respective owners. Throughout this document Hitachi has attempted to distinguish trademarks from descriptive terms by writing the name with the capitalization used by the manufacturer, or by writing the name with initial capital letters. Hitachi cannot attest to the accuracy of this information. Use of a trademark in this document should not be regarded as affecting the validity of the trademark.

■ Microsoft product screen shots

Microsoft product screen shots reprinted with permission from Microsoft Corporation.

■ Microsoft product name abbreviations

This manual uses the following abbreviations for Microsoft product names.

Abbreviation			Full name or meaning
Windows	Windows 8	Windows 8 x86	Windows(R) 8 Pro (32-bit)
			Windows(R) 8 Enterprise (32-bit)
		Windows 8 x64	Windows(R) 8 Pro (64-bit)
			Windows(R) 8 Enterprise (64-bit)
	Windows 7	Windows 7 x86	Microsoft(R) Windows(R) 7 Professional (32-bit)
			Microsoft(R) Windows(R) 7 Enterprise (32-bit)
			Microsoft(R) Windows(R) 7 Ultimate (32-bit)
		Windows 7 x64	Microsoft(R) Windows(R) 7 Professional (64-bit)
			Microsoft(R) Windows(R) 7 Enterprise (64-bit)
			Microsoft(R) Windows(R) 7 Ultimate (64-bit)
	Windows Server 2012 Standard		Microsoft(R) Windows Server(R) 2012 Standard
	Windows Server 2012 Datacenter		Microsoft(R) Windows Server(R) 2012 Datacenter
	Windows Server 2008 x86		Microsoft(R) Windows Server(R) 2008 Standard 32-bit
			Microsoft(R) Windows Server(R) 2008 Enterprise 32-bit

Abbreviation		Full name or meaning
Windows	Windows Server 2008 x64	Microsoft(R) Windows Server(R) 2008 Standard
		Microsoft(R) Windows Server(R) 2008 Enterprise
	Windows Server 2008 R2	Microsoft(R) Windows Server(R) 2008 R2 Standard
		Microsoft(R) Windows Server(R) 2008 R2 Enterprise
		Microsoft(R) Windows Server(R) 2008 R2 Datacenter
	Windows Vista	Microsoft(R) Windows Vista(R) Business (32-bit)
		Microsoft(R) Windows Vista(R) Enterprise (32-bit)
		Microsoft(R) Windows Vista(R) Ultimate (32-bit)
	Windows XP	Microsoft(R) Windows(R) XP Professional Operating System

Note that Windows 32 bit and Windows 64 bit are sometimes respectively referred to as Windows x86 and Windows x64.

■ Restrictions

■ Issued

Aug. 2013: 3020-3-Y23-10(E)

■ Copyright

Summary of amendments

The following table lists changes in the manual 3020-3-Y23-10(E) for uCosminexus Application Server 09-50, uCosminexus Application Server(64) 09-50, uCosminexus Client 09-50, uCosminexus Developer 09-50, uCosminexus Service Architect 09-50, uCosminexus Service Platform 09-50, and uCosminexus Service Platform(64) 09-50 and product changes related to the manual:

Changes	Location
The catalog functionality has been added. Accordingly, the following description has been changed: • The support range of the JAX-WS 2.2 specifications has been changed.	*1.1*, *14.1(2)*, *15.1.9(1)*, *16.2.1*, *19.1.2*, *19.2.2(4)*, *19.2.3(3)*, *21*, *27*
The description on client APIs for RESTful Web Services has been added. Accordingly, the following descriptions have been added or changed: • The description on some definitions has been added in the action definition file that you can overwrite through client APIs. • The settings of a common definition file have been added. • A description has been added under each annotation with respect to its functioning on the Web Service client side. • The description on the support range of the `UriBuilder` class has been added under the support range of the interfaces and classes of `JAX-RS API`. • The description on the output destination and the format of the logon client has been added. • The description on the performance analysis trace has been added with respect to the functioning of the performance analysis trace on the client side.	*1.3.2*, *1.5.2*, *11.4*, *12.5.1*, *12.6.3*, *13.1*, *13.1.2(3)*, *13.2*, *13.4*, *13.5*, *13.6*, *13.7.5*, *16.2.2*, *16.2.19*, *16.2.20*, *24.2*, *24.2.12*, *25*, *39.3.3(5)*, *39.3.5(2)*, *39.4.2*
The description on one-way operations has been added. Accordingly, the following descriptions have been added or changed: • The support range of the JAX-WS 2.2 specifications has been changed. • The description regarding response of outbounding and inbounding when using the handler framework has been added. • The description regarding the process of the `close` method when using the handler framework has been added. • The description on operations on receiving request messages when using the addressing functionality has been added. • The description regarding the trace collection point of the performance analysis trace has been added.	*1.4.2(1)*, *10.15*, *10.22*, *15.1.3(3)*, *15.1.4(1)*, *15.1.5(1)*, *16.2.1*, *16.2.4*, *16.2.9(2)*, *19.1.1*, *19.1.2*, *19.2.2(1)*, *19.2.2(2)*, *36.1*, *36.5.3*, *37.5.1*, *39.4.2(2)*
The description regarding the dynamic generation of wrapper beans has been added. Accordingly, the following descriptions have been added or changed: • The description regarding the dynamic generation of stubs by using the JAX-WS engine has been added. • The command that compiles Web Services Implementation Class when developing Web Services starting from SEI has been added to the `javac` command. • The description regarding the use of the `cjwsgen` command for checking errors has been added. • The description on operations when the `javax.xml.ws.Holder` type is specified in a parameter has been added.	*1.4.2(1)*, *1.5.1(1)*, *2.1.2*, *2.1.3*, *5.2*, *5.3.2*, *7.2*, *7.3.2*, *8.2*, *8.3.2*, *10.9(1)*, *10.23*, *16.1.5*, *19.1.2*, *29.2*, *29.3.2*, *31.2*, *31.3.2*, *33.2*, *33.3.2*, *38.2*, *38.3.2*

Changes	Location
The description on injecting service classes and ports has been added in the `javax.xml.ws.WebServiceRef` annotation.	*3.6.1*, *1.4.2(1)*, *10.21*, *19.3*
The description regarding injecting or reusing ports for generating service classes and acquiring ports has been added.	*3.6.1(3)*, *3.6.1(5)*, *3.6.1(7)*
The value of the context root has been changed and the current directory has been renamed.	*5.1*, *5.3.3*, *5.3.4*, *5.3.6*, *5.4.1*, *5.4.2*, *5.5.1*, *7.1*, *7.3.3*, *7.3.4*, *7.3.5*, *7.4.1*, *7.4.2*, *7.5.1*, *8.1*, *8.3.3*, *8.3.5*, *8.4.1*, *8.4.2*, *8.5.1*, *29.1*, *29.3.4*, *29.3.5*, *29.4.1*, *29.5.1*, *31.1*, *31.3.3*, *31.3.4*, *31.3.5*, *31.4.1*, *31.4.2*, *31.5.1*, *33.1*, *33.3.3*, *33.3.4*, *33.3.5*, *33.4.1*, *33.4.2*, *33.5.1*, *38.1*, *38.3.3*, *38.3.4*, *38.3.5*, *38.4.1*, *38.4.2*, *38.5.1*, *38.5.2*, *Appendix B(4)*
The binding results of the wrapper exception class have been changed for `soapenv12:Code` of the SOAP 1.2 specifications.	*10.4.1*
The description regarding injecting Web Services context in the `javax.annotation.Resource` annotation has been added	*10.19(1)*, *10.21*, *10.21.2*
A sample of the development example of RESTful Web Services has been provided.	*12.1*
The description on the JSON POJO mapping has been added.	*12.3.1*, *12.3.3*, *13.1.2(3)*, *17.1.2(1)*, *17.1.3*, *18*
The number of child elements of operations that can be coded in the WSDL when mapping from an operation to the name of a Java method has been changed.	*15.1.3(3)*
The notes on entity parameters and return values have been added.	*17.1.2(1)*, *17.1.2(2)*, *17.1.3*
The notes on parameter types for which you can specify the injection annotation have been added.	*17.1.4*
The description on the `javax.xml.ws.WebServiceContext` interface has been added.	*19.2.1(2)*, *19.2.3(2)*
The description on the `getBinding()` method of the `javax.xml.ws.BindingProvider` interface has been added.	*19.2.2(1)*
The description on the `javax.xml.ws.Binding` interface has been added.	*19.2.4(4)*
The description on the `setScope(java.lang.String name, MessageContext.Scope scope)` method of the `javax.xml.ws.handler.MessageContext` interface has been added.	*19.2.4(8)*
The description on how to access a message context from Web Services has been added.	*19.2.5*
The description on referencing and changing a message context property in Web Services has been added.	*19.2.5(2)(l)*
The description on the support range of the following elements of the WSDL 1.1 specifications has been changed: • `wsdl:output` element • `wsdl:operation` element (for the child element of the `wsdl:binding` element) • `wsdl:output` element (for the grandchild element of the `wsdl:binding` element)	*20.1.9*, *20.1.12*, *20.1.14*
The description on adding codes by using the APIs to set a handler chain of ports has been added.	*36.9.2(1)*
The description on notes has been moved from Release Notes.	*10.6(7)*, *19.2.5(2)(c)*, *30.6*, *39.3.5(3)*, *39.3.5(4)*, *Appendix A.1(4)*,

Changes	Location
The notes regarding the version upgrade from 08-00 to 08-70 through 09-00 have been added.	*Appendix A.1(4)*

In addition to the above changes, minor editorial corrections have been made.

Preface

For details on the prerequisites before reading this manual, see the manual *uCosminexus Application Server Overview*.

■ Non-supported functionality

Some functionality described in this manual are not supported. The non-supported functionality include:

- Audit log functionality
- Compatibility functionality
- Cosminexus Component Transaction Monitor
- Cosminexus DABroker Library
- Cosminexus Reliable Messaging
- Cosminexus TPBroker and VisiBroker
- Cosminexus Web Service - Security
- Cosminexus XML Security - Core functionality
- JP1 linkage functionality
- Management portal functionality
- Migration functionality
- SOAP applications complying with specifications other than JAX-WS 2.1
- uCosminexus OpenTP1 linkage functionality
- Virtualized system functionality
- XML Processor high-speed parse support functionality

■ Non-supported compatibility functionality

"Compatibility functionality" in the above list refers to the following functionality:

- Basic mode
- Check of JSP source compliance (cjjsp2java) with JSP1.1 and JSP1.2 specifications
- Database connection using Cosminexus DABroker Library
- EJB client application log subdirectory exclusive mode
- J2EE application test functionality
- Memory session failover functionality
- Servlet engine mode
- Simple Web server functionality
- Switching multiple existing execution environments
- Using EJB 2.1 and Servlet 2.4 annotation

Contents

Part 3: References

Part 5: Troubleshooting

1 Overview of Developing and Executing Web services

You can use the JAX-WS functionality of Cosminexus to develop SOAP Web services that are compliant with the JAX-WS 2.2 specifications. You can also use the JAX-RS functionality of Cosminexus to develop RESTful Web Services (Web resource) that are compliant with the JAX-RS 1.1 specifications.

This chapter gives an overview of Web services, and describes the prerequisites for developing them.

1.1 JAX-WS/JAX-RS specifications compliant version, prefix and name space URI

This manual describes how to develop and execute the web services compliant with the JAX-WS 2.2 or JAX-RS 1.1 specifications by using the functionality provided with Cosminexus.

The following are the details on the JAX-WS 2.2 specifications, JAX-RS1.1 specifications, and the mapping of the prefixes and name space URIs.

JAX-WS 2.2 specifications compliant version

The *JAX-WS 2.2 specifications* described in this manual are the specifications of the following version:

```
Specification: JSR-000224 - Java™ API for XML-Based Web services
Version: 2.2
Status: Maintenance Release 3
Release: 10 December 2009
```

The *schema of the binding declaration* described in this manual is the following schema:

```
http://java.sun.com/xml/ns/jaxws/wsdl_customizationschema_2_0.xsd(Date Published:
May 11, 2006)
```

JAX-RS 1.1 specifications compliant version

The JAX-RS 1.1 specifications described in this manual are the specifications of the following version:

```
Specification: JSR-000311 - JavaTM API for RESTful Web services
Version: 1.1
Status: Final Release
Release: September 17, 2009
```

Mapping of the prefixes and name space URIs

The mapping of the prefixes and name space URIs used in this manual is as follows. Unless otherwise specified, the following prefixes are used.

Table 1-1: Mapping of the prefixes and namespace URIs

No.	Prefix	Namespace URI
1	cwsrm	http://jaxws.cosminexus.com/cwsrm
2	er	urn:oasis:names:tc:entity:xmlns:xml:catalog
3	javaee	http://java.sun.com/xml/ns/javaee
4	jaxb	http://java.sun.com/xml/ns/jaxb
5	jaxws	http://java.sun.com/xml/ns/jaxws
6	jaxwsdd	http://java.sun.com/xml/ns/jax-ws/ri/runtime
7	net35rmp	http://schemas.microsoft.com/ws-rx/wsrmp/200702
8	S	Indicates S11 or S12
9	S11	http://schemas.xmlsoap.org/soap/envelope/
10	S12	http://www.w3.org/2003/05/soap-envelope
11	soap	http://schemas.xmlsoap.org/wsdl/soap/
12	soap12	http://schemas.xmlsoap.org/wsdl/soap12/
13	soapenv	http://schemas.xmlsoap.org/soap/envelope/
14	soapenv12	http://www.w3.org/2003/05/soap-envelope
15	wsa	http://www.w3.org/2005/08/addressing
16	wsam	http://www.w3.org/2007/05/addressing/metadata

No.	Prefix	Namespace URI
17	wsaw	http://www.w3.org/2006/05/addressing/wsdl
18	wsdl	http://schemas.xmlsoap.org/wsdl/
19	wsi	http://ws-i.org/profiles/basic/1.1/xsd
20	wsp	http://www.w3.org/ns/ws-policy
21	wsrm	http://docs.oasis-open.org/ws-rx/wsrm/200702
22	wsrmp	http://docs.oasis-open.org/ws-rx/wsrmp/200702
23	wsu	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd
24	xmime	http://www.w3.org/2005/05/xmlmime
25	xop	http://www.w3.org/2004/08/xop/include
26	xsd	http://www.w3.org/2001/XMLSchema
27	xsi	http://www.w3.org/2001/XMLSchema-instance

1.2 Overview of developing Web services

Cosminexus provides a JAX-WS engine and JAX-RS engine as the communication base for the Web services. The JAX-WS engine binds the SOAP Messages according to the JAX-WS 2.2 specifications, whereas the JAX-RS engine binds the RESTful HTTP messages according to the JAX-RS 1.1 specifications.

This manual describes how to develop the Web services that can be used through JAX-WS engines or JAX-RS engines.

1.2.1 Overview of developing SOAP Web services

You develop a SOAP Web service (Web service using the JAX-WS engine) by using one of the following methods:

- **Development starting from WSDL**

  The development starts from the WSDL that is used for coding Web Service definitions. Use commands to generate Java sources from the WSDL and implement the required processing.
- **Development starting from SEI (Service End Point Interface)**

  The development starts from SEI. Use commands to generate the additional Java source (Java Beans class) required for implementing Web services. You can create Web Service Implementation Class as a POJO and which is based on EJB.
- **Development starting from a provider**

  The development starts from a provider. Implement the provider implementation class.

For details on each of these development procedures, see the chapter *2.1 Development flow of SOAP Web Services*.

Reference note

Developing SOAP applications

You can also use the existing SOAP application development support functionality and the SOAP Communication Infrastructure functionality to develop Web services (applications developed using this functionality are called *SOAP applications*). However, as a prerequisite the SOAP applications must run on the SOAP Communication Infrastructure, and therefore, there are conditions for using the SOAP applications on JAX-WS engines.

For developing SOAP applications, see the *uCosminexus Application Server SOAP Application Development Guide*.

For the conditions and migration procedures when using SOAP applications on JAX-WS engines, see *Appendix A Migrating from an Earlier Version*.

1.2.2 Overview of developing RESTful Web services

You develop the RESTful Web Services (Web services using the JAX-RS engine) by implementing Web resources. For Web resources, implement the processes corresponding to the HTTP requests. Note that at times the RESTful Web services are called a Web resource.

For details on the flow of development, see *2.3 Development flow of RESTful Web Services*.

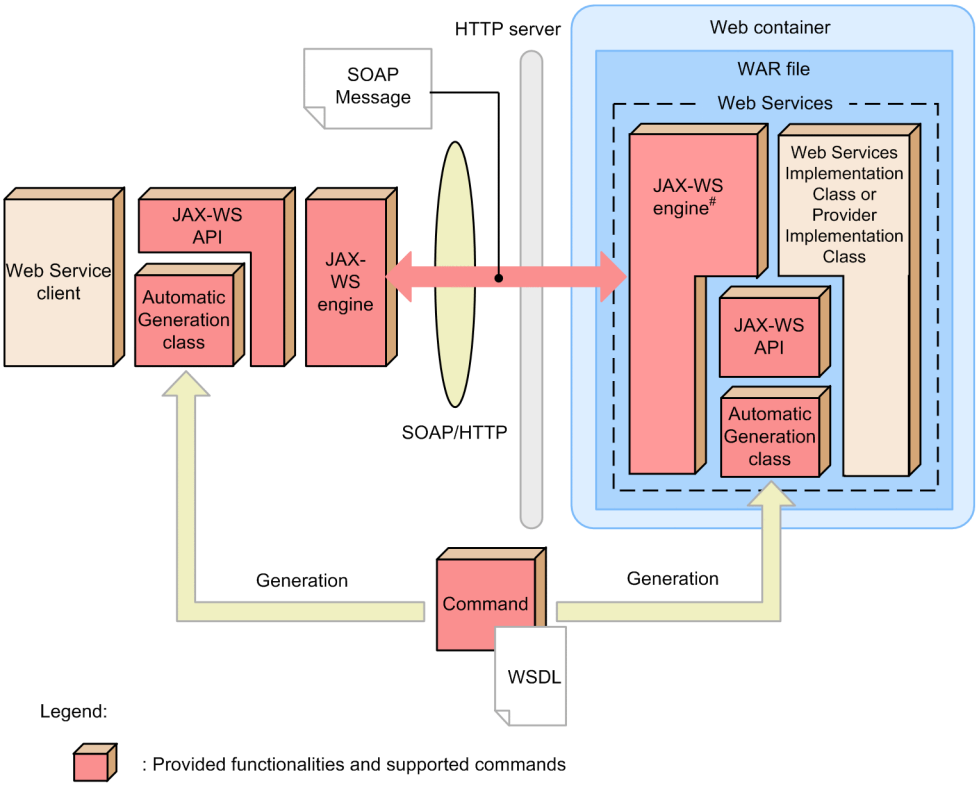
1.3 Functionality used for developing and executing the Web services

This section describes the functionality to be used for developing and executing SOAP Web services and RESTful Web Services (Web resources).

1.3.1 Functionality of SOAP Web services

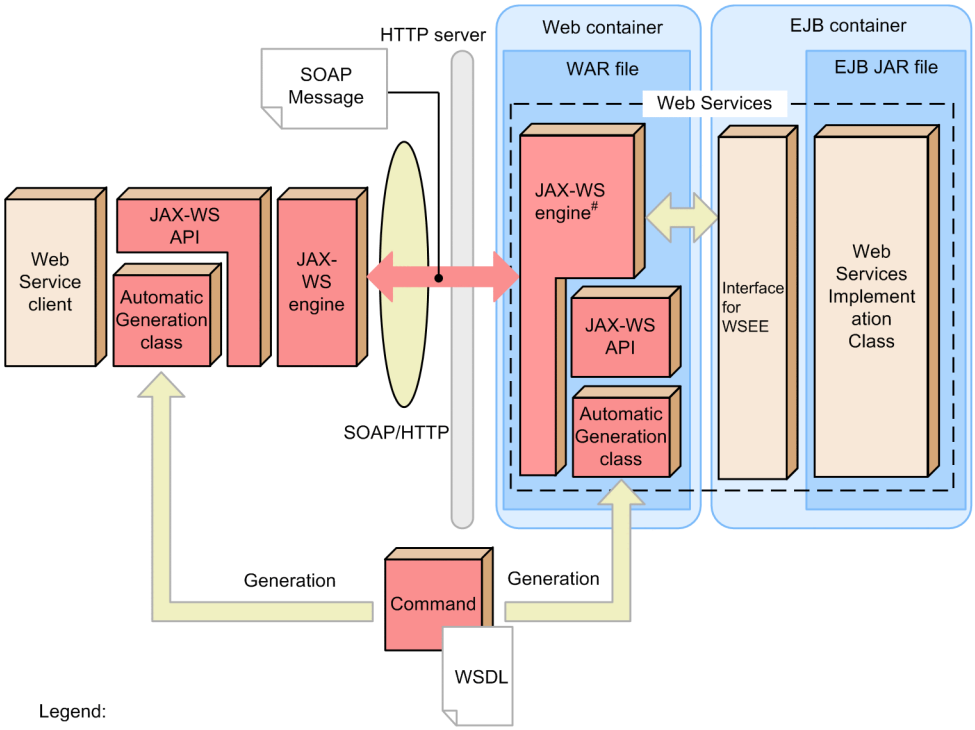
The functionalities of a JAX-WS engine are described based on the configuration of the SOAP Web services shown in the following figure.

Figure 1-1: Configuration of a SOAP Web service (POJO Web service)



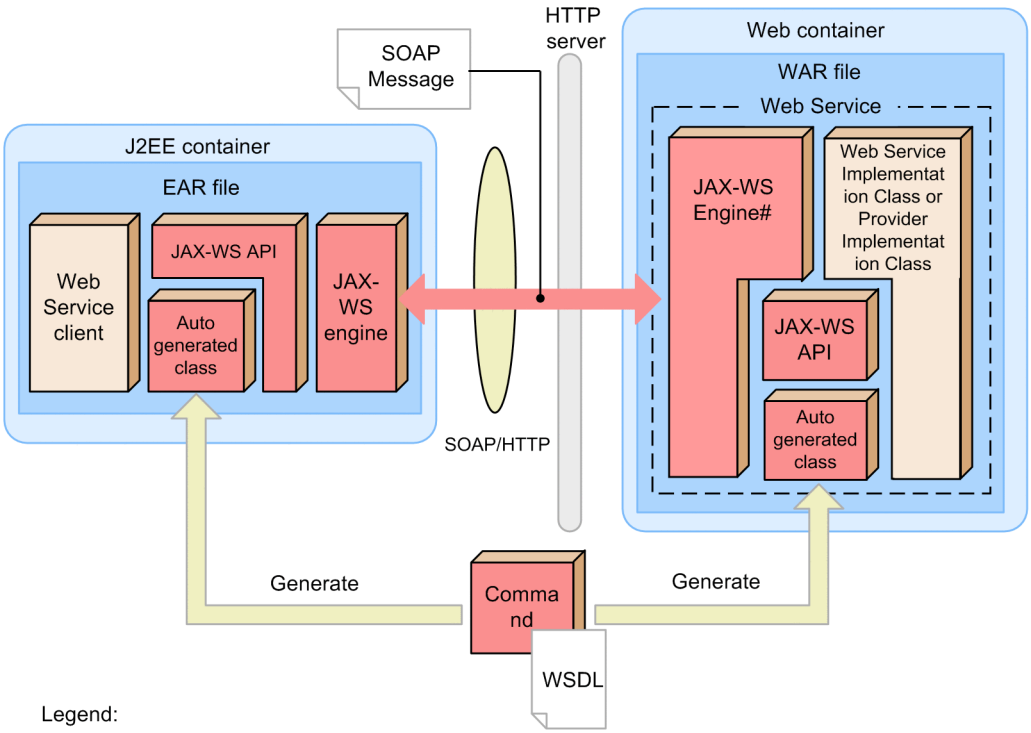
#
The JAX-WS engine and JAX-WS API of the Web Services side are called as the library, but are not included in the WAR file.

Figure 1-2: Configuration of a SOAP Web service (EJB Web service)



#
The JAX-WS engine and JAX-WS API on the Web Services machine are called as the library, but are not included in the WAR file.

Figure 1-3: Configuration of SOAP Web Services (Web service client that operates on a J2EE container)



- **JAX-WS engine**

  A JAX-WS engine is a communication infrastructure of SOAP Web services. The JAX-WS engine is deployed on a Web Service machine and on a Web Service client machine, and plays the role of marshalling/ un-marshalling of the sent and received SOAP Messages.

  JAX-WS engine on the Web Service client machine

  The JAX-WS engine receives Java objects through JAX-WS APIs from the Web Service client, and generates SOAP request messages (*marshalling*). The generated SOAP request messages are sent to the Web services of invoking destination.

  Moreover, the JAX-WS engine receives SOAP response messages from Web services and generates Java objects (*unmarshalling*) . The generated Java objects are returned to the Web Service client.

  JAX-WS engine on the Web Service machine

  The JAX-WS engine receives SOAP request messages from the Web Service client, and generates Java objects (unmarshalling). At this stage, the JAX-WS engine discovers the target Web Service Implementation Class or the Provider Implementation Class, (*discovery*)and invokes a method for the operation (dispatch). (*dispatch*)The JAX-WS engine also receives Java objects from the target Web Service Implementation Class or the Provider Implementation Class, and generates SOAP response messages (marshalling). The generated SOAP response message is returned to the Web Service client that is the invoking source.

- **Command**

  The commands are used for developing SOAP Web services. You can generate Java sources and WSDL required for implementing Web services and Web Service clients. Use the following commands for developing Web services using the JAX-WS engine:

  - `cjwsimport` command
  - `apt` command
  - `cjwsgen` command

For using commands, see the chapter *14. Commands*.

- **Automatically generated class**

  Execute commands to generate Java sources. Implement Web services and Web Service clients by using the generated Java sources.

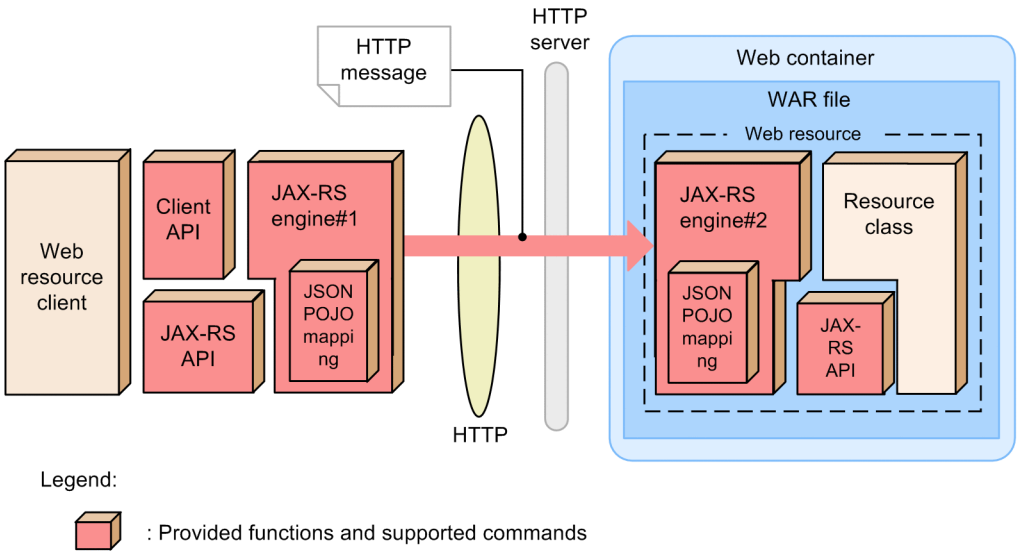
- **JAX-WS APIs**

  The APIs of the JAX-WS 2.2 specifications. Use the APIs of JAX-WS for developing Web services starting from a provider and developing dispatch-based Web Service clients. You can also use APIs for adding the handler framework and addressing functionality.

1.3.2 Functionality of RESTful Web services

This section describes the functionality of a JAX-RS engine based on the configuration of RESTful Web Services (Web resources) shown in the following figure.

Figure 1-4: Configuration of RESTful Web Services



- **JAX-RS** engine

  The JAX-RS engine acts as a communication base of the RESTful Web Services (Web resource).

  - *JAX-RS engine on a Web resource client*

    The JAX-RS engine receives Java objects from Web resource clients through client APIs for RESTful Web Services and generates HTTP requests. The engine then sends the generated HTTP requests to the Web resource to be invoked.

    Next, the engine receives HTTP responses from Web resources and generates Java objects and then returns the generated Java objects to the Web resource client.

  - *JAX-RS engine on a Web resource*

    The engine, deployed on a Web resource, receives the HTTP requests from the client, finds the target resource class (discovery), and calls a method corresponding to the request (dispatch). The engine also generates the HTTP response from the target resource class, and returns the response to the caller client. When dispatching, the JAX-RS engine executes the required injections based on the annotations included in the resource class.

- **Client APIs for RESTful Web Services**

  This is an API that can be used on a client that calls RESTful Web Services (Web resource).

1.4 Prerequisites for developing and executing Web services

This section describes the prerequisites for developing Web services on Cosminexus, and support range.

1.4.1 Prerequisite component software

To develop and execute Web services, use Developer and the Application Server respectively.

For the prerequisite software for Developer, see *1.3 Machine configuration of the development environment* in the *uCosminexus Application Server Application Development Guide*.

For the prerequisite software for Application Servers, see *2.2 Component software* in the *uCosminexus Application Server Overview*.

1.4.2 Prerequisites related to functionality and specifications

(1) Preconditions related to the functionality and specifications of SOAP Web services

This subsection describes the functionality and specifications that you can use for developing SOAP Web services in Cosminexus. For details on the support range of the standard specifications, see the following sections:

- *19. Support Range of the JAX-WS specifications*
- *20. Support Range of the WSDL specifications*
- *22. Support Range of the SAAJ specifications*
- *23. Support Range of the WS-RM specifications*

(a) Default mapping

The commands provided with the JAX-WS functionality of Cosminexus operate according to the default mapping of WSDL to Java that is defined in *Chapter 2* of *JAX-WS 2.2 specifications* and according to the default mapping of Java to WSDL that is defined in *Chapter 3* of *JAX-WS 2.2 specifications*.

Also, when the JAX-WS engine on the Web Service machine requires the WSDL that is the meta data for Web services, and if the WSDL does not exist in WAR files or EJB JAR files generate the WSDL according to the default mapping of Java to the WSDL defined in *Chapter 3* of JAX-WS 2.2 specifications.

For the default mapping of the WSDL to Java, see the subsection *15.1 Default mapping of WSDL to Java*. For the default mapping of Java to WSDL, see the subsection *16.1 Default mapping of Java to WSDL*.

(b) Customization of mapping

The commands provided with the JAX-WS functionality of Cosminexus operate according to the customized mapping (binding declaration) of the WSDL to Java that is defined in *Chapter 7* of JAX-WS 2.2 specifications and according to the customized mapping (annotation) of Java to the WSDL that is defined in *Chapter 8* of JAX-WS 2.2 specifications.

Also, when the JAX-WS engine on the Web Service machine requires the WSDL that is the meta data for Web services, and if the WSDL does not exist in WAR files or EJB JAR files, generate the WSDL according to the customized mapping of Java to the WSDL defined in *Chapter 7* of JAX-WS 2.2 specifications.

For the WAR files, see *3.5.1 Configuring WAR files*. For the EJB JAR files, see *3.5.2 Configuring EJB JAR files*. For the customized mapping of the WSDL to Java, see *15.2 Customized mapping of WSDL to Java*. For the customized mapping of Java to the WSDL, see *16.2 Customized mapping of Java to WSDL*.

(c) Binding between Java and WSDL

The JAX-WS engine of Cosminexus binds Java and the WSDL according to *Chapter 2* and *Chapter 3* of the JAX-WS 2.2 specifications for both Web services and Web Service clients.

For the support range of JAX-WS engines of Cosminexus, see *10.2 Operations of the JAX-WS engine*.

(d) WSDL specifications

The JAX-WS functionality of Cosminexus supports the WSDL of WSDL 1.1 specifications. Only the document/literal style is supported as the WSDL definition style. For the document/literal style, you can use both the wrapper style and the non-wrapper style.

For the support range of WSDL 1.1 specifications, see *19.1 Support range of WSDL 1.1 specifications*.

(e) SOAP specifications

The JAX-WS functionality of Cosminexus supports SOAP Messages of the SOAP 1.1 and SOAP 1.2 specifications.

(f) Message Exchange Pattern (MEP)

The JAX-WS functionality of Cosminexus only supports the request-response operations and one-way operations as MEP. For details on how to define the request-response operations and one-way operations, see the following subsections:

- *15.1.4 Mapping the message part to the parameter and the return value (For wrapper style)*
- *15.1.5 Mapping the message part to the parameter and the return value (For non-wrapper style)*
- *16.2.4 javax.jws.Oneway annotation*

For details on one-way operations and notes, see *10.22 One-way operations*.

(g) Functionality for asynchronous invocation

The JAX-WS functionality of Cosminexus does not support the functionality used for implementing asynchronous invocation of Web Service clients that are described in *Section 2.3.4* and *Chapter 4* of JAX-WS 2.2 specifications.

(h) Functionality related to Dispatch/ Provider interface

The JAX-WS functionality of Cosminexus supports the `Dispatch` interface described in *Chapter 4* of the JAX-WS 2.2 specifications, the `Provider` interface described in *Chapter 5* of the JAX-WS 2.2 specifications, and the functionality related to the `Dispatch` interface and `Provider` interface. However, among all the objects described in *Chapter 4* of the JAX-WS 2.2specifications, the JAX-WS functionality does not support the following objects:

- `javax.activation.DataSource`
- `javax.xml.transform.stax.StAXSource`

(i) Functionality related to Endpoint class and issuing

The JAX-WS functionality of Cosminexus does not support the functionality for dynamically creating and issuing the Web Service endpoints that are described in *Chapter 5* of JAX-WS 2.2 specifications.

(j) Functionality related to handler

The JAX-WS functionality of Cosminexus supports the API-based dynamic handler settings for implementing Web Service clients. The functionality also supports the annotation-based dynamic handler settings for implementing Web services. The static settings assuming JSR-109 specifications are not supported.

(k) Using the attachments

The JAX-WS functionality of Cosminexus supports the attachments of the SAAJ 1.3 specifications, attachments with the format for coding the `wsi:swaRef` format in a WSDL, and attachments in the MTOM/XOP specification format. The functionality does not support coding that uses the MIME extension element of the WSDL 1.1 specifications (MIME binding). For using attachments in the `wsi:swaRef` format, see *28. Attachment Functionality (wsi:swaRef format)*. For using attachments in the MTOM/XOP specification format, see *30. Attachment Functionality* (*MTOM/XOP)*.

(l) Message context

With the JAX-WS functionality of Cosminexus, the standard message context property described in *Chapter 9* of JAX-WS 2.2 specifications is read-only. The functionality supports the property used for specifying a timeout when implementing Web services. For the usage and precautions for the message context, see the section *19.2.5 Using a message context*.

(m) API

The JAX-WS functionality of Cosminexus supports the APIs of the JAX-WS 2.2 specifications. For the support range of the JAX-WS APIs, see the section *19.2 Support Range of APIs*.

(n) XML/ HTTP binding

The JAX-WS functionality of Cosminexus does not support the XML/ HTTP binding described in *Chapter 11* of the JAX-WS 2.2 specifications.

(o) Dynamically generating wrapper beans

For Web Services developed, starting from SEI, the JAX-WS functionality of Application Server supports the functionality of the JAX-WS engine (hereafter, also referred to as the *functionality to dynamically generate wrapper beans*) that dynamically generates JavaBeans classes of wrapper beans (request beans and response beans) and fault beans described in the section *3.6.2.1* and section *3.7* of the JAX-WS2.2 specifications. The Web service clients or Web Services that start from the WSDL do not support the functionality that dynamically generates wrapper beans.

For details on the functionality that dynamically generates wrapper beans, see *10.23 A functionality that dynamically generates wrapper beans*.

(p) Related standard specifications

The following points describe the related standard specifications:

- Functionality related to the MTOM specifications

  The JAX-WS functionality of Cosminexus supports the functionality related to the MTOM specifications described in *Sections 2.4* and *Section 6.5* of the JAX-WS 2.2 specifications.

- WS-Addressing specifications

  The JAX-WS functionality of Cosminexus supports the functionality related to the WS-Addressing specifications described in *Section 5.2.8* of the JAX-WS 2.2 specifications. For details, see *37. Addressing Functionality*.

- XML Catalogs 1.1 specifications

  The JAX-WS functionality of Cosminexus supports the functionality related to the XML Catalogs 1.1 specifications described in section *4.4* of the JAX-WS 2.2 specifications.

  For details on the support range of the XML Catalogs 1.1 specifications, see *21. Support Range of XML Catalogs 1.1*.

- WSEE (JSR-109) specifications

  The JAX-WS functionality of Cosminexus supports the WSEE (JSR-109) specifications. The following table describes the items and the support of the WSEE (JSR-109) specifications.

  Table 1–2: Items and support of WSEE (JSR-109) specifications

No.	Item		Support
1	The methods of Stateless session bean and singletone session bean are as follows: • Method of mapping or binding to WSDL • Method of mapping or binding to SOAP	Stateless	Y
		Singletone	--
2	Client model. A method for detecting service interface by using JNDI and a method of using `WebServiceRef` annotation of JAX-WS specifications	Contents of the section *4.2.2 (service class and port injection)*	Y
		Contents of sections other than *4.2.2*	--
3	Deployment model. A method of packaging to EAR file and life cycle.	Contents described in section 5.4	Y
		Contents other than those described in section 5.4	--

No.	Item	Support
4	Deployment descriptor. The contents that must be coded along with `WebServices.xml` syntax and mapping with annotation that is defined in JAX-WS specifications (Web services Metadata (JSR-181) specifications).	--
5	Mapping with the existing Java EE container functionality such as role	--

Legend:

Y: Indicates that WSEE specifications are supported.

--: Indicates that WSEE specifications are not supported.

Also supports operations when the `Web.xml` is omitted. For the operations when `Web.xml` is omitted, see the subsection *3.4 (3) Operations when Web.xml is not included in a WAR file*.

- SAAJ 1.3 specifications

  The JAX-WS functionality of Cosminexus supports the SAAJ 1.3 specifications. For the APIs of the SAAJ 1.3 specifications, see the section *22.1 Support range of the SAAJ 1.3 specifications*.

(2) Prerequisites for the functionalities and specifications of RESTful Web Services

This section describes the functionality and specifications you can use for developing RESTful Web Services (Web resource) with Cosminexus. For details on the support range of the standard specifications, see *24. Support range of the JAX-RS specifications*.

(a) Resource Class

A resource class is used for implementing Web resources. Chapter 3(8) of the JAX-RS 1.1 specifications defines resource classes. There are two types of resource class; root resource classes and sub resource classes.(8) The JAX-RS functionality of Cosminexus supports both the resource classes.

For details on root resource classes, see *17.1.1 Root resource class*. For details on sub resource classes, see *17.1. 7 Sub resource class*.

(b) Provider

A provider is an extended functionality for the JAX-RS engine. Chapter 4 of the JAX-RS 1.1 specifications defines Provider. There are three types of providers; entity providers, context providers, and exception mapping providers.

An entity provider maps an HTTP entity body and a Java type. The JAX-RS functionality supports a built-in entity provider. For details on the MIME types and Java types supported by a built-in entity provider, see *17.1.1(4)(c) Entity parameter*.

A context provider provides the context to a resource, or to the other resource providers. The JAX-RS functionality supports a built-in context provider. For details on the context supported by a built-in context provider, see *24.4 Context*.

An exception mapping provider customizes the mapping between the Web resource exceptions and HTTP responses. For details on the exception mapping provider, see *17.2.2 Exception mapping provider*.

(c) *Application*

The *Application* is a factory of the resource classes and providers. Chapter 2 of the JAX-RS 1.1 specifications defines the Application. The JAX-RS 1.1 specifications support a built-in *Application*. For details on the built-in *Application*, see *11.3.1 Configuration of WAR files*.

1.5 Format of Web services and Clients

This section describes the format of Web services and clients supported by Cosminexus.

1.5.1 Format of Web services

Cosminexus supports the following formats of Web services:

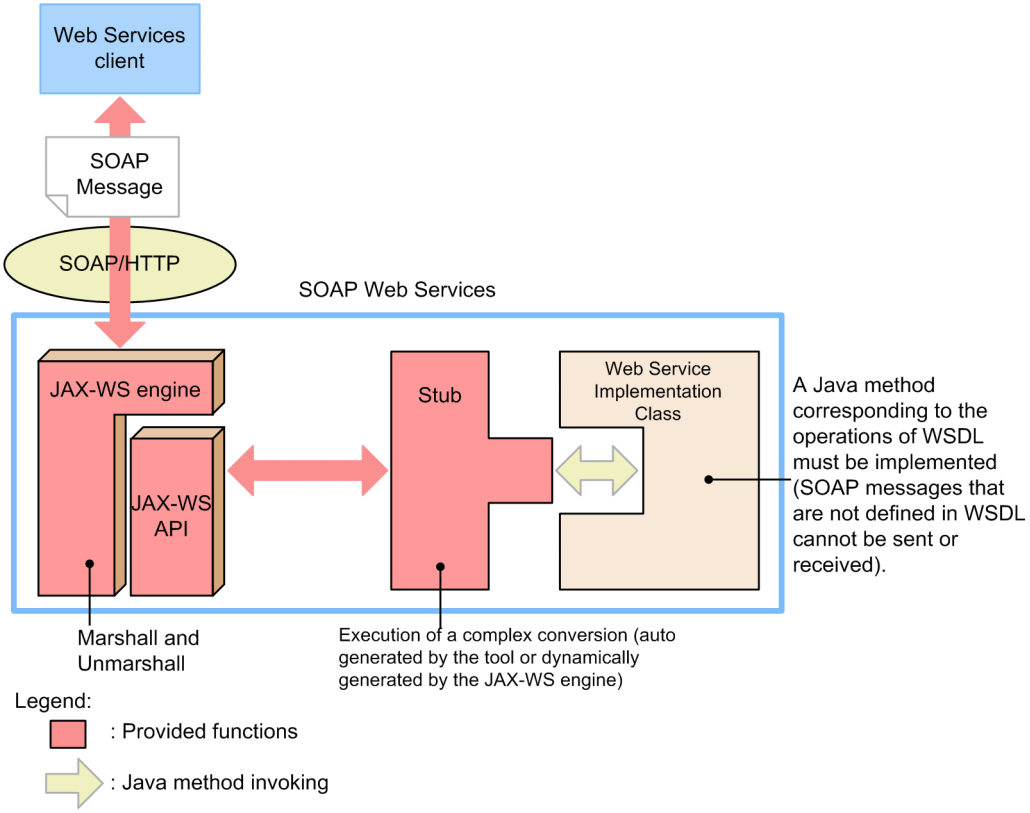
- SOAP Web services using Web Service Implementation Classes
- SOAP Web services using Provider Implementation Classes
- RESTful Web Services (Web resource)

The following subsections describe each of the above formats:

(1) SOAP Web services using Web Service Implementation Classes

The following figure shows the format of Web services using Web Service Implementation Classes:

Figure 1-5: Format of a SOAP Web service that uses a Web service implementation class



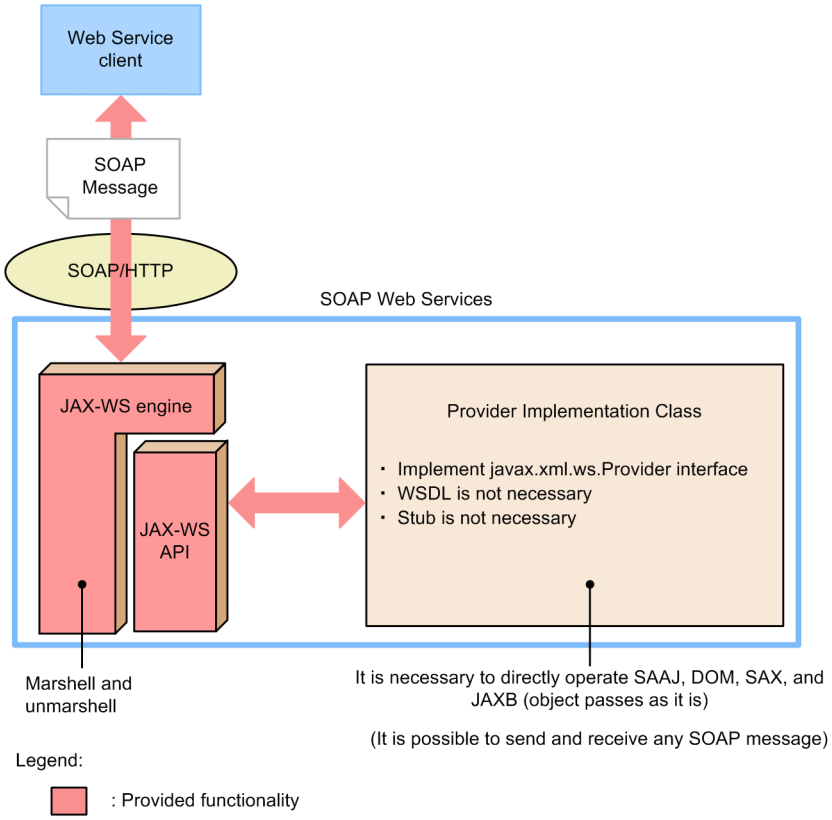
When using Web Service Implementation Classes, Web services can be realized only by implementing a Java method that corresponds with the operations of a WSDL. The automatically generated stubs execute complicated transformation, so for developing Web Services, you need not perform complicated programming such as using APIs to assemble XMLs that configure SOAP Messages.

Either the tool automatically generates the stubs or the JAX-WS engine dynamically generates the stubs. The WSDL is mandatory and can be automatically generated. Note that you will not be able to send and receive the SOAP Messages that are not included in the WSDL. You can create a Web service implementation class as a POJO and also as a class based on EJB.

(2) SOAP Web services using Provider Implementation Classes

The following figure shows the format of Web services using Provider Implementation Classes.

Figure 1-6: Format of a SOAP Web service that uses (1)Provider Implementation Classes

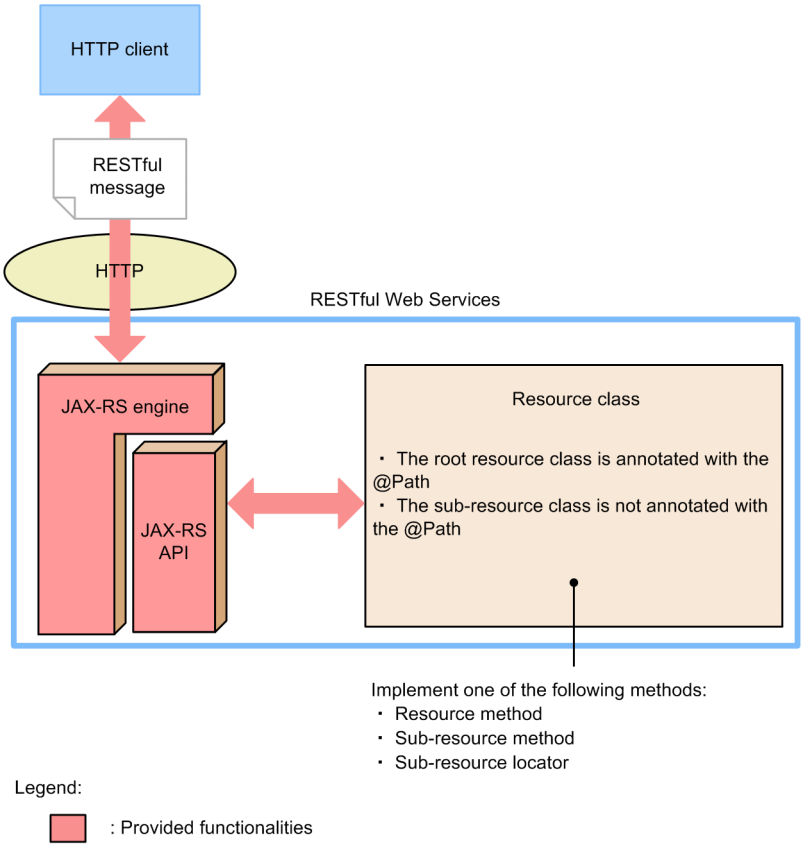


When using Provider Implementation Classes, you neither need stubs nor a WSDL, so Provider Implementation Classes are suitable for dynamically sending and receiving any SOAP Message. A Provider Implementation Class accepts an object marshaled by the JAX-WS engine as it is, and therefore for developing Web services, you must directly acquire values from XMLs that configure SOAP Messages using APIs, and then assemble the XMLs.

(3) RESTful Web Services (Web resource)

You can execute RESTful Web Services (Web resource) by implementing a resource class.

Figure 1-7: RESTful Web service (Web resource)



A resource class consists of a root resource class and sub resource class.

A root resource class is a Java class annotated with the `Path` annotation at the class level, whereas a sub-resource class is a Java class that is not annotated with the `Path` annotation at the class level.

You can implement any of the following methods in a resource class:

- Resource method
- Sub resource method
- Sub resource locator

1.5.2 Format of clients

Cosminexus supports the following forms of clients to call the SOAP Web services:

- Stub-based Web Service clients
- Dispatch-based Web Service clients

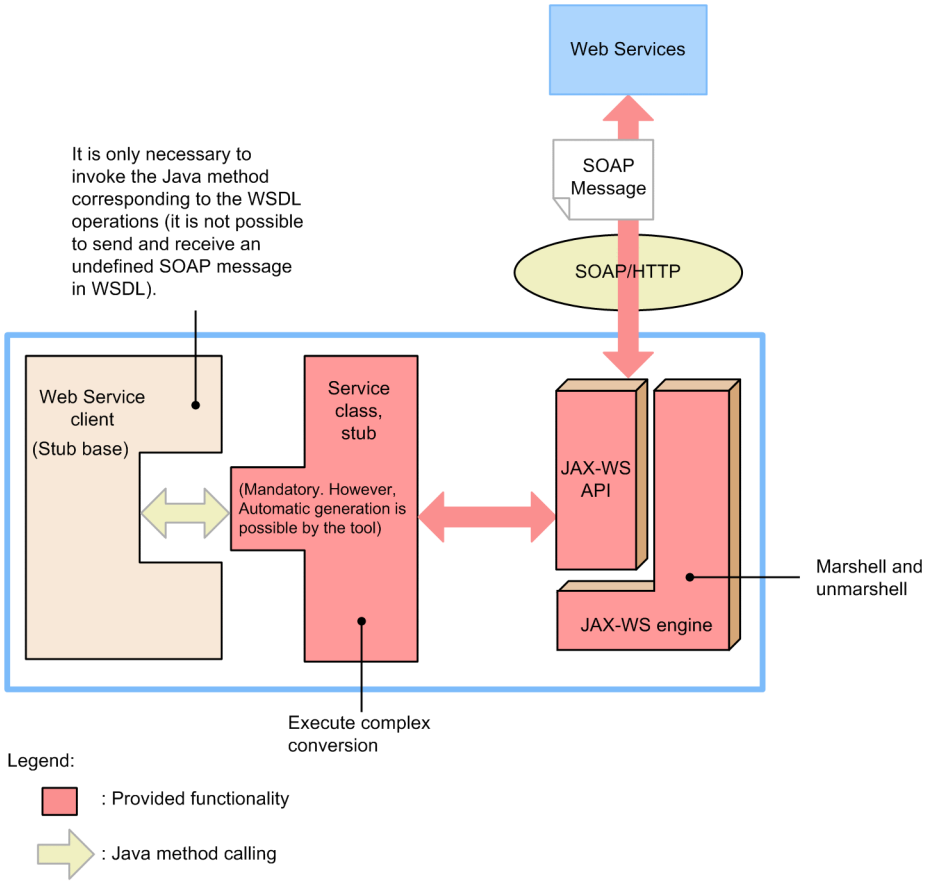
The following section describes each format.

Note that you must use either the client APIs for RESTful Web Services or standard Java APIs such as `java.net.URL` or `java.net.HttpURLConnection` to implement a Web resource client.

(1) Stub-based Web Service clients

The following figure shows the format using a stub-based Web Service client:

Figure 1-8: Web Service client (Stub-based)

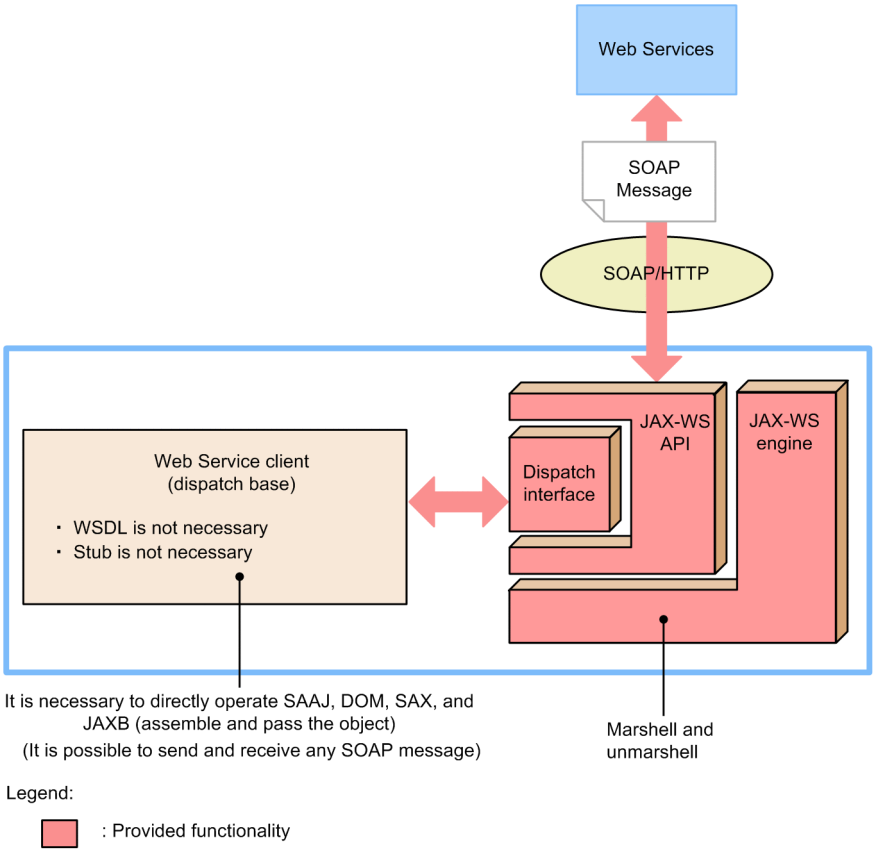


When using stub-based Web Service clients, Web services can be invoked only by invoking the Java method that corresponds with the operations of a WSDL. The automatically generated stubs execute complicated transformation, so for developing Web Service clients you need not perform complicated programming such as using APIs to assemble XMLs configuring SOAP Messages. However, note that it is not possible to send and receive SOAP Messages that are not defined in the WSDL. Also, the stubs and WSDL will be required. You must acquire a WSDL from the corresponding Web Service.

(2) Dispatch-based Web Service clients

The following figure shows the format using dispatch-based Web Service clients:

Figure 1-9: Web Service client (Dispatch-based)



When using dispatch-based Web Service clients, you neither need stubs nor a WSDL, so dispatch-based Web Service clients are suitable for dynamically sending and receiving any SOAP Message. A dispatch-based Web Service client transfers the generated objects to the JAX-WS engine via the `javax.xml.ws.Dispatch` interface, and therefore, when developing a Web Service client, you must directly acquire values from XMLs and use APIs to assemble the XMLs configuring SOAP Messages.

! Important note

With the dispatch-based Web Service client, you can also send and receive SOAP Messages using the `javax.xml.soap.SOAPConnection` class instead of the `javax.xml.ws.Dispatch` interface. However, note that when you use the `javax.xml.soap.SOAPConnection` class, the settings of operation definition files and the message context are disabled and a log is also not output.

Also, this manual does not describe operations for using the `javax.xml.soap.SOAPConnection` class. For details, see the *JDK documentation* as you would for the other APIs of the SAAJ 1.3 specifications. If there is no problem, use the `javax.xml.ws.Dispatch` interface instead of the `javax.xml.soap.SOAPConnection` class.

1.6 Setting up JAX-WS and JAX-RS engine

To use a JAX-WS engine and JAX-RS engine, you must specify definitions, for enabling the JAX-WS engine and JAX-RS engine, in the option definition file for a J2EE server.

For the definitions to be used for enabling the JAX-WS engine and JAX-RS engine, see the description of using the JAX-WS engine and JAX-RS engine in *Appendix A.1 (3) Switching the operating environment*.

2 Procedures for Development

You develop JAX-WS 2.2 compliant SOAP Web Services based on a WSDL, SEI, or provider. The client that calls SOAP Web Services is developed either on a stub base or a dispatch base.

You develop RESTful Web Services (Web resources) by implementing a resource class. You develop a client that calls RESTful Web Services by using a client API for RESTful Web resources or the standard Java API.

This chapter describes the development procedure of Web Services and Web Service clients.

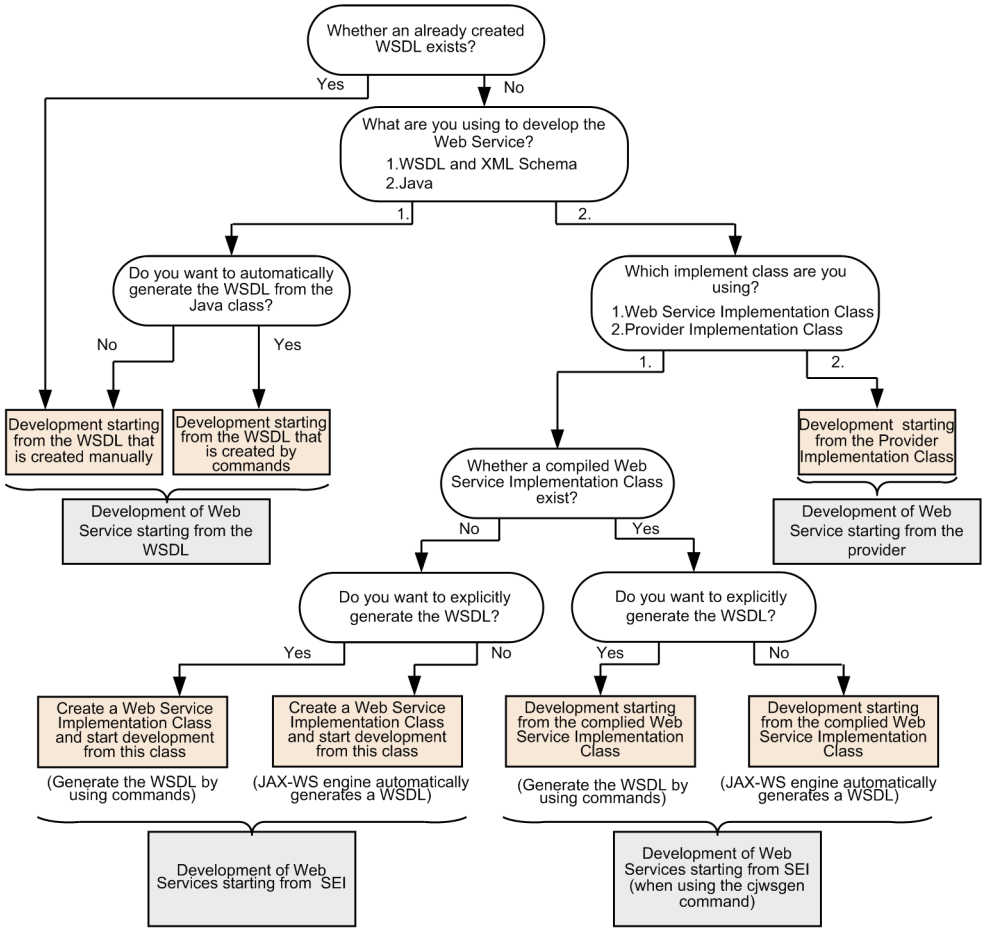
2.1 Development flow of SOAP Web Services

Use the following methods for developing SOAP Web Services:

- Development starting from a WSDL (*2.1.1*)
- Development starting from SEI (*2.1.2*)
- Development starting from SEI (when using the `cjwsgen` command) (*2.1.3*)
- Development starting from a provider (*2.1.4*)

The following figure shows how to select a development method:

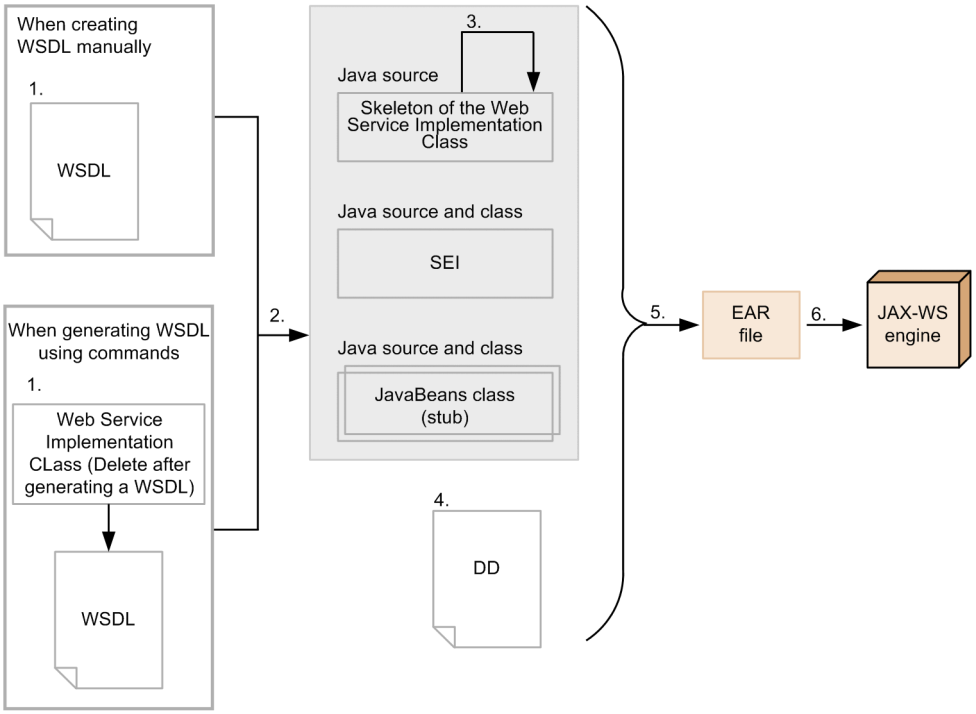
Figure 2-1: How to select a method for developing SOAP Web Services



2.1.1 Development starting from WSDL

The following figure shows the flow of the development starting from the WSDL:

Figure 2-2: Development of Web Services starting from the WSDL



1. Creating a WSDL file

   You can create a WSDL file either manually or with commands.

   - Creating manually

     As the meta data of Web Services, create a WSDL file according to the WSDL 1.1 specifications, XML Schema specifications, and WS-I Basic Profile 1.1. For receiving messages of SOAP 1.1, code extension elements of the SOAP 1.1 specifications and for receiving messages of SOAP 1.2, code extension elements of the SOAP 1.2 specifications.

     For the support range of the WSDL 1.1 specifications, see *20.1 Support range of the WSDL 1.1 specifications*.

   - Generating with commands

     We recommend the generation of a WSDL using commands, when a user is accustomed of developing with Java language rather than with syntaxes of a WSDL and XML Schema.

     After temporarily creating or compiling a Web Service Implementation Class, specify the `-wsdl` option and execute the WSDL generation functionality of the `cjwsgen` command to generate a WSDL file. Change the generated WSDL, as and when required. Use the generated Web Service Implementation Class only for entering the `cjwsgen` command. Therefore, you need not execute any method.

     For receiving the messages of the SOAP 1.2 specifications, specify `http://www.w3.org/2003/05/soap/bindings/HTTP/` in the `javax.xml.ws.BindingType` annotation while creating the Web Service Implementation Class.

     After generating the WSDL, delete the Web Service Implementation Class that is not required anymore.

2. Executing the cjwsimport command (Generating Java sources)

   Execute the `cjwsimport` command to generate Java sources such as SEI, skeleton of the Web Service Implementation Class, and JavaBeans class (stubs) that are required for developing and executing Web Services, from the created WSDL file. Specify the `-generateService` option and execute the `cjwsimport` command. For the `cjwsimport` command, see *14.1 cjwsimport command*.

3. Implementing Web Services

   Using the stubs generated in step 2, code the required processing in the skeleton of the Web Service Implementation Class and implement the Web Services. Also, compile the implemented Web Service Implementation Class. Note that the `javax.xml.ws.BindingType` annotation is automatically added according to the contents of the WSDL.

4. Creating a DD

Create `web.xml` and `application.xml`. Code the Web Service-specific information in `web.xml`. For creating `web.xml`, see *3.4 Creating web.xml*.

5. Creating an EAR file
   Create an EAR file containing the created file. For creating EAR files, see *3.5.3 Creating EAR files*.
6. Deploying and starting the EAR file
   Deploy the created EAR file and start the file as a J2EE application (Web Service). For details on the `import` and `start` commands of J2EE applications, see *cjimportapp (Importing J2EE applications) and cjstartapp (Starting J2EE applications)* in the *uCosminexus Application Server Command Reference Guide*.
   For the method to deploy (import) J2EE applications by using the management portal, see *12.3.3 Importing J2EE applications* in the *uCosminexus Application Server Management Portal User Guide*.
   For the method to start J2EE applications by using the management portal, see *12.3.1 Starting the J2EE applications* in the *uCosminexus Application Server Management Portal User Guide*.

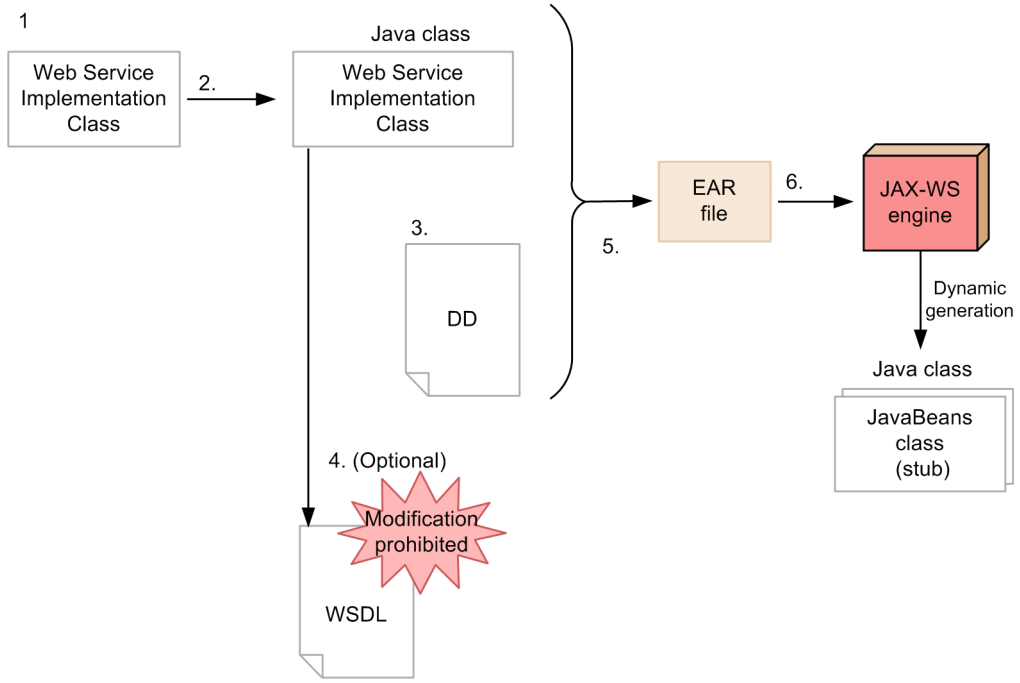
For examples in which the development of Web Services starts from the WSDL, see the following sections:

- *4.3 Examples of the Web Service development (Starting from WSDL)*
- *35.3 Examples of Web Service development (Starting from WSDL/WS-RM 1.2)*

2.1.2 Development starting from SEI

The following figure shows the flow of the development starting from SEI:

Figure 2-3: Development of Web Services starting from SEI



Although the JAX-WS engine automatically generates WSDLs, you can also generate WSDLs by executing the commands according to step 4. If you are not generating a WSDL with the commands, step 4 will not be required.

1. Creating a Web Service Implementation Class
   There are cases when a Web Service Implementation Class is created as POJO or created based on EJB. In both the cases, create a Web Service Implementation Class according to the JAX-WS 2.2 specifications and JAXB 2.2 specifications. For receiving messages of the SOAP 1.2 specifications, specify `http://www.w3.org/2003/05/soap/bindings/HTTP/` in the `javax.xml.ws.BindingType` annotation.

For the support range of the JAX-WS 2.2 specifications, see *19.1 Support range of the JAX-WS 2.2 specifications*. For the support range of the JAXB 2.2 specifications, see *Appendix B Support Range of the JAXB Specifications* in the *uCosminexus Application Server XML Processor User Guide*.

2. Compiling Web Services Implementation Classes

   Execute the `javac` command to compile the created Web Services Implementation Classes. For details on the `javac` command, see the *JDK documentation*.

3. Creating a DD

   Create `web.xml` and `application.xml`. Code the Web Service-specific information in `web.xml`. For creating `web.xml`, see *3.4 Creating web.xml*.

4. Executing the WSDL generation functionality of the cjwsgen command (optional)

   Specify the `-wsdl` option to execute the `cjwsgen` command, generate a WSDL from Web Services Implementation Class, and check errors such as the annotation errors. To ensure that no errors occur when dynamically generating the `JavaBeans` class (stub) mentioned in step 6, execute the `cjwsgen` command for the compiled Web Services Implementation Classes. This step is optional. Do not change the WSDL that is generated in this step.

   To deploy the WSDL generated with the WSDL generation functionality of the `cjwsgen` command, you can develop any method other than the meta data issuing functionality, such as sending through mail.

   For the `cjwsgen` command, see *14.3 cjwsgen command*.

5. Creating an EAR file

   Create an EAR file containing the created file. For creating EAR files, see *3.5.3 Creating EAR files*.

6. Deploying and starting the EAR file

   Deploy the created EAR file and start the file as a J2EE application (Web Service). For details on the `import` and `start` commands of J2EE applications, see *cjimportapp (Importing J2EE applications) and cjstartapp (Starting J2EE applications)* in the *uCosminexus Application Server Command Reference Guide*.

   For the method to deploy (import) J2EE applications by using the management portal, see *12.3.3 Importing J2EE applications* in the *uCosminexus Application Server Management Portal User Guide*.

   For the method to start J2EE applications by using the management portal, see *12.3.1 Starting J2EE applications* in the *uCosminexus Application Server Management Portal User Guide*.

   The JAX-WS engine dynamically generates the `JavaBeans` class (stub) when you start a J2EE application (Web Services). Also, to ensure that no error occurs when dynamically generating the JavaBeans class (stub), if you execute commands for the compiled Web Services Implementation Classes as mentioned in step 4 of this section, you can check any errors such as annotation errors in advance. See *10.23(1) Using the cjwsgen command to check errors*.

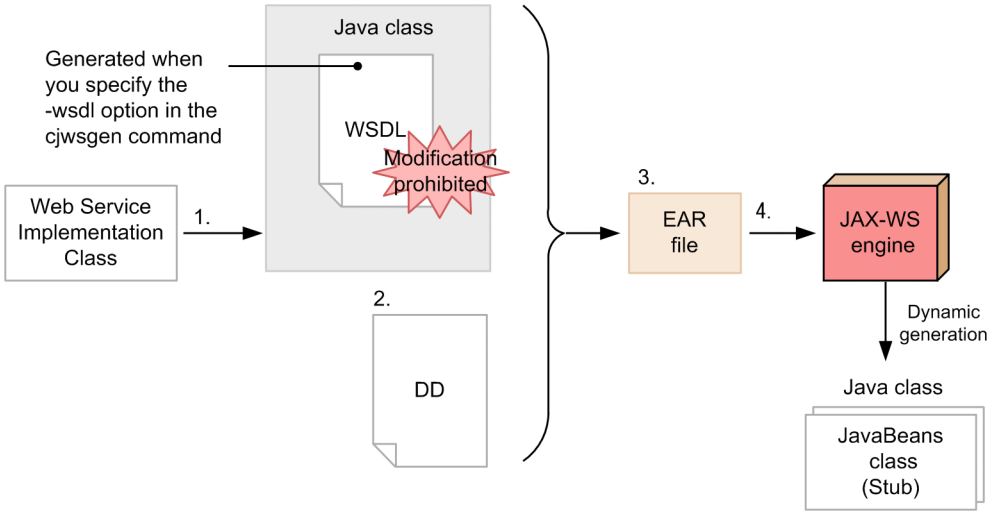
For examples of the development of Web Services starting from SEI, see the following sections:

- *5.3 Examples of Web Service development (Starting from SEI)*
- *6.3 Examples of Web Service development (Starting from SEI or cjwsgen command)*
- *7.3 Examples for developing Web Services (Starting from SEI or customization)*
- *8.3 Examples of Web Service development (Starting from SEI/EJB Web Service)*
- *29.3 Examples of Web Service development (Starting from SEI and attachments of wsi:swaref format)*
- *33.3 Examples of Web Service development (Starting from SEI and streaming)*
- *38.3 Examples of Web Service development (Starting from SEI and addressing)*

2.1.3 Development starting from SEI (When using the cjwsgen command)

The following figure shows how to develop Web Services starting from SEI, when you generate Java sources with the `cjwsgen` command from a compiled Web Service Implementation Class:

Figure 2-4: Developing Web Services starting from SEI (When using the cjwsgen command)



1. Executing the `cjwsgen` command

   There are cases when a Web Services implementation class is created as POJO or created based on EJB. In both the cases, if you specify the `-wsdl` option when executing the `cjwsgen` command, you can generate a WSDL or check annotation errors. In such cases, do not change the generated WSDL.

   For the `cjwsgen` command, see *14.3 cjwsgen command*.

2. Creating a DD

   Create `web.xml` and `application.xml`. Code the Web Service-specific information in `web.xml`. For creating `web.xml`, see *3.4 Creating web.xml*.

3. Creating an EAR file

   Create an EAR file containing the created file. For creating EAR files, see *3.5.3 Creating EAR files*.

4. Deploying and starting the EAR file

   Deploy the created EAR file and start the file as a J2EE application (Web Services). For details on the `import` and `start` commands of J2EE applications, see *cjimportapp (Importing J2EE applications) and cjstartapp (Starting J2EE applications)* in the *uCosminexus Application Server Command Reference Guide*.

   For the method to deploy (import) J2EE applications by using the management portal, see *12.3.3 Importing J2EE applications*in the *uCosminexus Application Server Management Portal User Guide*.

   For the method to start J2EE applications by using the management portal, see *12.3.1 Starting J2EE applications* in the *uCosminexus Application Server Management Portal User Guide*.

   The JAX-WS engine dynamically generates the `JavaBeans` class (stub) when you start a J2EE application (Web Services).

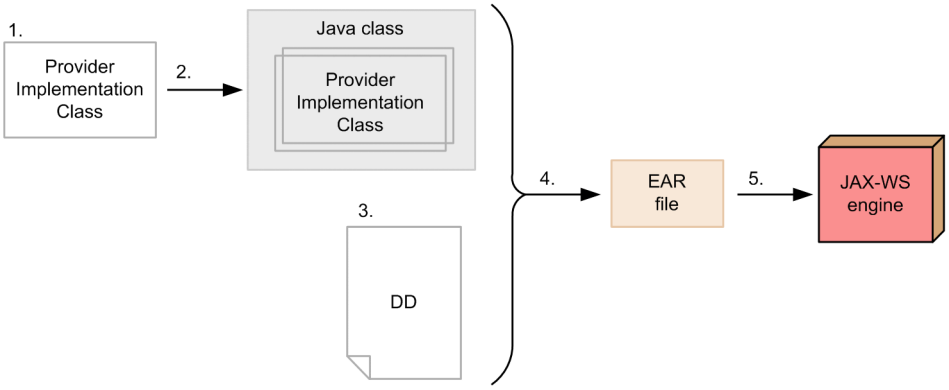
For the examples of developing the Web Services starting from SEI, see the following sections:

- *5.3 Examples of Web Service development (Starting from SEI)*
- *6.3 Examples of Web Service development (Starting from SEI or cjwsgen command)*
- *7.3 Examples for developing Web Services (Starting from SEI or customization)*
- *8.3 Examples of Web Service development (Starting from SEI/EJB Web Service)*
- *29.3 Examples of Web Service development (Starting from SEI and attachments of wsi:swaref format)*
- *33.3 Examples of Web Service development (Starting from SEI and streaming)*
- *38.3 Examples of Web Service development (Starting from SEI and addressing)*

2.1.4 Development starting from a provider

The following figure shows how to develop Web Services starting from a provider:

Figure 2-5: Developing Web Services starting from a provider



1. Creating a Provider Implementation Class

   Create the Web Service Implementation Class as POJO.

   Create Provider Implementation Classes according to the JAX-WS 2.2 and JAXB 2.2 specifications. For receiving messages of the SOAP 1.2 specifications, specify `http://www.w3.org/2003/05/soap/bindings/HTTP/` in the `javax.xml.ws.BindingType` annotation.

   For the support range of the JAX-WS 2.2specifications, see *19.1 Support range of the JAX-WS 2.2 specifications*. For the support range of the JAXB 2.2 specifications, see *Appendix B Support Range of the JAXB Specifications* in the *uCosminexus Application Server XML Processor User Guide*.

2. Executing the apt command (Compiling and error checking)

   Execute the `apt` command for compiling the created Provider Implementation Classes and for error checking.

3. Creating a DD

   Create `web.xml` and `application.xml`. Code the Web Service-specific information in `web.xml`. For creating `web.xml`, see *3.4 Creating web.xml*.

4. Creating an EAR file

   Create an EAR file containing the created file. For creating EAR files, see *3.5.3 Creating EAR files*.

5. Deploying and starting the EAR file

   Deploy the created EAR file and start the file as a J2EE application (Web Services). For details on the `import` and `start` commands of J2EE applications, see *cjimportapp (Importing J2EE applications) and cjstartapp (Starting J2EE applications)* in the *uCosminexus Application Server Command Reference Guide*.

   For the method to deploy (import) J2EE applications by using the management portal, see *12.3.3 Importing J2EE applications* in the *uCosminexus Application Server Management Portal User Guide*.

   For the method to start J2EE applications by using the management portal, see *12.3.1 Starting J2EE applications* in the *uCosminexus Application Server Management Portal User Guide*.

For the examples of developing Web Services starting from a provider, see *9.3 Examples of developing Web Services (Starting from a provider and SAAJ)*.

2.2 Procedure of developing Web Service clients

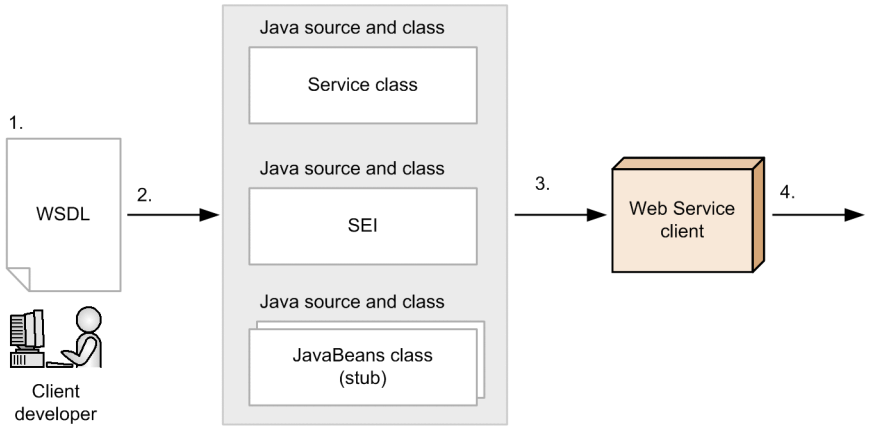
Use the following methods to develop clients that call SOAP Web Services:

- Developing stub-based Web Service clients (*2.2.1*)
- Developing dispatch-based Web Service clients (*2.2.2*)

2.2.1 Developing stub-based Web Service clients

The following figure shows how to develop stub-based Web Service clients:

Figure 2-6: Procedure of developing a Web Service client (Stub-based)



1. Acquiring a WSDL file (Or acquiring a public WSDL URL)
   Acquire a WSDL file coding the meta data for the Web Service that you are trying to invoke, or if the Web Service that you are trying to invoke has the WSDL file URL public, acquire the URL.
2. Executing the cjwsimport command (Generating Java sources)
   Execute the `cjwsimport` command to generate Java sources, such as service class and SEI that are required for developing and executing the Web Service client, from the acquired WSDL file or the acquired WSDL file URL. Execute the `cjwsimport` command without specifying the `-generateService` option. For the `cjwsimport` command, see *14.1 cjwsimport command*.
3. Implementing a Web Service client
   Use the generated Java source to implement a Web Service client. Compile the implemented Web Service client using the `javac` command. For implementing Web Service clients, see *3.6 Implementing Web Service clients*.
4. Invoking a Web Service
   Execute the created Web Service client to invoke a Web Service.

For examples of the development of stub-based Web Service clients starting from the WSDL, see *4.5 Examples for deploying Web Service clients (Starting from WSDL)*.

For the examples of developing stub-based Web Service clients starting from SEI, see the following sections:

- *5.5 Examples of developing Web Service clients (Starting from SEI)*
- *6.5 Examples of Web Service client development (Starting from SEI or cjwsgen command)*
- *7.5 Examples of developing Web Service clients (Starting from SEI or customization)*
- *8.5 Examples of Web Service client development (Starting from SEI/EJB Web Service*)
- *29.5 Examples of Web Service client development (Starting from SEI and attachments of wsi:swaRef format)*
- *33.3 Examples of Web Service development (Starting from SEI and streaming)*
- *38.5 Examples of Web Service client development (Starting from SEI and addressing)*

2.2.2 Developing dispatch-based Web Service clients

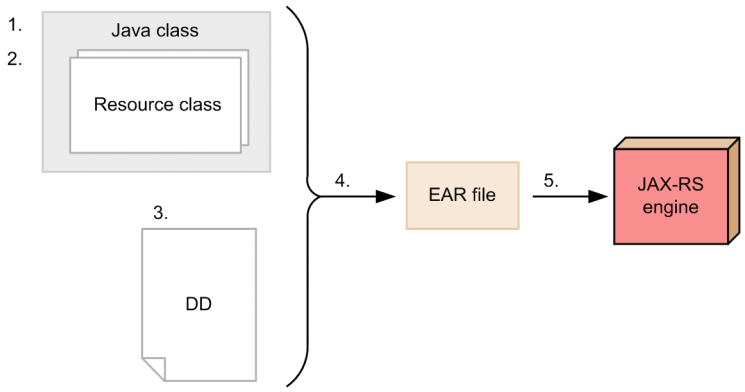
For developing dispatch-based Web Service clients, implement Web Service clients according to the JAX-WS 2.2 and JAXB 2.2 specifications.

For the examples of developing dispatch-based Web Service clients, see *9.5 Examples of Web Service client development (Starting from a provider and SAAJ)*.

2.3 Development flow of RESTful Web Services

The following figure shows the development flow of RESTful Web Services (Web Services).

Figure 2-7: Development flow of RESTful Web Services



1. Creating a root resource class

   Create a root resource class and implement at least one from among the resource method, sub-resource method, or the sub-resource locator. Also create a sub-resource class or an exception mapping provider as and when required.

2. Executing the `javac` command

   Execute the `javac` command to compile the created Java source.

3. Creating a DD

   Create `web.xml` and `application.xml`. Code the information specific to Web Services in `web.xml`. For details on creating `web.xml`, see *11.2 Creating web.xml*.

4. Creating EAR files

   Create an EAR file that includes the created file. For creating EAR files, see *11.3.2 Creating EAR files*.

5. Deploying and starting EAR files

   Deploy the created EAR file and start the file as a J2EE application (Web Service). For details on the `import` and `start` commands of J2EE applications, see *cjimportapp (Importing J2EE applications) and cjstartapp (Starting J2EE applications)* in the *uCosminexus Application Server Command Reference Guide*.

   For the method to deploy (import) J2EE applications by using the management portal, see *12.3.3 Importing J2EE applications* in the *uCosminexus Application Server Management Portal User Guide*.

   For the method to start J2EE applications by using the management portal, see *12.3.1 Starting J2EE applications* in the *uCosminexus Application Server Management Portal User Guide*.

For examples of developing RESTful Web Services, see *12.3 Examples of developing Web resources*.

3 Points on developing SOAP Web Services

This chapter describes the points you must understand in advance and the precautions you must take for each operation of developing Web Services.

3.1 Creating WSDL

For the development starting from WSDL, create a WSDL according to WSDL 1.1 specifications and WS-I Basic Profile 1.1. For the support range of the WSDL 1.1 specifications, see *20.1 Support range of the WSDL 1.1 specifications*.

For POJO Web Services, store the created WSDL in the wsdl directory configuring WAR files. For the WSDL storage destination, see *3.5.1 Configuring WAR files*.

For EJB Web Services, store the created WSDL in the wsdl directory configuring EJB JAR files. For the WSDL storage destination, see *3.5.2 Configuring EJB JAR files*.

If WSDL is not included in the wsdl directory, a WSDL will be automatically generated according to the JAX-WS 2.2 specifications using the JAX-WS engine of the Web Service machine, when deploying the Web Service.

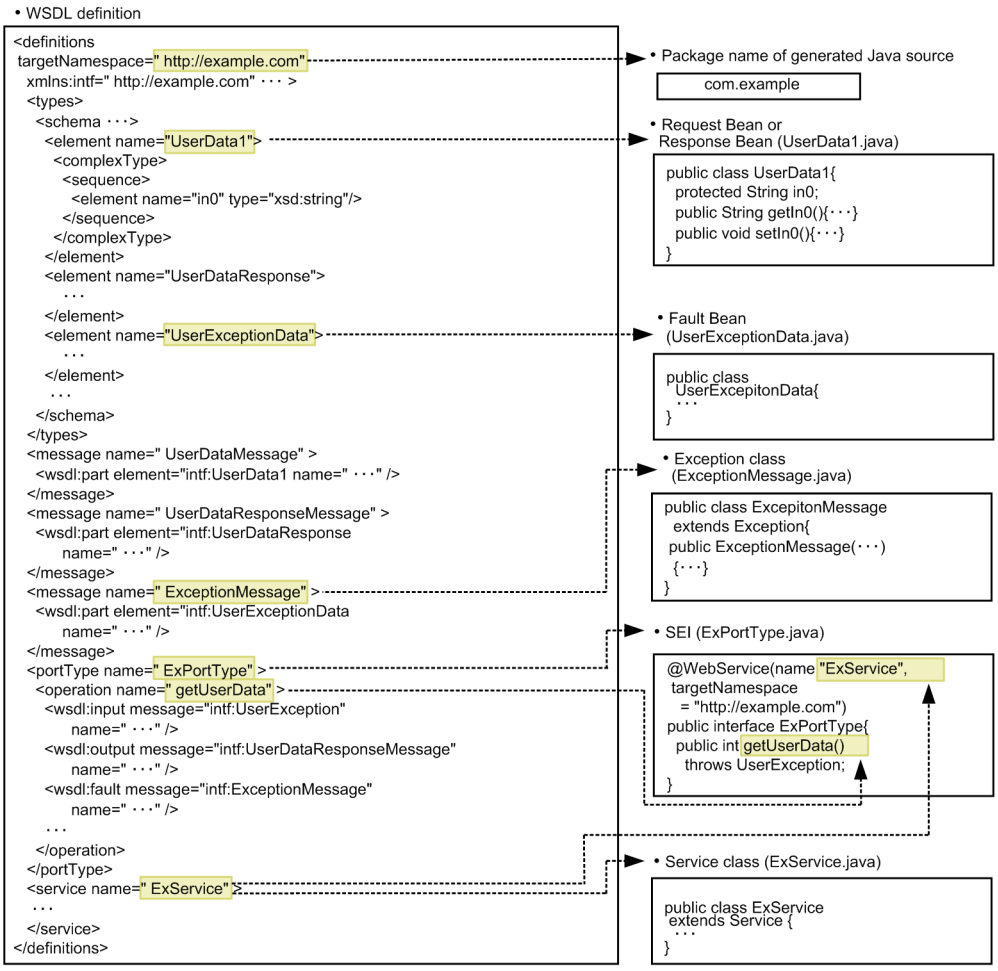
3.2 Mapping between WSDL and Java sources

When executing the `cjwsimport` command and the `apt` command, a WSDL and a Java source will be mapped. This section describes the examples of mapping between a WSDL and a Java source (default mapping).

3.2.1 Examples of mapping WSDL to Java sources

When executing the `cjwsimport` command, a WSDL will be mapped to a Java source. The following figure shows an example of mapping a WSDL to a Java source:

Figure 3-1: Example of mapping WSDL to Java sources

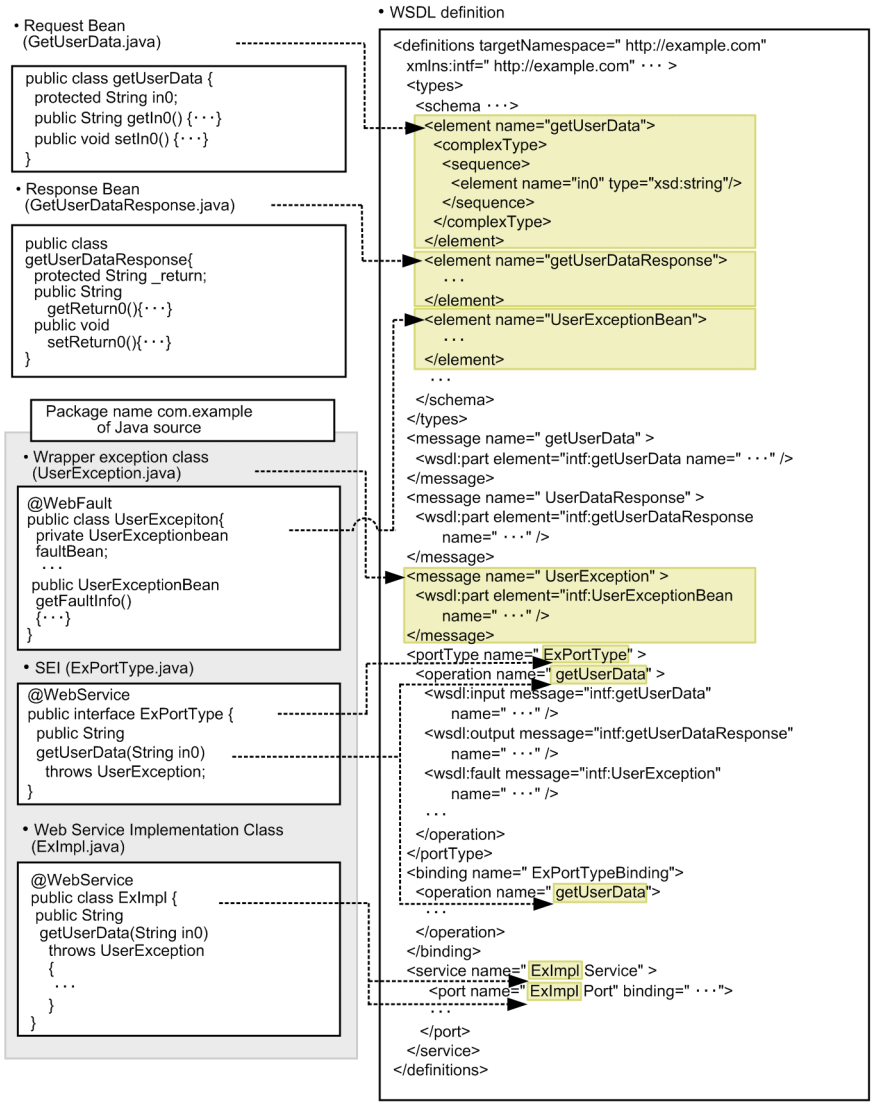


For mapping, see *15. Mapping WSDL to Java*.

3.2.2 Examples of mapping Java sources to WSDL

When executing the `apt` command, a Java source will be mapped to a WSDL. The following figure shows an example of mapping Java sources to the WSDL:

Figure 3-2: Example of mapping Java sources to WSDL



For mapping, see *16. Mapping Java to WSDL*.

3.3 Creating Web Service Implementation Classes and Provider Implementation Classes

Create a Web Service Implementation Class or a Provider Implementation Class as a compiled Java class file (`*.class`).

(1) In POJO Web Service

Web Service Implementation Class or Provider Implementation Class is included in the directory configuring WAR files. Include one or more Web Service Implementation Classes in one or both of the following:

- Below the `classes` directory
- In the JAR file included below the `lib` directory

For the storage destination of the Web Service Implementation Class or the Provider Implementation Class, see *3.5.1 Configuring WAR files*.

(2) In EJB Web Service

Web Service Implementation Class is included in the directory configuring the EJB JAR files. Include one or more classes in the following directory:

- Below the classes directory

For the storage destination of the Web Service Implementation Class, see *3.5.2 Configuring EJB JAR files*.

3.4 Creating web.xml

This section describes the `web.xml` included in the WAR file to be used in POJO Web Service.

When creating `web.xml`, specify the file name as `web.xml` and save directly under the `WEB-INF` directory configuring WAR files. The requirement of saving the `web.xml` file differs as per a value specified in the `webserver.container.jaxws.webservice.no_webxml.enabled` property of the user property file (`usrconf.properties`) that is used for J2EE servers.

- **When specifying "strict" or "true" (we recommend that you specify "strict")**
  It is optional for you to save the `web.xml` file with the name `web.xml` directly under the `WEB-INF` directory. If saving the `web.xml` file, the definitions required for executing Web Services must be coded.
- **When specifying "lax"**
  It is optional for you to save the `web.xml` file with the name `web.xml` directly under the `WEB-INF` directory. If saving the `web.xml` file, you are not required to code the definitions for executing Web Services.
- **When specifying "none" or "false" (we recommend that you specify "none")**
  Always save the `web.xml` file with the name `web.xml` directly under the `WEB-INF` directory.

The following are the definitions required to execute Web Services, examples of `web.xml`, and operations when `web.xml` is not included in a WAR file:

(1) Definitions required for executing Web Services

For specifying `strict` in the `webserver.container.jaxws.webservice.no_webxml.enabled` property to include `web.xml` in the WAR file, or for specifying `none`, create `web.xml` in such a way so that the following conditions are fulfilled:

- **Version**
  The `web.xml` version must be 2.5 or later.
- **Listener**
  Include the following *listener* elements in the `web-app` element:

```
<listener>
 <listener-class>
 com.cosminexus.xml.ws.transport.http.servlet.WSServletContextListener
 </listener-class>
</listener>
```

- **Servlet**
  Include the following servlet elements in the `web-app` element:

```
<servlet>
 <description>Endpoint servlet for Cosminexus JAX-WS</description>
 <display-name>Endpoint_servlet_for_Cosminexus_JAX_WS</display-name>
 <servlet-name>CosminexusJaxwsServlet</servlet-name>
 <servlet-class>
 com.cosminexus.xml.ws.transport.http.servlet.WSServlet
 </servlet-class>
</servlet>
```

- **Servlet mapping**
  Code the `servlet-mapping` element under the `web-app` element and include the same number of `url-pattern` elements as the number of Web Service Implementation Classes or Provider Implementation Classes.
  The following is a *coding example of servlet-mapping element*:

```
<servlet-mapping>
 <servlet-name>CosminexusJaxwsServlet</servlet-name>
 <url-pattern>"/"+Service name of Web Service 1</url-pattern>
 <url-pattern>"/"+Service name of Web Service 2</url-pattern>
 ...
 <url-pattern>"/"+Service name of Web Service n</url-pattern>
</servlet-mapping>
```

For "/" + Service name of Web Service 1 in the url-pattern element, code a string with / (slash) added as a prefix to the following values:

- Value of serviceName attribute of the javax.jws.WebService annotation of the Web Service Implementation Class
- Values of the serviceName attribute of the javax.xml.ws.WebServiceProvider annotation for the Provider Implementation Class

Reference note

To include cosminexus-jaxws.xml in a WAR file

Code a value for the url-pattern attribute of the endpoint element corresponding to the Web Service Implementation Class or the Provider Implementation Class of cosminexus-jaxws.xml. For cosminexus-jaxws.xml, see *10.3 Customization using cosminexus-jaxws.xml*.

- **Other elements**

  You can code any other element. To include the servlet, listener, and JSP created in the WAR file, specify the definition in web.xml appropriately.

(2) Examples of web.xml

An example of web.xml is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/
javaee/web-app_3_0.xsd">
 <description>Sample web service &amp;quot;fromwsdl&amp;quot;</description>
 <display-name>Sample_web_service_fromwsdl</display-name>
 <listener>
 <listener-class>
 com.cosminexus.xml.ws.transport.http.servlet.WSServletContextListener
 </listener-class>
 </listener>
 <servlet>
 <description>Endpoint servlet for Cosminexus JAX-WS</description>
 <display-name>Endpoint_servlet_for_Cosminexus_JAX_WS</display-name>
 <servlet-name>CosminexusJaxwsServlet</servlet-name>
 <servlet-class>
 com.cosminexus.xml.ws.transport.http.servlet.WSServlet
 </servlet-class>
 <load-on-startup>1</load-on-startup>
 </servlet>
 <servlet-mapping>
 <servlet-name>CosminexusJaxwsServlet</servlet-name>
 <url-pattern>/TestJaxWsService</url-pattern>
 </servlet-mapping>
 <session-config>
 <session-timeout>60</session-timeout>
 </session-config>
</web-app>
```

When creating web.xml of version 2.5, specify 2.5 in the version attribute of the web-app element and specify http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd as the second location information in the xsd:schemaLocation attribute.

The above example is based on the assumption that the value of the following attributes is TestJaxWsService:

- The serviceName attribute of the javax.jws.WebService annotation for the Web Service Implementation Class
- The serviceName attribute of the javax.xml.ws.WebServiceProvider annotation for the Provider Implementation Class

At this time, add / (slash) and code the value of the url-pattern element as /TestJaxWsService.

Reference note

To include cosminexus-jaxws.xml in a WAR file

If a value for the `url-pattern` attribute of the `endpoint` element that corresponds to the Web Service Implementation Class or the Provider Implementation Class is `/TestJaxWsService`, specify the coding as described in the example.

(3) Operations when web.xml is not included in a WAR file

With the JAX-WS functionality of Cosminexus, if you do not specify `strict` or `lax` in the `webserver.container.jaxws.webservice.no_webxml.enabled` property to include `web.xml` in the WAR file, the processing will be performed considering that the `web.xml` with the following contents exist while invoking Web Services:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- code the web-app element-->
 <description>Cosminexus JAX-WS Default web.xml</description>
 <display-name>Cosminexus_JAX_WS_Default_web_xml</display-name>
 <listener>
 <listener-class>
 com.cosminexus.xml.ws.transport.http.servlet.WSServletContextListener
 </listener-class>
 </listener>
 <servlet>
 <description>Endpoint servlet for Cosminexus JAX-WS</description>
 <display-name>Endpoint_servlet_for_Cosminexus_JAX_WS</display-name>
 <servlet-name>CosminexusJaxwsServlet</servlet-name>
 <servlet-class>
 com.cosminexus.xml.ws.transport.http.servlet.WSServlet
 </servlet-class>
 <load-on-startup>1</load-on-startup>
 </servlet>
 <servlet-mapping>
 <servlet-name>CosminexusJaxwsServlet</servlet-name>
 <url-pattern>"/"+Service name of Web Service 1</url-pattern>
 <url-pattern>"/"+Service name of Web Service 2</url-pattern>
 :
 <url-pattern>"/"+Service name of Web Service n</url-pattern>
 </servlet-mapping>
 <session-config>
 <session-timeout>60</session-timeout>
 </session-config>
</web-app>
```

For `"/" + Service name of Web Service 1`, a string is defined with / (slash) that is added as a prefix to values of the following attributes:

- A value of the `serviceName` attribute of the `javax.jws.WebService` annotation for the Web Service Implementation Class.
- A value of the `serviceName` attribute of the `javax.xml.ws.WebServiceProvider` annotation for the Provider Implementation Class

The operations for all the Web Service Implementation Classes or the Provider Implementation Classes stored in the WAR file will be performed assuming that the `url-pattern` element exists.

When you specify `lax` in the `webserver.container.jaxws.webservice.no_webxml.enabled` property to include `web.xml` in a WAR file, and if the contents of the section *3.4(1) Definitions required for executing Web Services* are not coded correctly in `web.xml`, invoke Web Services considering that the `web.xml` with the above contents exist, in the same way as you would consider when `web.xml` is not included. Note that the `web.xml` file that is considered in such cases will have the above mentioned contents excluding the contents of the `session-config` element.

! Important note

- The `web.xml` file that the JAX-WS engine considers need not be actually generated within the WAR file. This assumption is only applicable when invoking Web Services.

Example:

The attribute files that can be acquired with the `cjgetappprop` command do not include the information related to `web.xml`. Also, when specifying `lax` in the `webserver.container.jaxws.webservice.no_webxml.enabled` property and including `web.xml` with invalid contents in a WAR file, the information that is actually included in `web.xml` can only be acquired.

- When specifying `lax` in the `webserver.container.jaxws.webservice.no_webxml.enabled` property and including `web.xml` with invalid contents in a WAR file, the contents of the `web.xml` file that the JAX-WS engine considers while invoking the Web Services are not necessarily be coded in the actual `web.xml`.
- When specifying `lax` in the `webserver.container.jaxws.webservice.no_webxml.enabled` property, include `web.xml` defining all the contents of the section *3.4(1) Definitions required for executing Web Services* in a WAR file. If `web.xml` with only partial contents is included in the WAR file, the operation cannot be guaranteed.

3.5 Creating an archive

This section describes the configuration of WAR and EJB JAR files, and how to create EAR files.

3.5.1 Configuring WAR files

The following table describes the configuration of the *WAR file* to be used in a POJO Web Service:

Table 3-1: Configuration of a WAR file

Directory	Remarks
/	--
META-INF/	--
MANIFEST.MF	--
WEB-INF/	--
web.xml	The created web.xml.
classes/	Stores the compiled Java class.
lib/	Stores the JAR file that contains the compiled Java classes.
wsdl/	Stores the created WSDL. To publish WSDL as the meta data of the Web Service, WSDL must be included in this directory. Also, do not include the WSDL files that are included in any other WAR file or EJB JAR file in this directory.

Legend:
--: Indicates that there is no particular description or supplementary notes

Note:

- When you include cosminexus-jaxws.xml in a WAR file, include it directly below the WEB-INF directory. For cosminexus-jaxws.xml, see *10.3 Customization using cosminexus-jaxws.xml*.
- With the JAX-WS engine of Cosminexus, multiple Web Services are included in the same WAR file. When multiple Web Services are included, the class names of the classes with different functionality cannot be repeated. If the class names of the classes with different functionality are repeated, the Web Service might not operate properly. However, if the same class is used in multiple Web Services, you can repeat the class name.

3.5.2 Configuring EJB JAR files

The following table describes the configuration of the *EJB JAR file* to be used in EJB Web Service:

Table 3-2: Configuration of the EJB JAR file

Directory	Remarks
/	Stores the compiled Java class. You must include the EJB Web Service Implementation Class in this directory.
META-INF/	--
wsdl/	Stores the created WSDL. To publish the WSDL as a meta data of the Web Service, you must include the WSDL in this directory. Also, do not include the WSDL files included in any other WAR file or EJB JAR file in this directory.
MANIFEST.MF	--

Legend:
--: Indicates that there is no particular description or supplementary note.

#
In the JAX-WS engine of Cosminexus, multiple Web Services can be included in the same EJB JAR file. If multiple Web Services are included, you cannot duplicate the class names of classes with different functions. If the class names of classes with different functions are duplicated, the Web Service might operate abnormally. However, if the same class is used in multiple Web Services, you can duplicate the class name.

3.5.3 Creating EAR files

To deploy a Web Service on a J2EE server, you create an *EAR file* containing the created WAR file or EJB JAR file. To create an EAR file, `application.xml` is required.

For the EAR file configuration, see *14.2 J2EE applications in archive format* in the *uCosminexus Application Server Application Development Guide*.

3.5.4 Creating WAR file for the settings of EJB Web Service

You can use the EJB Web Service invoke functionality in a configuration where the EAR files do not include the WAR file for settings. However, to use the Application Server functionality where the specification of WAR file is required, you can include the WAR file for settings in the EAR file. The following subsections describe about the WAR file for settings.

(1) When WAR file for settings is not included in the EAR file

When EJB JAR file is included in EAR file, and EJB Web Service Implementation Class is included in EJB JAR file, the JAX-WS engine operates assuming that the WAR file for settings is also included. The following table describes the configuration of the assumed WAR file for settings:

Table 3-3: Configuration of assumed WAR file for settings

Directory	Remarks
/	--
WEB-INF/	--
web.xml	See *3.5.4(4)*
META-INF/	--

Legend:
--: Indicates that there is no particular description or supplementary note.

(2) When WAR file for settings is included in EAR file

When invoking the EJB Web Service Implementation Class, if you want to concurrently perform additional settings to `web.xml` such as applying the servlet filter functionality, store the created `web.xml` in the WAR file for settings and include the file in the EAR file. The following table describes the configuration of WAR file for settings. Note that the WAR files for settings have naming rules. For the file name of WAR file for settings, see *3.5.4(3) Name of the WAR file for settings*.

Table 3-4: Configuration of WAR file for settings

Directory	Remarks
/	--
WEB-INF/	--

Directory	Remarks
classes/	Stores the compiled Java classes. When using the filter, store the filter of the Java class in this directory.#
lib/	Stores the JAR files that include the compiled Java classes. When using the filter, store the JAR file including the Java class of the filter in this directory.#
web.xml	See *3.5.4(4)*.
META-INF/	--

Legend:

--: Indicates that there is no particular description or supplementary note.

#

Do not include POJO Web Service in the classes of WAR file for settings. If the POJO Web Service is included the operations cannot be guaranteed.

(3) Name of the WAR file for settings

If the WAR file for settings is included in the EAR file, the name of the WAR file for settings for EJB Web Service must be same as the file name specified in the `webserver.container.jaxws.webservice.wsee.warname` property of the User Property file (usrconf.properties) for J2EE server.

If WAR file for settings is not included in the EAR file, the operation is executed assuming that the WAR file for settings with the file name specified in the `webserver.container.jaxws.webservice.wsee.warname` property is included. When WAR file for settings is not included in the EAR file, a message is output in the J2EE server log. You can thereby confirm the assumption that the JAX-WS engine includes the WAR file for settings (KDJE42391-I). If this message is not output, you can check whether the WAR file for settings specified in the property is included in the EAR file.

Note that the default value of the property is `CosminexusWSEE.war`.

You can use the following settings for invoking the EJB Web Service. To use these settings, include the WAR file for settings in the EAR file and specify the name of the WAR file for settings.

- Settings of the context root

  Specify the name of the WAR file for settings in the "`<web-uri>` element of `application.xml`".

  When context root is not set for the WAR file for settings of EJB Web Services, the context root is assumed as "/".

- Settings to be specified in the `<war>` element of `cosminexus.xml`

  Specify the name of WAR file for settings in the "`<module-name>` element of `cosminexus.xml`".

  ! Important note

  To change the value of the `webserver.container.jaxws.webservice.wsee.warname` property, stop the Web application that includes the EJB Web Service. The operation cannot be guaranteed if the property value is changed when the Web application is running. Other applications might become invalid resulting in the occurrence of an unexpected exception.

(4) web.xml of the WAR file for settings

This subsection describes the `web.xml` that is included in the WAR file for settings of EJB Web Service.

When creating `web.xml`, specify the file name as `web.xml` and save directly below the `WEB-INF` directory configuring the WAR files. Whether the saving of the `web.xml` file is mandatory differs depending upon the value set in the `webserver.container.jaxws.webservice.no_webxml.enabled` property of the User Property file for J2EE server (usrconf.properties).

- **When "strict" is specified**

  Saving the `web.xml` with the name `web.xml` directly below the `WEB-INF` directory is optional. To save the `web.xml`, you must code the definitions required for executing the Web Services.

- **When "lax" is specified**

Saving the web.xml with the name web.xml directly below the WEB-INF directory is optional. To save the web.xml file, you need not code the definitions required for executing Web Services.

- **When "none" is specified**
  You must save the web.xml with the name web.xml directly below the WEB-INF directory.

The following subsections describe the operations when web.xml is not included in the WAR file for settings and the operations when web.xml is included in the WAR file for settings.

(a) When web.xml is not included in the WAR file for settings

If you specify strict or lax in setup value of the webserver.container.jaxws.webservice.wsee.no_webxml.enabled property and do not include the web.xml in the WAR file for settings of the EJB Web Service, the processing is performed considering that the web.xml with the following contents exist while invoking the Web Service.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- code the web-app element-->
  <description>Cosminexus JAX-WS Default web.xml</description>
  <display-name>Cosminexus_JAX_WS_Default_web_xml</display-name>
  <listener>
    <listener-class>
      com.cosminexus.xml.ws.transport.http.servlet.EJBWSServletContextListener
    </listener-class>
  </listener>
  <servlet>
    <description>EJB Endpoint servlet for Cosminexus JAX-WS</description>
    <display-name>EJB_Endpoint_servlet_for_Cosminexus_JAX_WS</display-name>
    <servlet-name>CosminexusJaxwsEjbServlet</servlet-name>
    <servlet-class>
      com.cosminexus.xml.ws.transport.http.servlet.EJBWSServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>CosminexusJaxwsEjbServlet</servlet-name>
    <url-pattern>"/" + Service name of Web Service 1 + "/" + Class name of Web
Service 1</url-pattern>
    <url-pattern>"/" + Service name of Web Service 2 + "/" + + Class name of Web
Service 2</url-pattern>
      :
    <url-pattern>"/" + Service name of Web Service n + "/" + Class name of Web
Service n</url-pattern>
  </servlet-mapping>
  <session-config>
    <session-timeout>60</session-timeout>
  </session-config>
</web-app>
```

For "/" + *Service-name-of-Web-Service-1* + "/" + *Class-name-of-Web-Service-1* in the url-pattern element, a string with / (forward slash) added as a prefix is defined in the value of the serviceName attribute and the name attribute of the javax.jws.WebService annotation for the Web Service Implementation Class.

Operations for all the Web Service Implementation Classes stored in the WAR file for settings are performed assuming that the url-pattern element exists.

! Important note

- The web.xml assumed by the JAX-WS engine need not be actually generated within the WAR file for settings. This assumption is only applicable when invoking Web Services.

Example

The property files that can be acquired by the cjgetappprop command do not include the information related to web.xml. Also, if lax is specified in the webserver.container.jaxws.webservice.wsee.no_webxml.enabled property and web.xml with invalid contents is included in the WAR file for settings, you can acquire only the contents related to the web.xml that are actually included.

- If lax is specified in the webserver.container.jaxws.webservice.wsee.no_webxml.enabled property and web.xml with invalid contents is included in the WAR file for settings, the contents of the web.xml that the JAX-WS engine assumes while invoking the Web Services might not be necessarily coded in the actual web.xml.

(b) When web.xml is included in the WAR file for settings

If you specify strict in the webserver.container.jaxws.webservice.wsee.no_webxml.enabled property and include the web.xml in the WAR file, or if you specify none, create web.xml so that the following conditions are fulfilled:

- **Version**

  The web.xml version must be 2.5 or later.

- **Listener**

  Include the following `listener` elements in the web-app element:

```
<listener>
  <listener-class>
    com.cosminexus.xml.ws.transport.http.servlet.EJBWSServletContextListener
  </listener-class>
</listener>
```

- **Servlet**

  Include the following `servlet` elements in the `web-app` element.

```
<servlet>
  <description>EJB Endpoint servlet for Cosminexus JAX-WS</description>
  <display-name>EJB_Endpoint_servlet_for_Cosminexus_JAX_WS</display-name>
  <servlet-name>CosminexusJaxwsEjbServlet</servlet-name>
  <servlet-class>
    com.cosminexus.xml.ws.transport.http.servlet.EJBWSServlet
  </servlet-class>
</servlet>
```

- **Servlet mapping**

  Code the servlet-mapping element under the web-app element and include the same number of url-pattern elements as the number of Web Service Implementation Classes.

  The following is an *coding example of servlet-mapping element*:

```
<servlet-mapping>
  <servlet-name>CosminexusJaxwsEjbServlet</servlet-name>
  <url-pattern>"/" + Service name of Web Service 1 + "/" + Class name of Web
Service 1</url-pattern>
  <url-pattern>"/" + Service name of Web Service 2 + "/" +  Class name of Web
Service 2</url-pattern>
    :
  <url-pattern>"/" + Service name of Web Service n + "/" + Class name of Web
Service n</url-pattern>
</servlet-mapping>
```

  Code the following strings in the "Service name of Web Service n" and the "Class name of Web Service n" in the `url-pattern` element.

  - Service name of Web Service n

    Code the serviceName attribute value of the javax.jws.WebService annotation of the Web Service Implementation Class. If the `serviceName` attribute is omitted, code a string with `Service` suffixed to the class name (simple name) of Web Service Implementation Class.

  - Class name of Web Service n

    Code the name attribute value of the javax.xml.ws.WebService annotation of the Web Service Implementation Class. If the `name` attribute is omitted, code the class name (simple name) of Web Service Implementation Class.

- **Other elements**

  You can code any other element. To include the servlet created in the WAR file, define the element appropriately in the web.xml.

The following is an example of `web.xml` that is included in WAR file for settings of EJB Web Service:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/
javaee/web-app_3_0.xsd">
  <description>Cosminexus JAX-WS Default web.xml</description>
  <display-name>Cosminexus_JAX_WS_Default_web_xml</display-name>
```

```
  <listener>
    <listener-class>
      com.cosminexus.xml.ws.transport.http.servlet.EJBWSServletContextListener
    </listener-class>
  </listener>
  <servlet>
    <description>EJB Endpoint servlet for Cosminexus JAX-WS</description>
    <display-name>EJB_Endpoint_servlet_for_Cosminexus_JAX_WS</display-name>
    <servlet-name>CosminexusJaxwsServlet</servlet-name>
    <servlet-class>
      com.cosminexus.xml.ws.transport.http.servlet.EJBWSServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>CosminexusJaxwsServlet</servlet-name>
    <url-pattern>/AddNumbersImplService/AddNumbersImpl</url-pattern>
  </servlet-mapping>
  <session-config>
    <session-timeout>60</session-timeout>
  </session-config>
</web-app>
```

When creating `web.xml` of version 2.5, specify `2.5` in the `version` attribute of the `web-app` element and specify `http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd` as the second location information in the `xsd:schemaLocation` attribute.

In this example, it is assumed that the value of the serviceName attribute of the javax.jws.WebService annotation for the Web Service Implementation Class is `AddNumbersImplService` and the value of the `name` attribute is `AddNumbersImpl`.

The operations cannot be guaranteed when `lax` is set in the `webserver.container.jaxws.webservice.wsee.no_webxml.enabled` property and `web.xml` is included in the WAR file for settings, and listener, servlet, servlet-mapping, or any other element is not completely included.

3.6 Implementing Web Service clients

There is no limit for the types of Web Service clients. For example, you can develop the Web Service clients mentioned below. When developing a Web Service client operating on a J2EE container, use the EJB version 3.0 or later, and Servlet version 2.5 or later for Web applications such as the JSP or servlets. For example, when including `web.xml` in a WAR file by using a Web application, use the `web.xml` version 2.5 or later.

- Java applications (Calling Web Services from a Java application)
- JSPs (Invoking Web Service from JSPs)
- Servlets (Invoking Web Services from servlets)
- EJBs (Invoking Web Services from EJBs)
- Web Services (Invoking another Web Service from the Web Service Implementation Class)

With the JAX-WS functionality of Cosminexus, you can implement a Web Service client using any of the following methods:

- Using a service class and stubs (Developing stub-based Web Service clients)
- Using the `javax.xml.ws.Dispatch` interface (Developing dispatch-based Web Service clients)
- Using other JAX-WS APIs (Developing API-based Web Service clients)

This section describes the examples of implementing Web Service clients. This section also describes the precautions to be taken for using a service class and ports.

3.6.1 Example of stub-based implementation

You can develop a Web Services client by using a service class or stub that is automatically generated by executing the `cjwsimport` command. You can use any of the following methods to generate a service class or to acquire a port by implementing the stub-based Web Services client.

- Using a constructor of service classes

  Create service class instances by using a constructor of service classes. Acquire ports from the generated service class instances.
- Using the `javax.xml.ws.WebServiceRef` annotation

  Inject service classes or ports by using the `javax.xml.ws.WebServiceRef` annotation.

For injecting service classes and ports, see *10.21.1 Injecting service classes and ports*, for the `javax.xml.ws.WebServiceRef` annotation, see *19.3 Support range of annotations*.

(1) Referenced WSDLs

For executing stub-based Web Service clients, you require a path or a URL for a WSDL. During this, WSDLs to be referenced differs depending on the combination of the following conditions:

- When injecting service classes and ports by using the `javax.xml.ws.WebServiceRef` annotation

  For the WSDL documents that are referenced when injecting service classes or ports, see *19.3.1(2) The wsdlLocation element (javax.xml.ws.WebServiceRef)*.
- When generating service class instances by using a constructor for service classes
- WSDLs to be referenced vary for different combinations of the following conditions:
  - Which constructor of a service class is to be used
  - Whether the `cjwsimport` command is executed by specifying the `-wsdllocation` option

The following table describes the mapping between each condition and the referenced WSDL. For the examples of each condition, see *3.6.1(5)(a) Creating service class instances*.

Table 3-5: Mapping between the combination of conditions and referenced WSDLs

Item No.	Constructor to be used	-wsdllocation option	Referenced WSDL
1	Default constructor	N	The WSDL specified in the arguments of the `cjwsimport` command[#1]
2	Default constructor	Y	The WSDL specified in the `-wsdllocation` option[#2]
3	Constructor with the `java.net.URL` and `javax.xml.namespace.QName` objects as parameters	--	The WSDL specified in the URL of the parameters[#2]

Legend:

Y: When the option is specified.

N: When the option is not specified.

--: Specifying the option does not affect the referenced WSDL.

#1

When you execute a Web service client, the WSDL must exist at the location same as would exist during the execution of the `cjwsimport` command. Even when you specify a relative path for executing the `cjwsimport` command, the WSDL must exist at the location same as would exist during the execution of the `cjwsimport` command.

#2

When you specify an absolute URL, the WSDL must exist at the location that is indicated in the URL during the execution of the Web Services client.

When you specify a relative URL, the WSDL must exist at the location where the relative URL is resolved with the current directory as a base.

(2) Referenced endpoint address

For a URL (endpoint address) of the Web Service to be connected, the address information (`location` attribute of the `soap:address` child element) included in the WSDL port (`wsdl:port` element) will be used by default. However, if using the `javax.xml.ws.service.endpoint.address` property of the message context, you can dynamically change the endpoint address. For the examples about dynamically changing the endpoint addresses, see *3.6.1(5)(c) Invoking a method of the port*.

(3) Generating service classes and acquiring ports

The generation of a service class and acquisition of ports involves a processing cost, and therefore, we recommend that you inject or reuse ports as follows:

- When implementing Web Services clients as servlets or EJB

  Inject ports when implementing Web Services clients as servlets or EJB. For details on the port injection, see *10.21 Injection*.

- When implementing a Web Services client by using a different application (such as a command line application)

  You can use the initialization process to generate service classes and acquire ports, and reuse the generated classes and the acquired ports. You need not generate a service class every time you acquire a port or acquire a port every time you call a method.

(4) Selecting a service

When a WSDL of the Web Service to be invoked contains multiple services (`wsdl:service` element), use a constructor with the `java.net.URL` and `javax.xml.namespace.QName` objects as parameters for generating service classes, and clearly specify whether to invoke the services (`wsdl:service` element) with the `javax.xml.namespace.QName` object.

(5) Basic implementation example

To implement a Web Services client, you can generate a service class by using a constructor or can use the port injection. This sub-section describes the procedure to implement the Web Services client when generating a service class by using a constructor. For details on the port injection, see *10.21.1 Injecting service classes and ports*.

When developing stub-based Web Service clients, use the `cjwsimport` command to generate the Java sources required for invoking Web Services. Execute the `cjwsimport` command without specifying the `-generateService` option.

The stub-based Web Service client uses the following Java sources to invoke Web Services:

- **Service class**
  A service class corresponds to the WSDL services (`wsdl:service` element) that are used for invoking Web Services defined in the JAX-WS 2.2 specifications. The `wsdl:service` element compiles multiple ports in such a way so that the `wsdl:port` elements are compiled.
  For implementing a Web Service client, first of all generate the instances of the service classes.
- **Port**
  This is an instance corresponding to the WSDL port (`wsdl:port` element), and the interface is the Service Endpoint Interface (SEI). To operate as a proxy of remote connection-destination Web Services, the Web Service client can invoke a method of this port for transparently invoking the operations of the Web Services.

To implement a Web Services client for invoking the operations of Web Services:

1. Create a service class instance
2. Acquire a port
3. Invoke a method of the port

This point describes an implementation example for a Web Service client that invokes a Web Service (Web Service that performs additions) with the configuration described in *5.1 Configuration of development examples (Starting from SEI)*.

The following table describes the classes and methods used for implementing Web Service clients. As and when required, see the contents of service class products described in *5.5.1 Generating a service class*.

Table 3-6: Classes and methods used in the implementation example for Web Service clients

No.	Type	Class and method
1	Service class	`AddNumbersImplService`
2	SEI	`AddNumbersImpl`
3	Method of SEI	`int add(int, int)`
4	Main class of the client	`TestClient`

(a) Creating service class instances

The following is an example of generating objects of a service class using the default constructor:

```
// Creating a service class instance
AddNumbersImplService service = new AddNumbersImplService();
```

In this case, the WSDL that is specified in the arguments of the `cjwsimport` command or in the `-wsdllocation` option is referenced.

The following are three examples of the cases when you do not specify the `-wsdllocation` option while executing the `cjwsimport` command:

Execution example 1

```
> "%COSMINEXUS_HOME%\jaxws\bin\cjwsimport.bat" D:\dev\development.wsdl
```

`D:\dev\development.wsdl` must also exist during the execution of the Web Service client, and must be available for the reference.

Execution example 2

```
> D:
> cd D:\dev\
> "%COSMINEXUS_HOME%\jaxws\bin\cjwsimport.bat" relative\development.wsdl
```

`D:\dev\relative\development.wsdl` must exist during the execution of the Web Service client, and must also be available for the reference.

Execution example 3

```
> "%COSMINEXUS_HOME%\jaxws\bin\cjwsimport.bat" http://sample.com/fromjava/
AddNumbersImplService?wsdl
```

`http://sample.com/fromjava/AddNumbersImplService?wsdl` must exist during the execution of the Web Service client, and must also be available for the reference.

The following are two examples of the cases when you specify the `-wsdllocation` option while executing the `cjwsimport` command:

Execution example 1

```
> "%COSMINEXUS_HOME%\jaxws\bin\cjwsimport.bat" -wsdllocation file:/home/
wsdl4runtime/master.wsdl D:\dev\development.wsdl
```

`file:/home/wsdl4runtime/master.wsdl` must exist during the execution of the Web Service client, and must also be available for the reference. `D:\dev\development.wsdl` is used only to generate the Java code required for the implementation, for developing a Web Service client. Therefore, there is no problem even if `D:\dev\development.wsdl` does not exist during the execution or `D:\dev\development.wsdl` is not available for the reference.

Execution example 2

```
> "%COSMINEXUS_HOME%\jaxws\bin\cjwsimport.bat" -wsdllocation ./wsdl4runtime/
master.wsdl D:\dev\development.wsdl
```

When you assume the current directory during the execution of the Web Service client as *runtime-current-directory*, *runtime-current-directory*`/wsdl4runtime/master.wsdl` must exist and available for the reference.

The following example describes a case when using a constructor having the `java.net.URL` object and the `javax.xml.namespace.QName` object as parameters, instead of using the default constructor:

```
// Creating a service class instance
java.net.URL wsdlLocation = new java.io.File( "./wsdl4runtime/master.wsdl" ).toURL();
javax.xml.namespace.QName serviceName =
  new javax.xml.namespace.QName( "http:/sample.com/", "AddNumbersImplService");
AddNumbersImplService service =
  new AddNumbersImplService( wsdlLocation, serviceName );
```

Because a WSDL with the URL specified by the `java.net.URL` object is referenced, it is all right even if the WSDL specified by the arguments of the `cjwsimport` command and by the `-wsdllocation` option do not exist or cannot be referenced during the execution of the Web Service client. However, when you assume *runtime-current-directory* as the current directory during the execution of the Web Service client, *runtime-current-directory*`/wsdl4runtime/master.wsdl` must exist and available for the reference.

(b) Acquiring a port

The following is an example of acquiring a port from the service class for which instances are created:

```
// Acquire a port
AddNumbersImpl port = service.getAddNumbersImplPort();
```

(c) Invoking a method of the port

The following is an example of invoking a method of the port acquired from the service class for which instances are created:

```
// Invoke a method of the port
int returnValue = port.add(205, 103)
```

In this execution example, if you pass two values to the arguments of the method of the port, the addition processing will be performed in the Web Services. The result of addition will be returned as the return value.

By default, the address information (`location` attribute of the `soap:address` child element) included in the WSDL port (`wsdl:port` element) is used as the URL (endpoint address) of the Web Service to be connected. However, if you use the `javax.xml.ws.service.endpoint.address` property of the message context, you can dynamically change the endpoint address.

If you are not dynamically changing the endpoint address, check that a URL that can be accessed from the Web Service client is coded in the `location` attribute of the `soap:address` child element of the referenced WSDL.

If you are dynamically changing the endpoint address, acquire a request context before invoking a method of the port, and then change the value of the `javax.xml.ws.service.endpoint.address` property. The following is an example:

```
// Acquire a request context
java.util.Map<String, Object> context =
  ( ( javax.xml.ws.BindingProvider )port ).getRequestContext();

// Change the endpoint address
// (javax.xml.ws.BindingProvider.ENDPOINT_ADDRESS_PROPERTY is a constant
//  defining "javax.xml.ws.service.endpoint.address")
context.put( javax.xml.ws.BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
   "http://other.remote.org/fromjava/AddNumbersImplService" );

// Invoke a method of the port
int returnValue = port.add(205, 103)
```

Notes

The generation of a service class and acquisition of a port involve a processing cost, so we recommend the re-usage of the generated or acquired service classes and ports. For invoking a Web method of port multiple times, you need not generate the service class and acquire the port multiple times However, when sharing the port with multiple threads, the changes in the property of request context of the port to be shared must be performed before operating multiple threads. If the changes are performed when operating multiple threads, communication might fail and an invalid SOAP message might be sent.

The following is an example of invoking a method of port multiple times:

```
// Create a service class instance
AddNumbersImplService service = new AddNumbersImplService();
// Acquire a port
AddNumbersImpl port = service.getAddNumbersImplPort();
// Invoke a method of the port multiple times
for (int i = 0; i < 10; i++) {
 int returnValue = port.add(i, i);
}
```

(6) Implementation example of Web Service clients for a Java application

From a Web Service client for a Java application, use the service classes to implement the processing for invoking a Web Service.

This point describes an implementation example for a Web Service client that invokes a Web Service (Web Service that performs additions) with the configuration described in *5.1 Configuration of development examples (Starting from SEI)*. The following table describes the classes and methods used for implementing Web Service clients. As and when required, see the contents of service class products described in *5.5.1 Generating a service class*.

Table 3-7: Classes and methods used in the implementation example for Web Service clients (Web Service client for a Java application)

No.	Type	Class and method
1	Service class	`AddNumbersImplService`
2	Port	`AddNumbersImpl`
3	Method of the port	`int add(int, int)`
4	Main class of the client of a Web Service client	`TestClient`

The following is an execution example for Java applications:

```
package com.example.sample.client;

import com.example.sample.AddNumbersImplTestJaxWs;
import com.example.sample.AddNumbersImplTestJaxWsService;
import com.sample.AddNumbersFault_Exception;

// Sample implementation of web service's client
public class TestClient {
    public static void main( String[] args ) {
        try {
            // Create a service class instance
            AddNumbersImplTestJaxWsService service = new
AddNumbersImplTestJaxWsService();
            // Acquire a port
            AddNumbersImplTestJaxWs port = service.getAddNumbersImplPortTestJaxWs();

            // Invoke a method of the port
            port.jaxWsTest1( ... );
            int number1 = 205;
            int number2 = 103;
            int returnValue = port.add(number1, number2);

            // Display the results
            System.out.println( "[RESULT] " + number1 + " + " + number2 + " = " +
returnValue );
        }
        catch( Exception e ){
            // Exception processing (Here, simply output the stack trace)
            e.printStackTrace();
        }
    }
}
```

The execution result of the program is as follows:

```
[RESULT] 205 + 103 = 308
```

(7) Implementation example of Web Services clients for servlets

From a Web Service client of the servlet type, implement the processing for invoking Web Services using the service class.

This point describes an implementation example for a Web Service client that invokes a Web Service (Web Service that performs additions) with the configuration described in *5.1 Configuration of development examples (Starting from SEI)*. The following table describes the classes and methods used for implementing Web Service clients. As and when required, see the contents of service class products described in *5.5.1 Generating a service class*.

Table 3-8: Classes and methods used in the implementation example for the Web Service client (When invoked from servlets)

No.	Type	Class and method
1	Service class	`AddNumbersImplService`
2	Port	`AddNumbersImpl`
3	Method of the port	`int add(int, int)`

No.	Type	Class and method
4	Servlet Implementation Class acting as a Web Service client	`TestClient`

(a) Injecting ports

When a Web Service is invoked from a servlet, specify the `javax.xml.ws.WebServiceRef` annotation in the port type field, and inject ports.

The following is an example of injecting a port:

```
...
public class TestClient extends HttpServlet {

        // Inject a port in the port(AddNumbersImpl)type field
       @WebServiceRef(AddNumbersImplService.class)
    AddNumbersImpl port;

    @Override
    public void init() {

    }
    ...
}
```

(b) Executing Web Service operations by invoking a method of the port

Use the port generated as a field of the servlet and invoke a method.

The following is an example of invoking the method:

```
...
public class TestClient extends HttpServlet {

        // Inject a port in the port(AddNumbersImpl)type field
       @WebServiceRef(AddNumbersImplService.class)
    AddNumbersImpl port;
    ...
    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        ...
            int number1 = ...; // Substitute the value acquired from the request
object
            int number2 = ...; // Substitute the value acquired from the request
object
            // Invoke the method of the port
            int returnValue = port.add(number1, number2);
            ...
    }
}
```

The following is a complete implementation example for invoking a Web Service from a servlet:

```
package com.sample.client;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.sample.AddNumbersFault_Exception;
import com.sample.AddNumbersImpl;
import com.sample.AddNumbersImplService;
public class TestClient extends HttpServlet {

        // Inject a port in the port(AddNumbersImpl)type field
       @WebServiceRef(AddNumbersImplService.class)
    AddNumbersImpl port;

    @Override
    public void init() {
```

```
    }

    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        PrintWriter out = response.getWriter();

        try {
            // Invoke a method of the target web service.
            int number1 = ...; // Substitute the value acquired from the request
object
            int number2 = ...; // Substitute the value acquired from the request
object
            // Invoke the method of the port
            int returnValue = port.add(number1, number2);
            // Display the results
            out.println("<html><body>");
            out.println("<h1>RESULT</h1>");
            out.println(number1 + " + " + number2 + " = " + returnValue);
            out.println("</body></html>");
        } catch(AddNumbersFault_Exception e) {
            // Exception processing (Here, simply output the detailed message of the
exception)
            out.println("<html><body>");
            out.println("<h1>" + e.getMessage() + "</h1>");
            out.println("</body></html>");
        }
    }
}
```

The following is the execution result of the program. The execution result is displayed when you connect browser to the servlet.

```
RESULT
205 + 103 = 308
```

(C) Notes

In an environment where a Web Services client (servlet) and Web Service you want to connect to, both are deployed on the same J2EE server, an exception occurs if you start the J2EE server in the following conditions:

- When generating a service class by acquiring the WSDL from Web Services, if you specify the `load-on-startup` element in `web.xml` of the WAR file that has the Web Services client, in methods in which you have specified the `init` method or the `javax.annotation.PostConstruct` annotation.
- If you inject a service class or a port by acquiring the WSDL from Web Services by using the `javax.xml.ws.WebServiceRef` annotation.

The following is a list of actions to be taken if an exception is thrown:

- End the Web application that includes the Web Services client before you stop the J2EE server. After you restart the J2EE server, restart a Web application that includes the Web Services client.
- After you deploy Web Service and a Web Services client on a different J2EE server and start the J2EE server on which Web Service is deployed, start the J2EE server on which the Web Services client is deployed.
- Use the catalog functionality to perform the settings in such a way that you can reference the WSDL document that is locally stored when service classes are generated.
- When generating a service class by using the `init` method or a method in which the `javax.annotation.PostConstruct` annotation is specified, specify the local WSDL document in a constructor that uses a URL as a parameter and then generate a service class. Alternatively, do not specify the `load-on-startup` element in `web.xml`.
- When using the `javax.xml.ws.WebServiceRef` annotation, specify the locally stored WSDL document in the `wsdlLocation` element with a relative path or absolute path.

3.6.2 Example of dispatch-based implementation

Use APIs of the `javax.xml.ws.Dispatch` interface, the JAX-WS 2.2 specifications, JAXB 2.2 specifications, and SAAJ 1.3 specifications that are supported by Cosminexus, and perform the development.

(1) Examples of implementing a dispatch-based Web Service client

The following is an *example of implementing a dispatch-based Web Service client*:

```
package com.example.sample.client;

import javax.xml.namespace.QName;
import javax.xml.soap.MessageFactory;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPBodyElement;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPException;
import javax.xml.soap.SOAPMessage;
import javax.xml.ws.Dispatch;
import javax.xml.ws.Service;
import javax.xml.ws.soap.SOAPBinding;
public class TestClient {
    public static void main( String[] args ) {
        QName port = new QName( "http://sample.com", "AddNumbersImplPort" );
        SOAPBody soapBody = null;

        // Generate a service
        Service service = Service.create(
            new QName("http://sample.com", "UserInfoPort") );

        // Add a port to the service
        service.addPort( port, SOAPBinding.SOAP11HTTP_BINDING,
            "http://localhost:80/dispatch_provider/UserInfoService" );

        // Generate a dispatch
        Dispatch<SOAPMessage> dispatch = service.createDispatch(
            port, SOAPMessage.class, Service.Mode.MESSAGE );

        SOAPMessage request = null;
        try{
            // Generate a request message using the APIs of the SAAJ 1.3
specifications
            request = MessageFactory.newInstance().createMessage();
            SOAPBody reqSoapBody = request.getSOAPBody();
            SOAPElement soapElement = null;
            // Add elements to the SOAP Body
            SOAPBodyElement requestRoot= reqSoapBody.addBodyElement(
                new QName( ... ) );
            soapElement = requestRoot.addChildElement(
                new QName( ... ) );
            soapElement.addTextNode( ... );
            // Add an attachment
            File attachment = new File( ... );
            FileDataSource fds = new FileDataSource( attachment );
            AttachmentPart apPart = request.createAttachmentPart( new
DataHandler( fds ) );
            request.addAttachmentPart(apPart);
        }
        catch( SOAPException e ){
            // Exception processing
        }
        // Specify the created request message, and invoke the Web Service using the
dispatch
        SOAPMessage response = dispatch.invoke( request );

        try{
            // Perform the required processing for the response message
            SOAPBody resSoapBody = response.getSOAPBody();
            ...
        }
        catch( SOAPException e ){
            // Exception processing
        }
    }
}
```

When using a SOAP fault sent from the provider implementation class for implementing a Web Service client, invoke the `invoke()` method within the `try-catch` block and acquire the `javax.xml.ws.soap.SOAPFaultException` exception. You can also acquire the SOAP fault from the `javax.xml.ws.soap.SOAPFaultException` exception. The following is an implementation example:

```
package com.example.sample.client;

import javax.xml.namespace.QName;
```

```
...
import javax.xml.ws.soap.SOAPBinding;
import javax.xml.ws.soap.SOAPFaultException;

public class TestClient {
    public static void main( String[] args ) {
...
        try{
            // Generate the request message using the APIs of SAAJ 1.3 specifications
            ...
        }
        catch( SOAPException e ){
            // Exception processing
        }
        SOAPMessage response = null;
        try{
            // Specify the created request message, and invoke a Web Service using
the dispatch
            response = dispatch.invoke( request );
        }
        catch( SOAPFaultException e ){
            // Processing for acquiring the SOAP fault
            SOAPFault fault = e.getFault();
            // Perform the required processing for the acquired SOAP fault
            String faultCode = fault.getFaultCode();
            ...
        }
        try{
            // Perform the required processing for the response message
            ...
        }
        catch( SOAPException e ){
            e.printStackTrace();
        }
    }
}
```

(2) Referenced endpoint address

You can specify and change a URL (endpoint address) of the Web Service to be connected with the `javax.xml.ws.service.endpoint.address` property of the message context. For the examples about specifying and changing the endpoint addresses, see *3.6.1(5)(c) Invoking a method of the port*.

(3) Reusing a service class and dispatch

The generation of a service class and dispatch requires processing cost, so we recommend that you reuse the generated service class and the dispatch. You need not generate a service class more than once to add a port and generate a dispatch. Also, you need not acquire a dispatch more than once to invoke a method of the dispatch multiple times However, when sharing the dispatch with multiple threads, the changes in the property of request context of the dispatch to be shared must be performed before operating multiple threads. If the changes are performed when operating multiple threads, communication might fail and an invalid SOAP message might be sent.

For implementing a Web Service client with servlets and EJBs, we recommend that you acquire service classes and the dispatch using each of the initialization methods, and then reuse them. Changes in the request context property of dispatch must be performed in each of the initialization methods.

3.6.3 Examples of implementation using JAX-WS API

You can use the JAX-WS API supported by the JAX-WS functionality of Cosminexus to implement a Web Service client. For the support range of the JAX-WS APIs, see *19.2 Support range of the JAX-WS APIs*.

(1) Example of implementing a Web Service client using JAX-WS API

An *example of implementing a Web Service client* using JAX-WS API is as follows:

```
package com.example.sample.client;

import com.example.sample.TestJaxWs;
import com.example.sample.TestJaxWsService;
```

```
import java.net.URL;
import java.net.MalformedURLException;
import java.util.Iterator;
import java.util.Map;
import javax.xml.namespace.QName;
import javax.xml.ws.BindingProvider;
import javax.xml.ws.Service;
import javax.xml.ws.WebServiceException;
public class TestClient {
 public static void main( String[] args ) {

 // Specify the WSDL URL and service name
 URL url = null;
 try {
  url = new URL("http://localhost:8085/fromwsdl/test?wsdl");
 } catch (MalformedURLException e) {
  // Exception processing
 }
 QName serviceName =
  new QName("http://example.com/sample", "TestJaxWsService");
 // Generating a Service instance
 Service service = Service.create(url, serviceName);
 System.out.println(service.getWSDLDocumentLocation());

 QName portName = null;

 // Display the list of port names
 Iterator it = service.getPorts();
 while(it.hasNext()) {
  portName = (QName)it.next();
  System.out.println(portName);
 }


 // Acquire a port
 TestJaxWs port = (TestJaxWs)service.getPort(TestJaxWs.class);

 // Acquire the sending context
 Map<java.lang.String, java.lang.Object>context =
((BindingProvider)port).getRequestContext();
 System.out.println(context.entrySet());

 // Invoke the service method
 try {
  port.jaxWsTest1( "TEST", 1c23);
 } catch (WebServiceException e) {
  // Exception processing
 }
 }
}
```

(2) Reusing the javax.xml.ws.Service object and the port

The generation of a `javax.xml.ws.Service` object and the acquisition of a port require processing cost, so we recommend that you reuse the generated `javax.xml.ws.Service` object. You need not generate `javax.xml.ws.Service` objects more than once to acquire a port.

Similarly, acquisition of the port also incurs processing costs and therefore we recommend the reuse of acquired ports. You need not acquire the port more than once for invoking the Web method of port multiple times. However, when sharing the port with multiple threads, the changes in the property of request context of the port to be shared must be performed before operating multiple threads. If the changes are performed when operating multiple threads, communication might fail and an invalid SOAP message might be sent.

To implement a Web Service client using servlets and EJBs, generate the `javax.xml.ws.Service` object or acquire the port in each of the initialization methods, and reuse the object. Changes in the request context property of the port must be executed in each of the initialization methods.

3.6.4 Notes

This subsection describes the notes for implementing Web Service clients.

(1) Reusing an object

The generation of a service class, port, and dispatch requires processing cost, so we recommend that you use the injection (only for the stub-based Web Service clients) or reuse the objects. For reusing the objects, see each of the following sections:

- For stub-based Web Service clients
  *3.6.1(3) Generating service classes and acquiring port*s
- For dispatch-based Web Service clients
  3.6.2(3) Reusing a service class and dispatch
- For API-based Web Service clients
  3.6.3(2) Reusing the javax.xml.ws.Service object and the port

(2) Setting a proxy, SSL connection, and basic authentication

As and when required, specify the settings for a proxy, SSL connection, and the basic authentication in the execution environment of Web Service clients. For details, see each of the following sections:

- Setting a proxy
  10.10 Connecting through a proxy server
- Setting an SSL connection
  10.11 Connection by SSL protocol
- Setting basic authentication
  10.12 Connection by basic authentication

(3) Notes for the Windows environment

In the environment in which you send a large number of requests from the Web Service client, the following exception might be recorded in some cases:

```
java.net.BindException: Address already in use: connect [errno=10048,
syscall=select]
```

For example, an exception occurs when large number of requests reach the Web Service client that is implemented as a servlet.

In such a case, revise either one or both of the following values:

- Increase the range of port numbers that are available in the OS
- Decrease the duration of `TIME_WAIT`

For example, revise the settings of `MaxUserPort` and `TcpTimedWaitDelay` in the registry. However, the specifications differ depending on the OS version, edition, and the application status of the security update program, and therefore, see the documentation of each OS for details. Also, you must note that the settings affect the entire OS.

3.6.5 Notes on accessing the Web Services that use the addressing functionality

When you access Web Services in which the addressing functionality is enabled, use stub-based Web Service clients. Note that you cannot use dispatch-based Web Service clients.

4 Examples of the Development Starting from WSDL

This chapter describes the examples for the development of Web Services starting from WSDL.

4.1 Configuration examples of development (Starting from WSDL)

This section describes the examples of the development of Web Services starting from WSDL.

The following table describes the configuration for developing Web Services:

Table 4-1: Web Service configuration (Starting from WSDL)

No.	Item		Value
1	Name of the J2EE server to be deployed		`jaxwsserver`
2	Host name and port number of the Web server		`webhost:8085`
3	URL of the naming server		`corbaname::testserver:900`
4	Context root		`fromwsdl`
5	Style		`document/literal/wrapped`
6	Namespace URI		`http://example.com/sample`
7	Port type	Number	`1`
8		Local name	`TestJaxWs`
9	Operation	Number	`1`
10		Local name	`jaxWsTest1`
11	Service	Number	`1`
12		Local name	`TestJaxWsService`
13	Port	Number	`1`
14		Local name	`testJaxWs`
15	WSDL file name		`input.wsdl`

The following table describes the configuration of the current directory when developing Web Services.

Table 4-2: Configuration of the current directory (Starting from a WSDL)

Directory	Description
`c:\temp\jaxws\works\fromwsdl`	This is the current directory.
`server\`	Used for the development of Web Services.
`META-INF\`	Corresponds to the META-INF directory of EAR files.
`application.xml`	Created in *4.3.6 Creating application.xml*.
`src\`	Stores the source file (`*.java`) for Web Services. Used in *4.3.2 Generating SEI* and *4.3.4 Compiling the Web Service Implementation Class*.
`WEB-INF\`	Corresponds to the WEB-INF directory of WAR files.
`web.xml`	Created in *4.3.5 Creating web.xml*.
`classes\`	Stores the compiled class file (`*.class`).
`wsdl\`	Stores the created wsdl.
`temporary\`	Saves temporary files, when creating a WSDL based on the WSDL that is coded and converted using Java. Temporary directory is optional.
`src\`	

Directory	Description
classes\	Saves temporary files, when creating a WSDL based on the WSDL that is coded and converted using Java. Temporary directory is optional.
fromwsdl.ear	Created in *4.3.7 Creating EAR files*.
fromwsdl.war	
client\	Used for the development of Web Service clients.
src\	Stores the source file (`*.java`) of the Web Service client. Used in *4.5.1 Generating a service class* and *4.5.2 Creating an implementation class for the Web Service client*.
classes\	Stores the compiled class file (`*.class`). Used in *4.5.3 Compiling the implementation class for the Web Service client*.
usrconf.cfg	Created in *4.6.1 Creating an option definition file for Java applications*.
usrconf.properties	Created in *4.6.2 Creating a user property file for Java applications*.

Change the current directory path according to the environment to be developed.

Note that the directory and file names listed in the above table will be used in the description hereafter. The part in *Bold* in the command execution examples and in the Java source indicates the specified values and the generated values that are used in examples. Read according to the environment that you want to build.

Also, in the development examples described in this chapter, Web Services and Web Service clients are developed in the same environment, but you can also develop them in different environments. For developing Web Services and Web Service clients in different environments, read the current directory path suitable to the respective environments.

4.2 Examples for the procedure of development (Starting from WSDL)

The development and execution flow described in the development examples of this chapter are as follows:

Developing a Web Service

1. Creating a WSDL file (*4.3.1*)
2. Executing the `cjwsimport` command and generating SEI (*4.3.2*)
3. Creating the Web Service Implementation Class (*4.3.3*)
4. Compiling the Web Service Implementation Class (*4.3.4*)
5. Creating `web.xml` (*4.3.5*)
6. Creating `application.xml` (*4.3.6*)
7. Creating an EAR file (*4.3.7*)

Deploying and starting

1. Deploying the EAR file (*4.4.1*)
2. Starting the Web Service (*4.4.2*)

Developing a Web Service client

1. Executing the `cjwsimport` command and generating a service class (*4.5.1*)
2. Creating the implementation class for the Web Service client (*4.5.2*)
3. Compiling the implementation class for the Web Service client (*4.5.3*)

Executing a Web Service

1. Creating the option definition file for Java applications (*4.6.1*)
2. Creating the user property file for Java applications (*4.6.2*)
3. Executing the Web Service client (*4.6.3*)

4.3 Examples for the development of Web Services (Starting from WSDL)

This section describes the examples of the development of Web Services starting from WSDL.

4.3.1 Creating a WSDL file

Create a WSDL file and define the meta data for Web Services. Define the WSDL definition within the support range of the following specifications:

- WSDL 1.1 specifications
  For the support range, see *20.1 Support range of the WSDL 1.1 specifications*.
- XML Schema specifications
  For the support range, see the *uCosminexus Application Server XML Processor User Guide*.
- WS-I Basic Profile 1.1

There are two methods for creating a WSDL file; create a new WSDL file or create a WSDL file by using other WSDL file and changing the Java source.

(1) Creating a new WSDL file

Create a WSDL file (`input.wsdl`). Save the created WSDL file in the `c:\temp\jaxws\works\fromwsdl\server\WEB-INF\wsdl\` directory with the UTF-8 format.

The following is an example of creating a new WSDL file for SOAP 1.1:

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="TestJaxWsService"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
 xmlns:tns="http://example.com/sample"
 targetNamespace="http://example.com/sample">

<wsdl:types>
 <xsd:schema targetNamespace="http://example.com/sample">
 <!-- wrapper element of the request message -->
 <xsd:element name="jaxWsTest1" type="tns:jaxWsTest1"/>

 <!-- wrapper element of the response message -->
 <xsd:element name="jaxWsTest1Response" type="tns:jaxWsTest1Response"/>

 <!-- wrapper element of the fault message -->
 <xsd:element name="UserDefinedFault" type="tns:UserDefinedFault"/>

 <!-- Type referenced by the wrapper element of the request message-->
 <xsd:complexType name="jaxWsTest1">
 <xsd:sequence>
  <xsd:element name="information" type="xsd:string"/>
  <xsd:element name="count" type="xsd:int"/>
 </xsd:sequence>
 </xsd:complexType>

 <!-- Type referenced by the wrapper element of the response message-->
 <xsd:complexType name="jaxWsTest1Response">
 <xsd:sequence>
  <xsd:element name="return" type="xsd:string"/>
 </xsd:sequence>
 </xsd:complexType>

 <!-- Type referenced by the wrapper element of the fault message-->
 <xsd:complexType name="UserDefinedFault">
 <xsd:sequence>
  <xsd:element name="additionalInfo" type="xsd:int"/>
  <xsd:element name="detail" type="xsd:string"/>
  <xsd:element name="message" type="xsd:string"/>
 </xsd:sequence>
 </xsd:complexType>
```

```
  </xsd:schema>
  </wsdl:types>
  <!-- Request message -->
  <wsdl:message name="jaxWsTest1Request">
  <wsdl:part name="inputParameters" element="tns:jaxWsTest1"/>
  </wsdl:message>

  <!-- Response message -->
  <wsdl:message name="jaxWsTest1Response">
  <wsdl:part name="outputParameters" element="tns:jaxWsTest1Response"/>
  </wsdl:message>

  <!-- Fault message -->
  <wsdl:message name="UserDefinedException">
  <wsdl:part name="fault" element="tns:UserDefinedFault"/>
  </wsdl:message>
  <!-- Port type -->
  <wsdl:portType name="TestJaxWs">
  <!-- Operation -->
  <wsdl:operation name="jaxWsTest1">
  <wsdl:input message="tns:jaxWsTest1Request"/>
  <wsdl:output message="tns:jaxWsTest1Response"/>
  <wsdl:fault name="UserDefinedFault"
  message="tns:UserDefinedException"/>
  </wsdl:operation>
  </wsdl:portType>
  <!-- Binding (SOAP 1.1/ HTTP binding) -->
  <wsdl:binding name="testJaxWsBinding" type="tns:TestJaxWs">
  <!-- document/literal/wrapped -->
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <!-- Operation -->
  <wsdl:operation name="jaxWsTest1">
  <soap:operation/>
  <wsdl:input>
  <soap:body use="literal"/>
  </wsdl:input>
  <wsdl:output>
  <soap:body use="literal"/>
  </wsdl:output>
  <wsdl:fault name="UserDefinedFault">
  <soap:fault name="UserDefinedFault" use="literal"/>
  </wsdl:fault>
  </wsdl:operation>
  </wsdl:binding>
  <!-- Service-->
  <wsdl:service name="TestJaxWsService">
  <!-- Port -->
  <wsdl:port name="testJaxWs" binding="tns:testJaxWsBinding">
  <soap:address location="http://webhost:8085/fromwsdl/TestJaxWsService"/>
  </wsdl:port>
  </wsdl:service>

 </wsdl:definitions>
```

The following is an example of creating a new WSDL file for SOAP 1.2:

```
 <?xml version="1.0" encoding="UTF-8"?>
 <wsdl:definitions name="TestJaxWsService"
   xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
   xmlns:tns="http://example.com/sample"
   targetNamespace="http://example.com/sample">

   <wsdl:types>
     <xsd:schema targetNamespace="http://example.com/sample">
       <!-- wrapper element of the request message -->
       <xsd:element name="jaxWsTest1" type="tns:jaxWsTest1"/>

       <!-- wrapper element of the response message -->
       <xsd:element name="jaxWsTest1Response" type="tns:jaxWsTest1Response"/>

       <!-- wrapper element of the fault message -->
       <xsd:element name="UserDefinedFault" type="tns:UserDefinedFault"/>

       <!-- Type referenced by the wrapper element of the request message -->
       <xsd:complexType name="jaxWsTest1">
         <xsd:sequence>
           <xsd:element name="information" type="xsd:string"/>
           <xsd:element name="count" type="xsd:int"/>
         </xsd:sequence>
```

```
      </xsd:complexType>

      <!-- Type referenced by the wrapper element of the response message -->
      <xsd:complexType name="jaxWsTest1Response">
        <xsd:sequence>
          <xsd:element name="return" type="xsd:string"/>
        </xsd:sequence>
      </xsd:complexType>

      <!-- Type referenced by the wrapper element of the fault message -->
      <xsd:complexType name="UserDefinedFault">
        <xsd:sequence>
          <xsd:element name="additionalInfo" type="xsd:int"/>
          <xsd:element name="detail" type="xsd:string"/>
          <xsd:element name="message" type="xsd:string"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:schema>
  </wsdl:types>

  <!-- Request message -->
  <wsdl:message name="jaxWsTest1Request">
    <wsdl:part name="inputParameters" element="tns:jaxWsTest1"/>
  </wsdl:message>
  <!-- Response message -->
  <wsdl:message name="jaxWsTest1Response">
    <wsdl:part name="outputParameters" element="tns:jaxWsTest1Response"/>
  </wsdl:message>

  <!-- Fault message -->
  <wsdl:message name="UserDefinedException">
    <wsdl:part name="fault" element="tns:UserDefinedFault"/>
  </wsdl:message>

  <!-- Port type -->
  <wsdl:portType name="TestJaxWs">
    <!-- Operation -->
    <wsdl:operation name="jaxWsTest1">
      <wsdl:input message="tns:jaxWsTest1Request"/>
      <wsdl:output message="tns:jaxWsTest1Response"/>
      <wsdl:fault name="UserDefinedFault"
        message="tns:UserDefinedException"/>
    </wsdl:operation>
  </wsdl:portType>
  <!-- Binding (SOAP 1.2/HTTP binding) -->
  <wsdl:binding name="testJaxWsBinding" type="tns:TestJaxWs">
    <!-- document/literal/wrapped -->
    <soap12:binding style="document" transport="http://www.w3.org/2003/05/soap/
bindings/HTTP/"/>
    <!-- Operation -->
    <wsdl:operation name="jaxWsTest1">
      <soap12:operation/>
      <wsdl:input>
        <soap12:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap12:body use="literal"/>
      </wsdl:output>
      <wsdl:fault name="UserDefinedFault">
        <soap12:fault name="UserDefinedFault" use="literal"/>
      </wsdl:fault>
    </wsdl:operation>
  </wsdl:binding>
  <!-- Service -->
  <wsdl:service name="TestJaxWsService">
    <!-- Port -->
    <wsdl:port name="testJaxWs" binding="tns:testJaxWsBinding">
      <soap12:address location="http://webhost:8085/fromwsdl/TestJaxWsService"/>
    </wsdl:port>
  </wsdl:service>

</wsdl:definitions>
```

(2) Creating a WSDL file based on the WSDL file in which the Java source is converted

Create a Web Service Implementation Class and an exception class to be temporarily implemented for the WSDL conversion, and execute the WSDL generation functionality of the `cjwsgen` command to create a WSDL file from the already compiled Java source. Use the `javax.jws.WebService` annotation to specify the annotations for the created class. You need not implement any method.

The following is an example of the temporarily implemented Web Service Implementation Class:

```
package com.example.sample;

@javax.jws.WebService
public class TestJaxWsImpl {

    public String jaxWsTest1(String information, int count)
        throws UserDefinedException
    {
        // Need not be implemented
        return null;
    }

}
```

The following is an example of the temporarily implemented exception class:

```
package com.example.sample;

public class UserDefinedFault extends Exception{
    // Need not be implemented
    public int additionalInfo;
    public String detail;
    public String message;
}
```

Save and compile the created classes `TestJaxWsImpl.java` and `UserDefinedFault.java` to the `c:\temp\jaxws\works\fromwsdl\server\temporary\src\com\example\sample\` directory with the UTF-8 format. The following is an compilation example:

```
> cd c:\temp\jaxws\works\fromwsdl\server\
> mkdir .\temporary
> mkdir .\temporary\classes
> javac -encoding UTF-8 -cp "%COSMINEXUS_HOME%\jaxws\lib\cjjaxws.jar;%COSMINEXUS_HOME%
\CC\client\lib\j2ee-javax.jar" -d .\temporary\classes .\temporary\src\com\example
\sample\TestJaxWsImpl.java .\temporary\src\com\example\sample\UserDefinedFault.java
```

If compilation is successful, `TestJaxWsImpl.class` and `UserDefinedFault.class` are generated in the `c:\temp\jaxws\works\fromwsdl\server\temporary\classes\com\example\sample\` directory. Use the class files and create a WSDL file with the WSDL generation functionality of the `cjwsgen` command.

The following is an example for executing the `cjwsgen` command:

```
> cd c:\temp\jaxws\works\fromwsdl\server\
> mkdir .\WEB-INF\wsdl\
> "%COSMINEXUS_HOME%\jaxws\bin\cjwsgen.bat" -r .\WEB-INF\wsdl -d .\temporary\classes -
cp .\temporary\classes com.example.sample.TestJaxWsImpl
```

If the `cjwsgen` command is successfully terminated, `TestJaxWsService.wsdl` and `TestJaxWsService_schema1.xsd` are generated in the `c:\temp\jaxws\works\fromwsdl\WEB-INF\wsdl\` directory. Delete the classes that exist in the `c:\temp\jaxws\works\fromwsdl\temporary\classes\` directory.

You must partially modify the generated `TestJaxWsService.wsdl` and `TestJaxWsService_schema1.xsd`.

The following is an example of modifying `TestJaxWsService.wsdl`. The text in italics indicates the modified part.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<definitions targetNamespace="http://example.com/sample" name="TestJaxWsImplService"
  xmlns:tns=http://example.com/sample
  xmlns:xsd=http://www.w3.org/2001/XMLSchema
  xmlns:soap=http://schemas.xmlsoap.org/wsdl/soap/
  xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types>
    <xsd:schema targetNamespace=" http://example.com/sample ">
      <xsd:include schemaLocation="TestJaxWsImplService_schema1.xsd"/>
    </xsd:schema>
```

```
  </types>
  <message name="jaxWsTest1">
    <part name="parameters" element="tns:jaxWsTest1"/>
  </message>
  <message name="jaxWsTest1Response">
    <part name="parameters" element="tns:jaxWsTest1Response"/>
  </message>
  <message name="UserDefinedFault">
    <part name="fault" element="tns:UserDefinedFault"/>
  </message>
  <portType name="TestJaxWs">
    <operation name="jaxWsTest1">
      <input message="tns:jaxWsTest1"/>
      <output message="tns:jaxWsTest1Response"/>
      <fault message="tns:UserDefinedFault" name="UserDefinedFault"/>
    </operation>
  </portType>
  <binding name="testJaxWsBinding" type="tns:TestJaxWs">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <operation name="jaxWsTest1">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
      <fault name="UserDefinedFault">
        <soap:fault name="UserDefinedFault" use="literal"/>
      </fault>
    </operation>
  </binding>
  <service name="TestJaxWsService">
    <port name="testJaxWs" binding="tns:testJaxWsBinding">
      <soap:address location="http://webhost:8085/fromwsdl/TestJaxWsService"/>
    </port>
  </service>
</definitions>
```

The following is an example of modifying `TestJaxWsService_schema1.xsd`. The text in italics indicates the modified part.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema version="1.0"
  targetNamespace=http://example.com/sample
  xmlns:tns=http://example.com/sample
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="UserDefinedFault" type="tns:UserDefinedFault"/>

  <xs:element name="jaxWsTest1" type="tns:jaxWsTest1"/>

  <xs:element name="jaxWsTest1Response" type="tns:jaxWsTest1Response"/>

  <xs:complexType name="jaxWsTest1">
    <xs:sequence>
      <xs:element name="arg0" type="xs:string" minOccurs="0"/>
      <xs:element name="arg1" type="xs:int"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="jaxWsTest1Response">
    <xs:sequence>
      <xs:element name="return" type="xs:string" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="UserDefinedFault">
    <xs:sequence>
      <xs:element name="message" type="xs:string" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Change the name of the modified `TestJaxWsService.wsdl` to `input.wsdl`, and save under the `c:\temp\jaxws\works\fromwsdl\server\WEB-INF\wsdl\` directory.

4.3.2 Generating SEI

If you execute the `cjwsimport` command, the Java source, such as SEI that you require for developing Web Services, will be generated. For the `cjwsimport` command, see *14.1 cjwsimport command*.

The following is an execution example of the `cjwsimport` command:

```
> cd c:\temp\jaxws\works\fromwsdl\server\
> mkdir src\
> mkdir WEB-INF\classes\
> "%COSMINEXUS_HOME%\jaxws\bin\cjwsimport.bat" -generateService -s src -d WEB-INF
\classes WEB-INF\wsdl\input.wsdl
```

If the `cjwsimport` command is terminated successfully, the Java source is generated in the `c:\temp\jaxws\works\fromwsdl\server\src\com\example\sample\directory`. The directory path `com\example\sample\` (*directory-path-corresponding-to-the-package*) changes as per the coding of the namespace URI. For the mapping between namespace URIs and packages, see *15.1.1 Mapping a namespace to a package name*.

The following table lists and describes the products:

Table 4-3: Products during SEI generation (Starting from WSDL)

File name	Description
`JaxWsTest1.java`	This is the `JavaBean` class corresponding to 'Type referenced by the wrapper element of the request message' in the WSDL definition.
`JaxWsTest1Response.java`	This is the `JavaBean` class corresponding to 'Type referenced by the wrapper element of the response message' in the WSDL definition.
`ObjectFactory.java`	This is the `ObjectFactory` class of the JAXB 2.2 specifications.
`package-info.java`	This is the `package-info.java` file.
`TestJaxWs.java`	This is an SEI corresponding to the `TestJaxWsPort` type.
`TestJaxWsImpl.java`	This is a skeleton class corresponding to the `TestJaxWsPort` type.
`UserDefinedFault.java`	This is the `JavaBean` class (fault bean) corresponding to 'Type referenced by the wrapper element of the fault message' in the WSDL definition.
`UserDefinedException.java`	This is the wrapper exception class of the fault bean.

The file names `JaxWsTest1`, `TestJaxWs` and `TestJaxWsImpl` change according to the coding of the local name for operation, local name for port type, and local name for services. For the mapping between a local name for operation, local name for port type, and local name for services and Java sources, see *15. Mapping WSDL to Java*.

4.3.3 Creating a Web Service Implementation Class

Add Web Service processing to a skeleton class to create a Web Service Implementation Class. This subsection describes how to add the processing for returning the contents of the received request message along with the date information as the response message.

The following is an example for creating a Web Service Implementation Class:

```
package com.example.sample;

import java.util.Calendar;
import javax.jws.WebService;


@WebService(endpointInterface = "com.example.sample.TestJaxWs", targetNamespace =
"http://example.com/sample", serviceName = "TestJaxWsService", portName = "testJaxWs")
public class TestJaxWsImpl {

public String jaxWsTest1(String information, int count)
 throws UserDefinedException
 {
```

```
  Calendar today = Calendar.getInstance();
  StringBuffer result = new StringBuffer( 256 );
  result.append( "We've got your #" );
  result.append( new Integer( count ) );
  result.append( " message \"" );
  result.append( information );
  result.append( "\"! It's " );
  result.append( String.format( "%04d.%02d.%02d %02d:%02d:%02d", new Object[]{
   new Integer( today.get( Calendar.YEAR ) ),
   new Integer( today.get( Calendar.MONTH ) + 1 ),
   new Integer( today.get( Calendar.DAY_OF_MONTH ) ),
   new Integer( today.get( Calendar.HOUR_OF_DAY ) ),
   new Integer( today.get( Calendar.MINUTE ) ),
   new Integer( today.get( Calendar.SECOND ) ) } ) );
  result.append( " now. See ya!" );

  return result.toString();
  }
 }
```

The locations in italics are the implementation added for skeletons.

4.3.4 Compiling the Web Service Implementation Class

Use the `javac` command to compile the created Web Service Implementation Class.

The following is a compilation example:

```
> cd c:\temp\jaxws\works\fromwsdl\server\
> javac -encoding UTF-8 -cp "%COSMINEXUS_HOME%\jaxws\lib\cjjaxws.jar;%COSMINEXUS_HOME%
\CC\client\lib\j2ee-javax.jar;.\WEB-INF\classes" -d .\WEB-INF\classes src\com\example
\sample\TestJaxWsImpl.java
```

If the `javac` command is terminated successfully, `TestJaxWsImpl.class` in `c:\temp\jaxws\works\fromwsdl\server\WEB-INF\classes\com\example\sample\directory` is overwritten.

For the `javac` command, see the *JDK documentation*.

4.3.5 Creating web.xml

Create the `web.xml` file that is required as a WAR file component.

The following is an example for creating `web.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/
javaee/web-app_3_0.xsd">
 <description>Sample web service &amp;quot;fromwsdl&amp;quot;</description>
 <display-name>Sample_web_service_fromwsdl</display-name>
 <listener>
 <listener-class>
 com.cosminexus.xml.ws.transport.http.servlet.WSServletContextListener
 </listener-class>
 </listener>

<servlet>
 <description>Endpoint servlet for Cosminexus JAX-WS</description>
 <display-name>Endpoint_servlet_for_Cosminexus_JAX_WS</display-name>
 <servlet-name>CosminexusJaxwsServlet</servlet-name>
 <servlet-class>
 com.cosminexus.xml.ws.transport.http.servlet.WSServlet
 </servlet-class>
 <load-on-startup>1</load-on-startup>
 </servlet>
 <servlet-mapping>
 <servlet-name>CosminexusJaxwsServlet</servlet-name>
 <url-pattern>/TestJaxWsService</url-pattern>
 </servlet-mapping>
 <session-config>
 <session-timeout>60</session-timeout>
```

```
  </session-config>
 </web-app>
```

When creating `web.xml` of version 2.5, specify `2.5` in the `version` attribute of the `web-app` element and specify `http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd` as the second location information in the `xsd:schemaLocation` attribute.

The created `web.xml` will be stored in `c:\temp\jaxws\works\fromwsdl\server\WEB-INF\directory` with the UTF-8 format. For the `web.xml` settings, see *3.4 Creating web.xml*.

4.3.6 Creating application.xml

Create `application.xml` that is required as an EAR file component.

The following is an example for creating `application.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<application version="6" xmlns="http://java.sun.com/xml/ns/javaee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/
javaee/application_6.xsd">

 <description>Sample application &quot;fromwsdl&quot;</description>
 <display-name>Sample_application_fromwsdl</display-name>
 <module>
 <web>
 <web-uri>fromwsdl.war</web-uri>
 <context-root>fromwsdl</context-root>
 </web>
 </module>
</application>
```

When creating `web.xml` of version 5, specify `5` in the `version` attribute of the `application` element and specify `http://java.sun.com/xml/ns/javaee/application_5.xsd` as the second location information in the `xsd:schemaLocation` attribute.

The created `application.xml` will be stored in `c:\temp\jaxws\works\fromwsdl\server\META-INF\directory` with the UTF-8 format. For notes on creating `application.xml`, see *5.2.2 Notes on editing application.xml* in the *uCosminexus Application Server Application Development Guide*.

4.3.7 Creating EAR files

Use the `jar` command to create an EAR file.

The following is an example for creating an EAR file:

```
> cd c:\temp\jaxws\works\fromwsdl\server\
> jar cvf fromwsdl.war .\WEB-INF
> jar cvf fromwsdl.ear .\fromwsdl.war .\META-INF\application.xml
```

If the `jar` command is terminated successfully, `fromwsdl.ear` is created in `c:\temp\jaxws\works\fromwsdl\server\directory`.

For the `jar` command, see the *JDK documentation*.

4.4 Examples of deployment and startup (Starting from WSDL)

This section describes the examples of the deployment and the startup, starting from the WSDL.

4.4.1 Deploying EAR files

Use the `cjimportapp` command to deploy the created EAR file on the J2EE server.

The following is an example of deployment:

```
> cd c:\temp\jaxws\works\fromwsdl\server\
> "%COSMINEXUS_HOME%\CC\admin\bin\cjimportapp" jaxwsserver -nameserver
corbaname::testserver:900 -f fromwsdl.ear
```

For the `cjimportapp` command, see *cjimportapp* (*Importing J2EE applications*) in the *uCosminexus Application Server Command Reference Guide*.

For the method to deploy (import) J2EE applications by using the management portal, see *12.3.3 Importing J2EE applications* in the *uCosminexus Application Server Management Portal User Guide*.

4.4.2 Starting Web Services

Use the `cjstartapp` command to start Web Services.

The following is an example for starting a Web Service:

```
> cd c:\temp\jaxws\works\fromwsdl\server\
> "%COSMINEXUS_HOME%\CC\admin\bin\cjstartapp" jaxwsserver -nameserver
corbaname::testserver:900 -name Sample_application_fromwsdl
```

For the `cjstartapp` command, see *cjstartapp* (*Starting J2EE applications*) in the *uCosminexus Application Server Command Reference Guide*.

For the method to start J2EE applications by using the management portal, see *12.3.1 Starting J2EE applications* in the *uCosminexus Application Server Management Portal User Guide*.

4.5 Examples for deploying Web Service clients (Starting from WSDL)

This section describes the examples of the development of Web Service clients starting from the WSDL.

4.5.1 Generating a service class

If you execute the `cjwsimport` command, the Java source, such as a service class, required for Web Service client development is generated. For the `cjwsimport` command, see *14.1 cjwsimport command*.

The following is an execution example when developing Web Service clients in the same environment in which the Web Service is developed:

```
> cd c:\temp\jaxws\works\fromwsdl\client\
> mkdir src/
> mkdir classes/
> "%COSMINEXUS_HOME%\jaxws\bin\cjwsimport.bat" -s src -d classes ..\server\WEB-INF
\wsdl\input.wsdl
```

The following is an execution example when developing Web Service clients on an environment different from the environment in which the Web Service is developed:

```
> cd c:\temp\jaxws\works\fromwsdl\client\
> mkdir src/
> mkdir classes/
> "%COSMINEXUS_HOME%\jaxws\bin\cjwsimport.bat" -s src -d classes http://webhost:8085/
fromwsdl/TestJaxWsService?wsdl
```

When the execution is successful, the Java source will be generated in `c:\temp\jaxws\works\fromwsdl\client\src\com\example\sample\directory`. The directory path `com\example\sample\` (*directory-path-corresponding-to-the-package*) changes as per the coding of the namespace URI. For the mapping between namespace URIs and packages, see *15.1.1 Mapping a namespace to a package name*.

The following table lists and describes the products:

Table 4-4: Products during service class generation (Starting from WSDL)

File name	Description
`JaxWsTest1.java`	This is a `JavaBean` class corresponding to 'Type referenced by the wrapper element of the request message' in the WSDL definition.
`JaxWsTest1Response.java`	This is a `JavaBean` class corresponding to 'Type referenced by the wrapper element of the response message' in the WSDL definition.
`ObjectFactory.java`	This is the `ObjectFactory` class of the JAXB 2.2 specifications.
`package-info.java`	This is the `package-info.java` file.
`TestJaxWs.java`	This is an SEI corresponding to the `TestJaxWsPort` type.
`TestJaxWsService.java`	This is the service class.
`UserDefinedFault.java`	This is a `JavaBean` class corresponding to 'Type referenced by the wrapper element of the fault message' in the WSDL definition.
`UserDefinedException.java`	This is the wrapper exception class of the fault bean.

The file names `JaxWsTest1`, `TestJaxWs`, and `TestJaxWsService` change according to the coding of the local name for operation, local name for port type, and local name for services. For the mapping between a local name for operation, local name for port type, and local name for services and Java sources, see the following sections:

- *15.1.2 Mapping a port type to a SEI name*

- *15.1.3 Mapping an operation to a method name*
- *15.1.4 Mapping a message part to a parameter and return value (For wrapper style)*
- *15.1.5 Mapping the message part to the parameter and return value (For non-wrapper style)*

4.5.2 Creating an implementation class for the Web Service client

Create an implementation class for the Web Service client that uses the Web Service.

The following is an example for creating an implementation class of the Web Service client that invokes a Web Service once:

```
package com.example.sample.client;

import com.example.sample.TestJaxWs;
import com.example.sample.TestJaxWsService;
import com.example.sample.UserDefinedException;

public class TestClient {
 public static void main( String[] args ) {
 try {
  TestJaxWsService service = new TestJaxWsService();
  TestJaxWs port = service.getTestJaxWs();

  String returnValue = port.jaxWsTest1( "Invocation test.", 1003 );

  System.out.println( "[RESULT] " + returnValue );
 }
 catch( UserDefinedException e ){
  e.printStackTrace();
 }
 }

}
```

The created `TestClient.java` is stored in `c:\temp\jaxws\works\fromwsdl\client\src\com\example\sample\client\directory` with the UTF-8 format. The `com.example.sample`, `TestJaxWs`, `TestJaxWsService`, `TestJaxWs`, and `jaxWsTest1` change according to the package name, class name, and method name in the class of the generated Java source. For developing a Web Service with a different configuration, you must review, and if necessary revise, the coding of the package name, class name, and the method name in the class.

4.5.3 Compiling the implementation class for the Web Service client

Use the `javac` command to compile the created Web Service client.

The following is an example of compilation:

```
> cd c:\temp\jaxws\works\fromwsdl\client\
> javac -encoding UTF-8 -cp "%COSMINEXUS_HOME%\jaxws\lib\cjjaxws.jar;%COSMINEXUS_HOME%
\CC\client\lib\j2ee-javax.jar;%COSMINEXUS_HOME%\jaxp\lib\csmjaxb.jar;.\classes" -d .
\classes src\com\example\sample\client\TestClient.java
```

If the `javac` command is terminated successfully, the `TestClient.class` is generated in `c:\temp\jaxws\works\fromwsdl\client\classes\com\example\sample\client\directory`.

For the `javac` command, see the *JDK documentation*.

4.6 Examples for executing Web Services (Starting from WSDL)

This section describes the examples for the development of Web Service clients, starting from the WSDL.

4.6.1 Creating an option definition file for Java applications

Create an option definition file for Java applications (`usrconf.cfg`) required for executing Web Services.

The following is an example for creating the option definition file for Java applications:

```
add.class.path=Cosminexus-installation-directory\jaxws\lib\cjjaxws.jar
add.class.path=.\classes
ejb.client.log.directory=logs
add.jvm.arg=-Dcosminexus.home= Cosminexus-installation-directory
add.jvm.arg=-Dejbserver.server.prf.PRFID=PRF-ID
```

For *Cosminexus-installation-directory*, use the absolute path to specify the path on which Cosminexus is installed.

The created option definition file for Java applications is stored in `c:\temp\jaxws\works\fromwsdl\client\directory` with the UTF-8 format. For the option definition file for Java applications, see *14.2 usrconf.cfg (Option definition file for the Java applications)* in the *uCosminexus Application Server Definition Reference Guide*.

4.6.2 Creating a user property file for Java applications

Create a user property file for the Java applications required for executing Web Services.

Since the settings are not specially changed, create an empty file named *usrconf.properties* in `c:\temp\jaxws\works\fromwsdl\client\directory`. For the user property file for Java applications, see 14.3 *usrconf.properties* (*User property file for Java applications*) in the *uCosminexus Application Server Definition Reference Guide*.

4.6.3 Executing Web Service clients

Use the `cjclstartap` command to execute Web Service clients.

The following is an example for executing Web Service clients:

```
> cd c:\temp\jaxws\works\fromwsdl\client\
> "%COSMINEXUS_HOME%\CC\client\bin\cjclstartap" com.example.sample.client.TestClient
```

If the `cjclstartap` command is terminated successfully, the result of Web Service client execution is displayed. The following is an example for displaying execution results:

```
KDJE40053-I The cjclstartap command will now start. (directory for the user
definition file = c:\temp\jaxws\works\fromwsdl\client, PID = 2636)
--------------------
[RESULT] We've got your #1003 message "Invocation test."! It's 2007.11.28 14:50:50
now. See ya!
KDJE40054-I The cjclstartap command was stopped. (PID = 2636, exit status = 0)
```

The part in italics changes according to the execution timing and environment.

For the `cjclstartap` command, see *cjclstartap* (*Starting Java applications*) in the *uCosminexus Application Server Command Reference Guide*.

5 Examples for the Development Starting from SEI

This chapter describes the examples for the development of Web Services starting from SEI.

5.1 Configuration of development examples (Starting from SEI)

The development examples of this chapter describe how to develop Web Services starting from SEI.

The following table describes the configuration for Web Services to be developed:

Table 5-1: Web Service configuration (Starting from SEI)

No.	Item		Value
1	Name of the J2EE server to be deployed		`jaxwsserver`
2	Host name and port number of the Web server		`webhost:8085`
3	URL of the naming server		`corbaname::testserver:900`
4	Context root		`fromjava_dynamic_generate`
5	Style		`document/literal/wrapped`
6	Namespace URI		`http://sample.com`
7	Port type	Number	`1`
8		Local name	`AddNumbersImpl`
9	Operation	Number	`1`
10		Local name	`add`
11	Service	Number	`1`
12		Local name	`AddNumbersImplService`
13	Port	Number	`1`
14		Local name	`AddNumbersImplPort`
15	Web Service Implementation Class		`com.sample.AddNumbersImpl`
16	Method published in the Web Service Implementation Class	Number	`1`
17		Local name	`add`
18	Exception thrown with the methods of the Web Service implementation	Number	`1`
19		Local name	`com.sample.AddNumbersFault`

The following table describes the configuration of the current directory when developing Web Services:

Table 5-2: Configuration of the current directory (Starting from SEI)

Directory	Description
`c:\temp\jaxws\works\fromjava`	This is the current directory.
`server\`	Used for Web Service development.
`META-INF\`	Corresponds to the META-INF directory of the EAR file.
`application.xml`	Created in *5.3.4 Creating application.xml*.
`src\`	Stores the source file (`*.java`) for the Web Service. Used in *5.3.1 Creating Web Services Implementation Class* and *5.3.2 Compiling Web Services Implementation Class*.
`WEB-INF\`	Corresponds to the WEB-INF directory of the WAR file.

Directory	Description
web.xml	Created in *5.3.3 Creating web.xml*.
classes\	Stores the compiled class file (*.class). Created in *5.3.2 Compiling Web Services Implementation Class*.
wsdl\	Created in the section *5.3.5 Creating a WSDL file (Optional)*.
fromjava_dynamic_generate.ear	Created in *5.3.6 Creating EAR files*.
fromjava_dynamic_generate.war	
client\	Used for the development of the Web Service client.
src\	Stores the source file (*.java) of the Web Service client. Used in *5.5.1 Generating a service class* and *5.5.2 Creating an implementation class for the Web Service client*.
classes\	Stores the compiled class file (*.class). Used in *5.5.3 Compiling the implementation class for the Web Service client*.
usrconf.cfg	Created in *5.6.1 Creating an option definition file for Java applications*.
usrconf.properties	Created in *5.6.2 Creating a user property file for Java applications*.

Change the current directory path according to the environment to be developed.

Note that the directory and file names listed in the above table will be used in the description hereafter. The parts in *Bold* in the command execution examples and in the Java source indicate the specified values and the generated values that are used in this example. Read according to the environment that you want to build.

Also, in the development examples described in this chapter, the Web Service and Web Service client are developed in the same environment, but you can also develop them in separate environments. For developing Web Services and Web Service clients in different environments, read the current directory path suitable to the respective environments.

5.2 Example of development flow (Starting from SEI)

The development and execution flow described in the development examples of this chapter are as follows:

Developing a Web Service

1. Creating Web Services Implementation Class (*5.3.1*)
2. Compiling Web Services Implementation Class (*5.3.2*)
3. Creating `web.xml` (*5.3.3*)
4. Creating `application.xml` (*5.3.4*)
5. Creating a WSDL file (Optional) (*5.3.5*)
6. Creating an EAR file (*5.3.6*)

Deploying and starting

1. Deploying EAR files (*5.4.1*)
2. Starting Web Services (*5.4.2*)

Developing a Web Service client

1. Executing the `cjwsimport` command and generating a service class (*5.5.1*)
2. Creating the implementation class for Web Service clients (*5.5.2*)
3. Compiling the implementation class for Web Service clients (*5.5.3*)

Executing a Web Service

1. Creating the option definition file for Java applications (*5.6.1*)
2. Creating the user property file for Java applications (*5.6.2*)
3. Executing the Web Service client (*5.6.3*)

5.3 Examples of Web Service development (starting from SEI)

This section describes the examples for the development of Web Services, starting from SEI.

5.3.1 Creating Web Services Implementation Class

Create a new Web Service Implementation Class that codes the processing of the Web Service.

In this subsection, calculate the contents of the received request message and create a Web Service Implementation Class that returns the response message.

The following is an example for creating a Web Service Implementation Class in SOAP 1.1. The created `AddNumbersImpl.java` is stored in the `c:\temp\jaxws\works\fromjava\server\src\com\sample\` directory with the UTF-8 format.

```
package com.sample;

@javax.jws.WebService
public class AddNumbersImpl{

 public int add( int number1, int number2 ) throws AddNumbersFault{

  if( ( number1 < 0 ) || ( number2 < 0 ) ){
   throw new AddNumbersFault( "Negative number cannot be added!",
    "Numbers: " + number1 + ", " + number2 );
  }
  return number1 + number2;
 }

}
```

The exception class `com.sample.AddNumbersFault` thrown using `com.sample.AddNumbersImpl`will also be created. Normally, the creation of the exception class is optional, but an exception class is created in this subsection.

The following is an example of the creation of an exception class. The created `AddNumbersFault.java` is stored in the `c:\temp\jaxws\works\fromjava\server\src\com\sample\` directory with the UTF-8 format.

```
package com.sample;

public class AddNumbersFault extends Exception {

 String detail;

 public AddNumbersFault( String message, String detail ){
  super( message );
  this.detail = detail;
 }

 public String getDetail(){
  return detail;
 }
}
```

The following is an example of creating a Web Service Implementation Class for SOAP 1.2:

```
package com.sample;

@javax.jws.WebService
@javax.xml.ws.BindingType(javax.xml.ws.soap.SOAPBinding.SOAP12HTTP_BINDING)
public class AddNumbersImpl{

    public int add( int number1, int number2 ) throws AddNumbersFault{

        if( ( number1 < 0 ) || ( number2 < 0 ) ){
            throw new AddNumbersFault( "Negative number cannot be added!",
                "Numbers: " + number1 + ", " + number2 );
        }
```

```
        return number1 + number2;
    }

}
```

5.3.2 Compiling Web Services Implementation Class

Execute the `javac` command to compile Web Services Implementation Class. For details on the `javac` command, see the *JDK documentation*.

The following is an example of the execution of the `javac` command.

```
> cd c:\temp\jaxws\works\fromjava\server\
> mkdir .\WEB-INF\classes\
> javac -cp "%COSMINEXUS_HOME%\jaxws\lib\cjjaxws.jar;%COSMINEXUS_HOME%\CC\client\lib
\j2ee-javax.jar;%COSMINEXUS_HOME%\jaxp\lib\csmjaxb.jar;%COSMINEXUS_HOME%\jaxp\lib
\csmjaxp.jar;%COSMINEXUS_HOME%\jaxp\lib\csmstax.jar" -d WEB-INF\classes\ -s src src
\com\sample\AddNumbersImpl.java src\com\sample\AddNumbersFault.java
```

On successful execution of the `javac` command, the compiled classes are output to the following path:

`c:\temp\jaxws\works\fromjava\server\WEB-INF\classes\com\sample\directory`

You can run the `cjwsgen` command for the compiled Web Services Implementation Class to check errors in advance. For details on the `cjwsgen` command, see *14.3 cjwsgen command*, and for details on the error check, see *10.23(1) Using the cjwsgen command for checking errors*.

5.3.3 Creating web.xml

Create `web.xml` that is required as a WAR file component.

The following is an example for creating `web.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/
javaee/web-app_3_0.xsd">
 <description>Sample web service &quot;fromjava_dynamic_generate &quot;</description>
 <display-name>Sample_web_service_fromjava_dynamic_generate </display-name>
 <listener>
 <listener-class>
  com.cosminexus.xml.ws.transport.http.servlet.WSServletContextListener
 </listener-class>
 </listener>
 <servlet>
 <description>Endpoint servlet for Cosminexus JAX-WS</description>
 <display-name>Endpoint_servlet_for_Cosminexus_JAX_WS</display-name>
 <servlet-name>CosminexusJaxwsServlet</servlet-name>
 <servlet-class>
  com.cosminexus.xml.ws.transport.http.servlet.WSServlet
 </servlet-class>
 <load-on-startup>1</load-on-startup>
 </servlet>
 <servlet-mapping>
 <servlet-name>CosminexusJaxwsServlet</servlet-name>
 <url-pattern>/AddNumbersImplService</url-pattern>
 </servlet-mapping>
 <session-config>
 <session-timeout>60</session-timeout>
 </session-config>
</web-app>
```

When creating `web.xml` of version 2.5, specify `2.5` in the `version` attribute of the `web-app` element and specify `http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd` as the second location information in the `xsd:schemaLocation` attribute.

The created `web.xml` is stored in the `c:\temp\jaxws\works\fromjava\server\WEB-INF\directory` in UTF-8 format. For the `web.xml` settings, see *3.4 Creating web.xml*.

5.3.4 Creating application.xml

Create `application.xml` that is required as an EAR file component.

The following is an example for creating `application.xml`. Note that the items are set in `application.xml` because the Web Service do not exist.

```
<?xml version="1.0" encoding="UTF-8"?>
<application version="6" xmlns="http://java.sun.com/xml/ns/javaee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/
javaee/application_6.xsd">

 <description>Sample application &quot;fromjava_dynamic_generate&quot;</description>
 <display-name>Sample_application_fromjava_dynamic_generate </display-name>
 <module>
 <web>
  <web-uri> fromjava_dynamic_generate </web-uri>
  <context-root>fromjava_dynamic_generate </context-root>
 </web>
 </module>
</application>
```

When creating `web.xml` of version 5, specify `5` in the `version` attribute of the `application` element and specify `http://java.sun.com/xml/ns/javaee/application_5.xsd` as the second location information in the `xsd:schemaLocation` attribute.

The created `application.xml` is stored in the `c:\temp\jaxws\works\fromjava\server\META-INF\directory` in UTF-8 format. For notes on creating `application.xml`, see *5.2.2 Notes on editing application.xml* in the *uCosminexus Application Server Application Development Guide*.

5.3.5 Creating a WSDL file (Optional)

Creating a WSDL file during the development starting from SEI is optional, and if a file is created, include that file in an EAR file. This subsection describes an example of executing the WSDL generation functionality of the `cjwsgen` command to create a WSDL file from the already compiled Java sources. For the `cjwsgen` command, see *14.3 cjwsgen command*.

The following is an example of executing the `cjwsgen` command:

```
> cd c:\temp\jaxws\works\fromwsdl\server\
> mkdir .\WEB-INF\wsdl\
> mkdir .\temporary
> "%COSMINEXUS_HOME%\jaxws\bin\cjwsgen.bat" -r .\WEB-INF\wsdl -d .\temporary -cp .
\WEB-INF\classes com.sample.AddNumbersImpl
> rmdir /S /Q .\temporary
```

If the `cjwsgen` command is terminated successfully, the resource file is generated in the `c:\temp\jaxws\works\fromjava\WEB-INF\wsdl` directory. The following table lists the generated files:

Table 5-3: Files generated during the execution of the cjwsgen command

File name	Description
`AddNumbersImplService.wsdl`	This is a WSDL file corresponding to the specified Java source.
`AddNumbersImplService_schema1.xsd`	This is an XML Schema definition that is referenced from the WSDL file.

Delete the files generated in the `c:\temp\jaxws\works\fromjava\temporary` directory because the files are not required.

5.3.6 Creating EAR files

Use the `jar` command to create an EAR file containing the files created until now.

The following is an example for creating an EAR file:

```
> cd c:\temp\jaxws\works\fromjava\server\
> jar cvf fromjava_dynamic_generate.war .\WEB-INF
> jar cvf fromjava_dynamic_generate.ear .\fromjava_dynamic_generate.war .\META-INF
\application.xml
```

If the `jar` command is terminated successfully, fromjava_dynamic_generate.ear is created in the c:\temp\jaxws\works\fromjava\server\directory.

For the `jar` command, see the *JDK documentation*.

5.4 Examples of deployment and startup (Starting from SEI)

This section describes the examples for the deployment and the startup, starting from SEI.

5.4.1 Deploying EAR files

Use the `cjimportapp` command to deploy the created EAR file on a J2EE server.

The following is an example of deployment:

```
> cd c:\temp\jaxws\works\fromjava\server\
> "%COSMINEXUS_HOME%\CC\admin\bin\cjimportapp" jaxwsserver -nameserver
corbaname::testserver:900 -f fromjava_dynamic_generate.ear
```

For the `cjimportapp` command, see *cjimportapp (Importing J2EE applications)* in the *uCosminexus Application Server Command Reference Guide*.

For the method to deploy (import) J2EE applications by using the management portal, see *12.3.3 Importing J2EE applications* in the *uCosminexus Application Server Management Portal User Guide*.

5.4.2 Starting Web Services

Use the `cjstartapp` command to start the Web Service.

The following is an example for starting the Web Service:

```
> cd c:\temp\jaxws\works\fromjava\server\
> "%COSMINEXUS_HOME%\CC\admin\bin\cjstartapp" jaxwsserver -nameserver
corbaname::testserver:900 -name Sample_application_fromjava_dynamic_generate
```

For the `cjstartapp` command, see *cjstartapp (Starting J2EE applications)* in the *uCosminexus Application Server Command Reference Guide*.

For the method to start J2EE applications by using the management portal, see *12.3.1 Starting J2EE applications* in the *uCosminexus Application Server Management Portal User Guide*.

5.5 Examples of developing Web Service clients (Starting from SEI)

This section describes the examples for the development of Web Service clients, starting from SEI.

5.5.1 Generating a service class

If you execute the `cjwsimport` command, the Java source, such as service class, required for Web Service client development, will be generated. For the `cjwsimport` command, see *14.1 cjwsimport command*.

The following is an example for executing the `cjwsimport` command:

```
> cd c:\temp\jaxws\works\fromjava\client\
> mkdir src\
> mkdir classes\
> "%COSMINEXUS_HOME%\jaxws\bin\cjwsimport.bat" -s src -d classes http://webhost:8085/
fromjava_dynamic_generate /AddNumbersImplService?wsdl
```

If the `cjwsimport` command is terminated successfully, the Java source will be generated in the `c:\temp\jaxws\works\fromjava\client\src\com\sample\directory`. Note that `com\sample\`(*directory-path-corresponding-to-the-package*) changes as per the coding of the namespace URI. For the mapping between namespace URIs and packages, see *15.1.1 Mapping a namespace to a package name*.

The following table lists and describes the products:

Table 5-4: Products during service class generation (Starting from SEI)

File name	Description
`Add.java`	This is a `JavaBean` class corresponding to the type referenced by the wrapper element of the request message for 'Operation' in the WSDL definition.
`AddResponse.java`	This is a `JavaBean` class corresponding to the type referenced by the wrapper element of the response message for 'Operation' in the WSDL definition.
`ObjectFactory.java`	This is the `ObjectFactory` class of the JAXB 2.2 specifications.
`package-info.java`	This is the `package-info.java` file.
`AddNumbersImpl.java`	This is SEI corresponding to 'Service' in the WSDL definition.
`AddNumbersImplService.java`	This is the service class.
`AddNumbersFault.java`	This is the `JavaBean` class corresponding to `AddNumbersFault`.
`AddNumbersFault_Exception.java`	This is the wrapper exception class of the fault bean.

The file names `Add`, `AddNumbersImpl`, and `AddNumbersImplService` change according to the coding of the local name for operation, local name for port type, and local name for services. For the mapping of a local name for operation, local name for port type, and local name for services, see the following sections:

- *15.1.2 Mapping a port type to a SEI name*
- *15.1.3 Mapping an operation to a method name*
- *15.1.4 Mapping a message part to a parameter and return value (For wrapper style)*
- *15.1.5 Mapping a message part to a parameter and return value (For non-wrapper style)*

5.5.2 Creating an implementation class for the Web Service client

Create an implementation class for the Web Service client that uses Web Services.

The following is an example for creating Web Service clients that invokes Web Services once:

```
package com.sample.client;

import com.sample.AddNumbersImpl;
import com.sample.AddNumbersImplService;
import com.sample.AddNumbersFault_Exception;

public class TestClient {
 public static void main( String[] args ) {
  try {
   AddNumbersImplService service = new AddNumbersImplService();
   AddNumbersImpl port = service.getAddNumbersImplPort();

   int returnValue = port.add( 205, 103 );

   System.out.println( "[RESULT] " + returnValue );
  }
  catch( AddNumbersFault_Exception e ){
   e.printStackTrace();
  }
 }

}
```

The created `TestClient.java` is stored in the `c:\temp\jaxws\works\fromjava\client\src\com\sample\client\directory` in UTF-8 format.

Note that `com.sample`, `AddNumbersImpl`, `AddNumbersImplService`, `AddNumbersImplPort`, and `add` change according to the package name, class name, and method name in the class of the generated Java sources. When you want to develop a Web Service with a different configuration, you must review, and if necessary revise, the coding of the package name, class name, and the method name in the class.

5.5.3 Compiling the implementation class for the Web Service client

Use the `javac` command to compile the created Web Service client.

The following is an example of compilation:

```
> cd c:\temp\jaxws\works\fromjava\client\
> javac -encoding UTF-8 -cp "%COSMINEXUS_HOME%\jaxws\lib\cjjaxws.jar;%COSMINEXUS_HOME%
\CC\client\lib\j2ee-javax.jar;%COSMINEXUS_HOME%\jaxp\lib\csmjaxb.jar;.\classes" -d .
\classes src\com\sample\client\TestClient.java
```

If the `javac` command is terminated successfully, the `TestClient.class` is generated in the `c:\temp\jaxws\works\fromjava\client\classes\com\sample\client\directory`.

For the `javac` command, see the *JDK documentation*.

5.6 Examples for executing Web Services (Starting from SEI)

This section describes the example for the execution of Web Service clients, starting from SEI.

5.6.1 Creating an option definition file for Java applications

Create an option definition file for Java applications (`usrconf.cfg`) required for executing Web Services.

The following is an example for creating the option definition file for Java applications:

```
add.class.path= Cosminexus-installation-directory\jaxws\lib\cjjaxws.jar
add.class.path=.\classes
ejb.client.log.directory=logs
add.jvm.arg=-Dcosminexus.home= Cosminexus-installation-directory
add.jvm.arg=-Dejbserver.server.prf.PRFID=PRF-ID
```

For the *Cosminexus-installation-directory* part, use the absolute path to specify the path where Cosminexus is installed.

The created option definition file for Java applications is stored in the `c:\temp\jaxws\works\fromjava\client\directory`. For the option definition file for Java applications, see *14.2 usrconf.cfg (Option definition file for Java applications)* in the *uCosminexus Application Server Definition Reference Guide*.

5.6.2 Creating a user property file for Java applications

Create a user property file for Java applications required for executing a Web Service.

Since the settings are not specially changed, create an empty file named `usrconf.properties` in `c:\temp\jaxws\works\fromjava\client\directory`. For the user property file for Java applications, see *14.3 usrconf.properties (User property file for Java applications)* in the *uCosminexus Application Server Definition Reference Guide*.

5.6.3 Executing Web Service clients

Use the `cjclstartap` command to execute Web Service client.

The following is an example for executing Web Service client:

```
> cd c:\temp\jaxws\works\fromjava\client\
> "%COSMINEXUS_HOME%\CC\client\bin\cjclstartap" com.sample.client.TestClient
```

If the `cjclstartap` command is terminated successfully, the results of Web Service client execution are displayed. Following is an example for displaying the execution results:

```
KDJE40053-I The cjclstartap command will now start. (directory for the user
definition file = c:\temp\jaxws\works\fromjava\client, PID = 2636)
[RESULT] 308
KDJE40054-I The cjclstartap command was stopped. (PID = 2636, exit status = 0)
```

The part in italics changes according to the execution timing and environment.

For the `cjclstartap` command, see *cjclstartap (Starting Java applications)* in the *uCosminexus Application Server Command Reference Guide*.

6 Examples of Development Starting from SEI (Using the cjwsgen Command)

This chapter describes the examples for developing Web Services using the `cjwsgen` command, with SEI as the starting point.

6.1 Configuration of development examples (starting from SEI and the cjwsgen command)

In the development examples described in this chapter, a Web Service starting from SEI is developed using the `cjwsgen` command provided by Cosminexus. For the method to use the `cjwsgen` command, see *14.3 cjwsgen command*.

The following table shows the configuration of the Web Service to be developed:

Table 6-1: Web Service configuration (Starting from SEI or the cjwsgen command)

No.	Item		Value
1	Name of the J2EE server to be deployed		`jaxwsserver`
2	Host name and port number of the Web server		`webhost:8085`
3	URL of the naming server		`corbaname::testserver:900`
4	Context root		`wsgen`
5	Style		`document/literal/wrapped`
6	Namespace URI		`http://example.org/sample`
7	Port type	Number	`1`
8		Local name	`TestJaxWs`
9	Operation	Number	`1`
10		Local name	`jaxWsTest1`
11	Service	Number	`1`
12		Local name	`TestJaxWsService`
13	Port	Number	`1`
14		Local name	`testJaxWs`
15	Web Service Implementation Class		`com.sample.AddNumbersImpl`
16	Method published in the Web Service Implementation Class	Number	`1`
17		Local name	`add`
18	Exception thrown in the methods in Web Service implementation	Number	`1`
19		Local name	`com.sample.AddNumbersFault`

The following table shows the configuration of the current directory when developing Web Services.

Table 6-2: Configuration of the current directory (Starting from SEI or the cjwsgen command)

Directory	Description
`c:\temp\jaxws\works\annotations`	This is the current directory.
`server\`	Used for Web Service development.
`META-INF\`	Corresponds to the META-INF directory of the EAR file.
`application.xml`	Created in *6.3.4 Creating application.xml*.
`src\`	Stores the source file (`*.java`) for the Web Service.

Directory	Description
src\	Used in *7.3.1 Creating a Web Service Implementation Class* and *7.3.2 Generating Java sources*.
WEB-INF\	Corresponds to the WEB-INF directory of the WAR file.
web.xml	Created in *6.3.3 Creating web.xml*.
classes\	Stores the compiled class file (*.class). Used in *6.3.2 Generating Java sources*.
wsdl\	Created in *6.3.2 Generating Java sources*.
wsgen.ear	Created in *6.3.5 Creating EAR files*.
wsgen.war	
client\	Used for the development of the Web Service client.
src\	Stores the source file (*.java) of the Web Service client. Used in *6.5.1 Creating service classes* and *6.5.2 Creating implementation classes for Web Service clients*.
classes\	Stores the compiled class file (*.class). Used in *6.5.3 Compiling implementation classes for Web Service clients*.
usrconf.cfg	Created in *6.6.1 Creating option definition files for Java applications*.
usrconf.properties	Created in *6.6.2 Creating user property files for Java applications*.

Change the current directory path according to the environment to be developed.

Note that the directory and file names listed in the above table will be used in the description hereafter. The parts in *Bold* in the command execution examples and in the Java source indicate the specified values and the generated values that are used in examples. Read according to the environment you want to build.

Also, in the development examples described in this chapter, the Web Service and Web Service client will be developed in the same environment, but you can also develop them in different environments. For developing Web Services and Web Service clients in different environments, read the current directory path suitable to the respective environment.

6.2 Flow of development examples (starting from SEI and the cjwsgen command)

In the development examples described in this chapter, flow of customization and execution is as follows:

Developing a Web Service

1. Saving the already compiled class files (*6.3.1*)
2. Generating an additional Java code by executing the `cjwsgen` command, and also generating a WSDL, if required (*6.3.2*)
3. Creating `web.xml` (*6.3.3*)
4. Creating `application.xml` (*6.3.4*)
5. Creating an EAR file (*6.3.5*)

Deploying and starting

1. Deploying EAR files (*6.4.1*)
2. Starting Web Services (*6.4.2*)

Developing a Web Service client

1. Executing the `cjwsimport` command and generating a service class (*6.5.1*)
2. Creating the implementation class for Web Service clients (*6.5.2*)
3. Compiling the implementation class for Web Service clients (*6.5.3*)

Executing a Web Service

1. Creating the option definition file for Java applications (*6.6.1*)
2. Creating the user property file for Java applications (*6.6.2*)
3. Executing the Web Service client (*6.6.3*)

6.3 Development example of Web Services (Starting from SEI and the cjwsgen command)

This section describes an example for developing Web Services using the `cjwsgen` command, with SEI as the starting point.

6.3.1 Saving the already compiled class files (starting from SEI and the cjwsgen command)

Save the already compiled Web Service Implementation Class `AddNumbersImpl.class` in the `c:\temp\jaxws\works\wsgen\server\WEB-INF\classes\com\sample\` directory. Save the already compiled exception class `AddNumbersFault.class` in the `c:\temp\jaxws\works\wsgen\server\WEB-INF\classes\com\sample\` directory.

6.3.2 Generating Java sources (starting from SEI and the cjwsgen command)

Execute the `cjwsgen` command to add the Java code required for developing Web Services, and if required, generate resource files (WSDL and XML Schema definition) that indicate the meta information of the Web Services.

The following is an example for executing the `cjwsgen` command to generate resource files:

```
> cd c:\temp\jaxws\works\wsgen\server\
> "%COSMINEXUS_HOME%\jaxws\bin\cjwsgen.bat" -r WEB-INF\wsdl -d WEB-INF\classes -cp
WEB-INF\classes com.sample.AddNumbersImpl
```

If the `cjwsgen` command is terminated successfully, Java source is generated in the `c:\temp\jaxws\works\wsgen\server\WEB-INF\classes\com\sample\` directory. Note that `com\sample\` (directory path corresponding to the package) changes depending on the coding of namespace URIs. For the mapping between namespace URIs and packages, see *15.1.1 Mapping a namespace to a package name*.

The following table lists the generated files:

Table 6-3: Files generated during the generation of Java sources (Starting from SEI or the cjwsgen command)

File name	Description
`Add.java`	This is the `JavaBean` class corresponding to the type referenced by the `wrapper` element of the request message in `Operation` of the WSDL definition.
`AddResponse.java`	This is the `JavaBean` class corresponding to the type referenced by the `wrapper` element of the response message in `Operation` of the WSDL definition.
`AddNumbersFaultBean.java`	This is the `JavaBean` class corresponding to `AddNumbersFault` of the exception class.

The file names `Add` and `AddNumbersFault` change according to the coding of the method name and local name of the port type published in Web Services Implementation Class, and the class names of the exceptions to be thrown in the Web Services Implementation Class. For mapping local names of the operations, see *15.1.3 Mapping an operation to a method name*.

Note that the resource files are generated in the `c:\temp\jaxws\works\wsgen\WEB-INF\wsdl\` directory.

6.3.3 Creating web.xml (starting from SEI and the cjwsgen command)

Create `web.xml` that is required as a WAR file component.

The following is an example for creating `web.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/
javaee/web-app_3_0.xsd">
 <description>Sample web service &quot;wsgen&quot;</description>
 <display-name>Sample_web_service_wsgen</display-name>
 <listener>
  <listener-class>
   com.cosminexus.xml.ws.transport.http.servlet.WSServletContextListener
  </listener-class>
 </listener>
 <servlet>
  <description>Endpoint servlet for Cosminexus JAX-WS</description>
  <display-name>Endpoint_servlet_for_Cosminexus_JAX_WS</display-name>
  <servlet-name>CosminexusJaxwsServlet</servlet-name>
  <servlet-class>
   com.cosminexus.xml.ws.transport.http.servlet.WSServlet
  </servlet-class>
  <load-on-startup>1</load-on-startup>
 </servlet>
 <servlet-mapping>
  <servlet-name>CosminexusJaxwsServlet</servlet-name>
  <url-pattern>/TestJaxWsService</url-pattern>
 </servlet-mapping>
 <session-config>
  <session-timeout>60</session-timeout>
 </session-config>
</web-app>
```

When creating `web.xml` of the version 2.5, specify `2.5` in the `version` attribute of the `web-app` element and specify `http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd` as the second location information in the `xsd:schemaLocation` attribute.

The created `web.xml` is stored in the `c:\temp\jaxws\works\wsgen\server\WEB-INF\` directory with the UTF-8 format. For the `web.xml` settings, see *3.4 Creating web.xml*.

6.3.4 Creating application.xml

Create `application.xml` that is required as an EAR file component.

The following is an example for creating `application.xml`. Note that the items are set in `application.xml` because a Web Service does not exist.

```
<?xml version="1.0" encoding="UTF-8"?>
<application version="6" xmlns="http://java.sun.com/xml/ns/javaee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/
javaee/application_6.xsd">

 <description>Sample application &quot;wsgen&quot;</description>
 <display-name>Sample_application_wsgen</display-name>
 <module>
  <web>
   <web-uri>wsgen.war</web-uri>
   <context-root>wsgen</context-root>
  </web>
 </module>
</application>
```

When creating `web.xml` of the version 5, specify `5` in the `version` attribute of the `application` element and specify `http://java.sun.com/xml/ns/javaee/application_5.xsd` as the second location information in the `xsd:schemaLocation` attribute.

The created `application.xml` is stored in the `c:\temp\jaxws\works\wsgen\server\META-INF\directory` in UTF-8 format. For notes on creating `application.xml`, see *5.2.2 Notes on editing application.xml* in the *uCosminexus Application Server Application Development Guide*.

6.3.5 Creating EAR files

Use the `jar` command to create an EAR file containing the files created until now.

```
> cd c:\temp\jaxws\works\annotations\server\
> jar cvf wsgen.war .\WEB-INF
> jar cvf wsgen.ear .\wsgen.war .\META-INF\application.xml
```

If the `jar` command is terminated successfully, `wsgen.ear` is created in the `c:\temp\jaxws\works\wsgen\server\` directory.

For the `jar` command, see the *JDK documentation*.

6.4 Examples of deployment and startup (Starting from SEI and the cjwsgen command)

This section describes the examples of deployment and startup using the `cjwsgen` command, with SEI as the starting point.

6.4.1 Deploying EAR files

Use the `cjimportapp` command to deploy the created EAR file on the J2EE server.

The following is an example of deployment:

```
> cd c:\temp\jaxws\works\wsgen\server\
> "%COSMINEXUS_HOME%\CC\admin\bin\cjimportapp" jaxwsserver -nameserver
corbaname::testserver:900 -f annotations.ear
```

For the `cjimportapp` command, see *cjimportapp (Importing J2EE applications)* in the *uCosminexus Application Server Command Reference Guide*.

For the method to deploy (import) J2EE applications by using the management portal, see *12.3.3 Importing J2EE applications* in the *uCosminexus Application Server Management Portal User Guide*.

6.4.2 Starting Web Services

Use the `cjstartapp` command to start the Web Service.

The following is an example for starting the Web Service:

```
> cd c:\temp\jaxws\works\wsgen\server\
> "%COSMINEXUS_HOME%\CC\admin\bin\cjstartapp" jaxwsserver -nameserver
corbaname::testserver:900 -name Sample_application_annotations
```

For the `cjstartapp` command, see *cjstartapp (Starting J2EE applications)* in the *uCosminexus Application Server Command Reference Guide*.

For the method to start J2EE applications by using the management portal, see *12.3.1 Starting J2EE applications* in the *uCosminexus Application Server Management Portal User Guide*.

6.5 Development examples of Web Services clients (Starting from SEI and the cjwsgen command)

This section describes the examples for developing a Web Service client using the `cjwsgen` command, with SEI as the starting point.

6.5.1 Creating service classes

If you execute the `cjwsimport` command, the Java source, such as service class, required for Web Service client development, will be created. For the `cjwsimport` command, see *14.1 cjwsimport command*.

The following is an example of the execution of the `cjwsimport` command:

```
> cd c:\temp\jaxws\works\wsgen\client\
> mkdir src\
> mkdir classes\
> "%COSMINEXUS_HOME%\jaxws\bin\cjwsimport.bat" -s src -d classes http://webhost:8085/
wsgen/AddNumbersImplService?wsdl
```

If the `cjwsimport` command is terminated successfully, the Java source is created in the `c:\temp\jaxws\works\wsgen\client\src\com\example\sample\directory`. Note that `com\example\sample\` (*directory-path-corresponding-to-the-package*) changes as per the coding of the namespace URI. For the mapping between namespace URIs and packages, see *15.1.1 Mapping a namespace to a package name*.

The following table lists the products:

Table 6-4: Products during service class generation (Starting from SEI and the cjwsgen command)

File name	Description
`JaxWsTest1.java`	This is a `JavaBean` class corresponding to type referenced by the wrapper element of the request message for 'Operation' in the WSDL definition.
`JaxWsTest1Response.java`	This is a `JavaBean` class corresponding to type referenced by the wrapper element of the response message for 'Operation' in the WSDL definition.
`ObjectFactory.java`	This is the `ObjectFactory` class of the JAXB 2.2 specifications.
`package-info.java`	This is the `package-info.java` file.
`TestJaxWs.java`	This is SEI corresponding to 'Service' in the WSDL definition.
`TestJaxWsService.java`	This is the service class.
`AddNumbersFault.java`	This is the `JavaBean` class corresponding to `AddNumbersFault`.
`AddNumbersFault_Exception.java`	This is the wrapper exception class of the fault bean.

The file names `JaxWsTest1`, `TestJaxWs` and `TestJaxWsService` change according to the coding of the local name for operation, local name for port type, and local name for services. For the mapping of a local name for operation, local name for port type, and local name for services see the following sections:

- *15.1.2 Mapping a port type to a SEI name*
- *15.1.3 Mapping an operation to a method name*
- *15.1.4 Mapping the message part to the parameter and the return value (For wrapper style)*
- *15.1.5 Mapping the message part to the parameter and the return value (For non-wrapper style)*

6.5.2 Creating Implementation Classes for Web Services clients

Create an implementation class for the Web Service client that uses the Web Service.

The following is an example for creating a Web Service client that invokes the Web Service once:

```
package org.example.sample.client;

import org.example.sample.TestJaxWs;
import org.example.sample.TestJaxWsService;
import org.example.sample.AddNumbersFault_Exception;

public class TestClient {
  public static void main( String[] args ) {
    try {
      TestJaxWsService service = new TestJaxWsService();
      TestJaxWs port = service.getTestJaxWs();

      int returnValue = port.jaxWsTest1( 205, 103 );

      System.out.println( "[RESULT] " + returnValue );
    }
    catch( AddNumbersFault_Exception e ){
      e.printStackTrace();
    }
  }

}
```

The created `TestClient.java` is stored in the `c:\temp\jaxws\works\wsgen\client\src\com\example\sample\client\` directory with the UTF-8 format.

Note that `org.example.sample`, `TestJaxWs`, `TestJaxWsService`, and `jaxWsTest1` change according to the package name, class name, and method name in the class of the generated Java source. When you want to develop a Web Service with a different configuration, you must review and if necessary revise the coding of the package name, class name, and the method name in the class.

6.5.3 Compiling Implementation Classes for Web Services clients

Use the `javac` command to compile the created Web Services client.

The following is an example of compilation:

```
> cd c:\temp\jaxws\works\wsgen\client\
> javac -encoding UTF-8 -cp "%COSMINEXUS_HOME%\jaxws\lib\cjjaxws.jar;%COSMINEXUS_HOME%
\CC\client\lib\j2ee-javax.jar;%COSMINEXUS_HOME%\jaxp\lib\csmjaxb.jar;.\classes" -d .
\classes src\org\example\sample\client\TestClient.java
```

If the `javac` command is terminated successfully, the `TestClient.class` is generated in the `c:\temp\jaxws\works\wsgen\client\classes\org\example\sample\client\directory`.

For the `javac` command, see the *JDK documentation*.

6.6 Examples of executing Web Services (starting from SEI and the cjwsgen command)

This section describes the examples of executing Web Service clients using the `cjwsgen` command, with SEI as the starting point.

6.6.1 Creating option definition files for Java applications

Create an option definition file for Java applications (`usrconf.cfg`) required for executing Web Services.

The following is an example for creating the option definition file for Java applications:

```
add.class.path=Cosminexus-installation-directory\jaxws\lib\cjjaxws.jar
add.class.path=.\classes
ejb.client.log.directory=logs
add.jvm.arg=-Dcosminexus.home=Cosminexus-installation-directory
add.jvm.arg=-Dejbserver.server.prf.PRFID=PRF-ID
```

For the *Cosminexus-installation-directory* part, use the absolute path to specify the path where *Cosminexus* is installed.

The created option definition file for Java applications is stored in the `c:\temp\jaxws\works\annotations\client\directory`. For the option definition file for Java applications, see *14.2 usrconf.cfg (Option definition file for Java applications)* in the *uCosminexus Application Server Definition Reference Guide*.

6.6.2 Creating user property files for Java applications

Create a user property file for Java applications required for executing a Web Service.

Since the settings are not specially changed, create an empty file named `usrconf.properties` in `c:\temp\jaxws\works\wsgen\client\directory`. For the user property file for Java applications, see *14.3 usrconf.properties (User property file for Java applications)* in the *uCosminexus Application Server Definition Reference Guide*.

6.6.3 Executing Web Services clients

Use the `cjclstartap` command to execute Web Service client.

The following is an example for executing a Web Service client:

```
> cd c:\temp\jaxws\works\wsgen\client\
> "%COSMINEXUS_HOME%\CC\client\bin\cjclstartap" org.example.sample.client.TestClient
```

If the `cjclstartap` command is terminated successfully, the results of Web Service client execution are displayed. Following is an example for displaying the execution results:

```
KDJE40053-I The cjclstartap command will now start. (directory for the user
definition file = c:\temp\jaxws\works\wsgen\client, PID = 2636)
[RESULT] 308
KDJE40054-I The cjclstartap command was stopped. (PID = 2636, exit status = 0)
```

The part in italics changes according to the execution timing and environment.

For the `cjclstartap` command, *see cjclstartap (Starting Java applications)* in the *uCosminexus Application Server Command Reference Guide*.

7 Examples of Development Starting from SEI (For Customization)

This chapter describes examples of the customization of Web Services starting from SEI.

7.1 Configuration examples for development (Starting from SEI and customization)

Referencing the development examples described in this chapter, develop the Web Services starting from SEI. Use annotations to customize the Web Service to be developed.

The following table describes the configuration of the Web Service to be developed:

Table 7-1: Web Service configuration (Starting from SEI and customization)

No.	Item		Value
1	Name of the J2EE server to be deployed		`jaxwsserver`
2	Host name and port number of the Web server		`webhost:8085`
3	URL of the naming server		`corbaname::testserver:900`
4	Context root		`annotations_dynamic_generate`
5	Style		`document/literal/wrapped`
6	Namespace URI		`http://example.org/sample`
7	Port type	Number	`1`
8		Local name	`TestJaxWs`
9	Operation	Number	`1`
10		Local name	`jaxWsTest1`
11	Service	Number	`1`
12		Local name	`TestJaxWsService`
13	Port	Number	`1`
14		Local name	`testJaxWs`
15	Web Service Implementation Class		`com.sample.AddNumbersImpl`
16	Method published in the Web Service Implementation Class	Number	`1`
17		Local name	`add`
18	Exception thrown in the methods in Web Service implementation	Number	`1`
19		Local name	`com.sample.AddNumbersFault`

The following table describes the configuration of the current directory when developing Web Services.

Table 7-2: Configuration of the current directory (Starting from SEI or customization)

Directory	Description
`c:\temp\jaxws\works\annotations`	This is the current directory.
`server\`	Used for Web Service development.
`META-INF\`	Corresponds to the META-INF directory of the EAR file.
`application.xml`	Created in *7.3.4 Creating application.xml*.
`src\`	Stores the source file (`*.java`) for the Web Service. Used in *7.3.1 Creating Web Services Implementation Class* and *7.3.2 Compiling Web Services Implementation Class*.

Directory	Description
WEB-INF\	Corresponds to the WEB-INF directory of the WAR file.
web.xml	Created in *7.3.3 Creating web.xml*.
classes\	Stores the compiled class file (`*.class`). Used in *7.3.2 Compiling Web Services Implementation Class*.
annotations_dynamic_generate.ear	Created in *7.3.5 Creating an EAR file*.
annotations_dynamic_generate.war	
client\	Used for the development of the Web Service client.
src\	Stores the source file (`*.java`) of the Web Service client. Used in *7.5.1 Creating a service class* and *7.5.2 Creating an implementation class for the Web Service client*.
classes\	Stores the compiled class file (`*.class`). Used in *7.5.3 Compiling the implementation class for the Web Service client*.
usrconf.cfg	Created in *7.6.1 Creating an option definition file for Java applications*.
usrconf.properties	Created in *7.6.2 Creating a user property file for Java applications*.

Change the current directory path according to the environment to be developed.

Note that the directory and file names listed in the above table will be used in the description hereafter. The parts in *bold* in the command execution examples and in the Java source indicate the specified values and the generated values that are used in examples. Read according to the environment you want to build.

Also, in the development examples described in this chapter, the Web Service and Web Service client will be developed in the same environment, but you can also develop them in different environments. For developing Web Services and Web Service clients in different environments, read the current directory path suitable to the respective environments.

7.2 Flow of development examples (Starting from SEI and customization)

In the development examples described in this chapter, the flow of customization and execution is as follows:

Developing Web Services

1. Creating Web Services Implementation Class (*7.3.1*)
2. Compiling Web Services Implementation Class (*7.3.2*)
3. Creating `web.xml` (*7.3.3*)
4. Creating `application.xml` (*7.3.4*)
5. Creating an EAR file (*7.3.5*)

Deploying and starting

1. Deploying EAR files (*7.4.1*)
2. Starting Web Services (*7.4.2*)

Developing a Web Service client

1. Executing the `cjwsimport` command and generating a service class (*7.5.1*)
2. Creating the implementation class for Web Service clients (*7.5.2*)
3. Compiling the implementation class for Web Service clients (*7.5.3*)

Executing a Web Service

1. Creating the option definition file for Java applications (*7.6.1*)
2. Creating the user property file for Java applications (*7.6.2*)
3. Executing the Web Service client (*7.6.3*)

7.3 Examples of developing Web Services (Starting from SEI or customization)

This section describes examples of the development of Web Services (customized) starting from SEI.

7.3.1 Creating Web Services Implementation Class

Create a Web Service Implementation Class that codes the processing of the Web Service. In this subsection, calculate the contents of the received request message and create the Web Service Implementation Class that returns the response message.

The following is an example of creating a Web Service Implementation Class for SOAP 1.1. The created `AddNumbersImpl.java` file is saved in the `c:\temp\jaxws\works\annotations\server\src\com\sample\` directory with the UTF-8 format.

```
package com.sample;

@javax.jws.WebService(name = "TestJaxWs",
  targetNamespace = "http://example.org/sample",
  serviceName = "TestJaxWsService", portName = "testJaxWs")
@javax.xml.ws.BindingType(javax.xml.ws.soap.SOAPBinding.SOAP11HTTP_BINDING)
@javax.jws.soap.SOAPBinding(style = javax.jws.soap.SOAPBinding.Style.DOCUMENT, use =
javax.jws.soap.SOAPBinding.Use.LITERAL)
public class AddNumbersImpl{

  @javax.jws.WebMethod(operationName = "jaxWsTest1")
  @javax.jws.WebResult(name = "return")
  public int add( @javax.jws.WebParam(name="num1")int number1,
@javax.jws.WebParam(name="num2")int number2 ) throws AddNumbersFault{

    if( ( number1 < 0 ) || ( number2 < 0 ) ){
      throw new AddNumbersFault( "Negative number cannot be added!",
        "Numbers: " + number1 + ", " + number2 );
    }
    return number1 + number2;
  }

}
```

The exception class `com.sample.AddNumbersFault` thrown using `com.sample.AddNumbersImpl`will also be created. Normally, the creation of the exception class is optional, but an exception class will be created here.

The following is an example of creating an exception class. The created `AddNumbersFault.java` file is saved in the `c:\temp\jaxws\works\annotations\server\src\com\sample\` directory with the UTF-8 format.

```
package com.sample;

public class AddNumbersFault extends Exception {

  String detail;

  public AddNumbersFault( String message, String detail ){
    super( message );
    this.detail = detail;
  }

  public String getDetail(){
    return detail;
  }
}
```

The following is an example of creating a Web Service Implementation Class for SOAP 1.2:

```
package com.sample;

@javax.jws.WebService(name = "TestJaxWs",
    targetNamespace = "http://example.org/sample",
    serviceName = "TestJaxWsService", portName = "testJaxWs")
@javax.xml.ws.BindingType(javax.xml.ws.soap.SOAPBinding.SOAP12HTTP_BINDING)
```

```
@javax.jws.soap.SOAPBinding(style = javax.jws.soap.SOAPBinding.Style.DOCUMENT, use =
javax.jws.soap.SOAPBinding.Use.LITERAL)
public class AddNumbersImpl{

    @javax.jws.WebMethod(operationName = "jaxWsTest1")
    @javax.jws.WebResult(name = "return")
    public int add( @javax.jws.WebParam(name="num1")int number1,
@javax.jws.WebParam(name="num2")int number2 ) throws AddNumbersFault{

        if( ( number1 < 0 ) || ( number2 < 0 ) ){
            throw new AddNumbersFault( "Negative number cannot be added!",
                "Numbers: " + number1 + ", " + number2 );
        }
        return number1 + number2;
    }

}
```

7.3.2 Compiling Web Services Implementation Class

Execute the `javac` command to compile Web Services Implementation Class. For details on the `javac` command, see the *JDK documentation*.

The following is an example of the execution of the `javac` command.

```
> cd c:\temp\jaxws\works\annotations\server\
> mkdir .\WEB-INF\classes\
> javac -cp "%COSMINEXUS_HOME%\jaxws\lib\cjjaxws.jar;%COSMINEXUS_HOME%\CC\client\lib
\j2ee-javax.jar;%COSMINEXUS_HOME%\jaxp\lib\csmjaxb.jar;%COSMINEXUS_HOME%\jaxp\lib
\csmjaxp.jar;%COSMINEXUS_HOME%\jaxp\lib\csmstax.jar" -d WEB-INF\classes\ -s src src
\com\sample\AddNumbersImpl.java src\com\sample\AddNumbersFault.java
```

The compiled classes are output to `c:\temp\jaxws\works\addressing\server\WEB-INF\classes\com\sample\directory`, if the `javac` command successfully ends. When you execute the `cjwsgen` command for the compiled Web Services Implementation Class, you can check for errors in advance. For details on the `cjwsgen` command, see *14.3 cjwsgen command*, and for details on the error check, see *10.23 (1) Using the cjwsgen command for checking errors*.

7.3.3 Creating web.xml

Create the `web.xml` file that is required as a WAR file component.

The following is an example of creating `web.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/
javaee/web-app_3.0.xsd">
 <description>Sample web service &quot;annotations_dynamic_generate &quot;</
description>
 <display-name>Sample_web_service_annotations_dynamic_generate </display-name>
 <listener>
  <listener-class>
   com.cosminexus.xml.ws.transport.http.servlet.WSServletContextListener
  </listener-class>
 </listener>
 <servlet>
  <description>Endpoint servlet for Cosminexus JAX-WS</description>
  <display-name>Endpoint_servlet_for_Cosminexus_JAX_WS</display-name>
  <servlet-name>CosminexusJaxwsServlet</servlet-name>
  <servlet-class>
   com.cosminexus.xml.ws.transport.http.servlet.WSServlet
  </servlet-class>
  <load-on-startup>1</load-on-startup>
 </servlet>
 <servlet-mapping>
  <servlet-name>CosminexusJaxwsServlet</servlet-name>
  <url-pattern>/TestJaxWsService</url-pattern>
 </servlet-mapping>
 <session-config>
```

```
   <session-timeout>60</session-timeout>
  </session-config>
 </web-app>
```

When creating `web.xml` of version 2.5, specify `2.5` in the `version` attribute of the `web-app` element and specify `http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd` as the second location information in the `xsd:schemaLocation` attribute.

The created `web.xml` is saved in the `c:\temp\jaxws\works\annotations\server\WEB-INF\` directory with the UTF-8 format. For the `web.xml` settings, see the section *3.4 Creating web.xml*.

7.3.4 Creating application.xml

Create the `application.xml` file that is required as an EAR file component.

The following is an example of creating `application.xml`. Note that the items are set in `application.xml` because a Web Service does not exist.

```
 <?xml version="1.0" encoding="UTF-8"?>
 <application version="6" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/
javaee/application_6.xsd">

  <description>Sample application &quot;annotations_dynamic_generate &quot;</
description>
  <display-name>Sample_application_annotations_dynamic_generate </display-name>
  <module>
   <web>
    <web-uri>annotations_dynamic_generate.war</web-uri>
    <context-root>annotations_dynamic_generate </context-root>
   </web>
  </module>
 </application>
```

When creating `web.xml` of version 5, specify `5` in the `version` attribute of the `application` element and specify `http://java.sun.com/xml/ns/javaee/application_5.xsd` as the second location information in the `xsd:schemaLocation` attribute.

The created `application.xml` file is saved in the `c:\temp\jaxws\works\annotations\server\META-INF\` directory with the UTF-8 format. For notes on creating `application.xml`, see *5.2.2 Notes on editing application.xml* in the *uCosminexus Application Server Application Development Guide*.

7.3.5 Creating an EAR file

Use the `jar` command to create EAR files containing the files created until now.

```
 > cd c:\temp\jaxws\works\annotations\server\
 > jar cvf annotations_dynamic_generate.war .\WEB-INF
 > jar cvf annotations_dynamic_generate.ear .\annotations_dynamic_generate.war .\META-
 INF\application.xml
```

If the `jar` command is terminated successfully, `annotations_dynamic_generate.ear` is created under the `c:\temp\jaxws\works\annotations\server\` directory.

For the `jar` command, see the *JDK documentation*.

7.4 Examples of deployment and startup (Starting from SEI or customization)

This section describes examples of the deployment and startup, starting from SEI.

7.4.1 Deploying EAR files

Use the `cjimportapp` command to deploy the created EAR file on the J2EE server.

The following is an example of deployment:

```
> cd c:\temp\jaxws\works\annotations\server\
> "%COSMINEXUS_HOME%\CC\admin\bin\cjimportapp" jaxwsserver -nameserver
corbaname::testserver:900 -f annotations_dynamic_generate.ear
```

For the `cjimportapp` command, see *cjimportapp (Importing J2EE applications)* in the *uCosminexus Application Server Command Reference Guide*.

For the method to deploy (import) J2EE applications by using the management portal, see *12.3.3 Importing J2EE applications* in the *uCosminexus Application Server Management Portal User Guide*.

7.4.2 Starting Web Services

Use the `cjstartapp` command to start the Web Service.

The following is an example of starting the Web Service:

```
> cd c:\temp\jaxws\works\annotations\server\
> "%COSMINEXUS_HOME%\CC\admin\bin\cjstartapp" jaxwsserver -nameserver
corbaname::testserver:900 -name Sample_application_annotations_dynamic_generate
```

For the `cjstartapp` command, see *cjstartapp (Starting J2EE applications)* in the *uCosminexus Application Server Command Reference Guide*..

For the method to start J2EE applications by using the management portal, see *12.3.1 Starting J2EE applications* in the *uCosminexus Application Server Management Portal User Guide*.

7.5 Examples of developing Web Service clients (Starting from SEI or customization)

This section describes the examples of the customization of Web Service clients, starting from SEI.

7.5.1 Creating a service class

If you execute the `cjwsimport` command, the Java sources, such as service class, required for Web Service client development will be created. For the `cjwsimport` command, see *14.1 cjwsimport command*.

The following is an example of the execution of the `cjwsimport` command:

```
> cd c:\temp\jaxws\works\annotations\client\
> mkdir src\
> mkdir classes\
> "%COSMINEXUS_HOME%\jaxws\bin\cjwsimport.bat" -s src -d classes http://webhost:8085/
annotations_dynamic_generate /TestJaxWsService?wsdl
```

If the `cjwsimport` command is terminated successfully, the Java source is created in the `c:\temp\jaxws\works\annotations\client\src\com\example\sample\` directory. Note that `com\example\sample\`(*directory-path-corresponding-to-the-package*) changes as per the coding of the namespace URI. For the mapping between namespace URIs and packages, see *15.1.1 Mapping a namespace to a package name*.

The following table lists and describes the products:

Table 7-3: Products during service class generation (Starting from SEI or customization)

File name	Description
`JaxWsTest1.java`	This is a `JavaBean` class corresponding to type referenced by the wrapper element of the request message for 'Operation' in the WSDL definition.
`JaxWsTest1Response.java`	This is a `JavaBean` class corresponding to type referenced by the wrapper element of the response message for 'Operation' in the WSDL definition.
`ObjectFactory.java`	This is the `ObjectFactory` class of the JAXB 2.2 specifications.
`package-info.java`	This is the `package-info.java` file.
`TestJaxWs.java`	This is SEI corresponding to 'Service' in the WSDL definition.
`TestJaxWsService.java`	This is the service class.
`AddNumbersFault.java`	This is the `JavaBean` class corresponding to `AddNumbersFault`.
`AddNumbersFault_Exception.java`	This is the wrapper exception class of the fault bean.

The file names `JaxWsTest1`, `TestJaxWs` and `TestJaxWsService` change according to the coding of the local name for operation, local name for port type, and local name for services. For the mapping of a local name for operation, local name for port type, and local name for services see the following sections:

- *15.1.2 Mapping a port type to a SEI name*
- *15.1.3 Mapping an operation to a method name*
- *15.1.4 Mapping a message part to a parameter and return value (For wrapper style)*
- *15.1.5 Mapping a message part to a parameter and return value (For non-wrapper style)*

7.5.2 Creating an implementation class for the Web Service client

Create an implementation class for the Web Service client that uses the Web Service.

The following is an example of creating a Web Service client that invokes the Web Service once:

```
package org.example.sample.client;

import org.example.sample.TestJaxWs;
import org.example.sample.TestJaxWsService;
import org.example.sample.AddNumbersFault_Exception;

public class TestClient {
  public static void main( String[] args ) {
    try {
      TestJaxWsService service = new TestJaxWsService();
      TestJaxWs port = service.getTestJaxWs();

      int returnValue = port.jaxWsTest1( 205, 103 );

      System.out.println( "[RESULT] " + returnValue );
    }
    catch( AddNumbersFault_Exception e ){
      e.printStackTrace();
    }
  }

}
```

The created `TestClient.java` is stored in the `c:\temp\jaxws\works\annotations\client\src\com\example\sample\client\directory` in UTF-8 format.

Note that `org.example.sample`, `TestJaxWs`, `TestJaxWsService`, and `jaxWsTest1` change according to the package name, class name, and method name in the class of the generated Java source. When you want to develop a Web Service with a different configuration, you must review and if necessary revise the coding of the package name, class name, and the method name in the class.

7.5.3 Compiling the implementation class for the Web Service client

Use the `javac` command to compile the created Web Service client.

The following is an example of compilation:

```
> cd c:\temp\jaxws\works\annotations\client\
> javac -encoding UTF-8 -cp "%COSMINEXUS_HOME%\jaxws\lib\cjjaxws.jar;%COSMINEXUS_HOME%
\CC\client\lib\j2ee-javax.jar;%COSMINEXUS_HOME%\jaxp\lib\csmjaxb.jar;.\classes" -d .
\classes src\org\example\sample\client\TestClient.java
```

If the `javac` command is terminated successfully, the `TestClient.class` file is generated in the `c:\temp\jaxws\works\annotations\client\classes\org\example\sample\client\` directory.

For the `javac` command, see the *JDK documentation*.

7.6 Examples for executing Web Services (Starting from SEI or customization)

This section describes the execution example of the customization of Web Service clients starting from SEI.

7.6.1 Creating an option definition file for Java applications

Create an option definition file for Java applications (`usrconf.cfg`) required for executing Web Services.

The following is an example of creating the option definition file for Java applications:

```
add.class.path=Cosminexus-installation-directory\jaxws\lib\cjjaxws.jar
add.class.path=.\classes
ejb.client.log.directory=logs
add.jvm.arg=-Dcosminexus.home=Cosminexus-installation-directory
add.jvm.arg=-Dejbserver.server.prf.PRFID=PRF-ID
```

For the *Cosminexus-installation-directory* part, use the absolute path to specify the path where *Cosminexus* is installed.

The created option definition file for Java applications is stored in the `c:\temp\jaxws\works\annotations\client\directory`. For the option definition file for Java applications, see *14.2 usrconf.cfg (Option definition file for Java applications)* in the *uCosminexus Application Server Definition Reference Guide*.

7.6.2 Creating a user property file for Java applications

Create a user property file for Java applications required for executing a Web Service.

Since the settings are not specially changed, create an empty file named `usrconf.properties` under `c:\temp\jaxws\works\annotations\client\directory`. For the user property file for Java applications, see *14.3 usrconf.properties (User property file for Java applications)* in the *uCosminexus Application Server Definition Reference Guide*.

7.6.3 Executing Web Service clients

Use the `cjclstartap` command to execute Web Service client.

The following is an example of executing a Web Service client:

```
> cd c:\temp\jaxws\works\annotations\client\
> "%COSMINEXUS_HOME%\CC\client\bin\cjclstartap" org.example.sample.client.TestClient
```

If the `cjclstartap` command is terminated successfully, the results of the Web Service client execution are displayed. The following is an example of displaying the execution results:

```
KDJE40053-I The cjclstartap command will now start. (directory for the user
definition file = c:\temp\jaxws\works\annotations\client, PID = 2636)
[RESULT] 308
KDJE40054-I The cjclstartap command was stopped. (PID = 2636, exit status = 0)
```

The part in italics changes according to the execution timing and environment.

For the `cjclstartap` command, see *cjclstartap (Starting Java applications)* in the *uCosminexus Application Server Command Reference Guide*.

8 Examples of the Development Starting from SEI (For EJB Web Services)

This chapter describes examples of the development of EJB Web Services starting from SEI.

8.1 Configuration of the development examples (Starting from SEI and EJB Web Services)

This chapter describes an example of developing a Web Service starting from SEI. Although this section describes a stub-based development example, you can perform dispatch-based or API-based development of Web Services.

The following table describes the configuration of Web Services to be developed:

Table 8-1: Web Service configuration (Starting from SEI)

Sr.No.	Item		Value
1	Name of the J2EE server to be deployed		`jaxwsserver`
2	Host name and port number of the Web Server		`webhost:8085`
3	URL of the naming server		`corbaname::testserver:900`
4	Context root		`statelessjava_dynamic_generate`
5	Style		`document/literal/wrapped`
6	Namespace URI		`http://sample.com`
7	Port type	Count	`1`
8		Local name	`AddNumbersImpl`
9	Operation	Count	`1`
10		Local name	`add`
11	Service	Count	`1`
12		Local name	`AddNumbersImplService`
13	Port	Count	`1`
14		Local name	`AddNumbersImplPort`
15	Web Service Implementation Class		`com.sample.AddNumbersImpl`
16	Method that is to be made public in the Web Service Implementation Class	Count	`1`
17		Method name	`add`
18	Exceptions thrown in the methods used in Web Service implementation	Count	`1`
19		Class name	`com.sample.AddNumbersFault`

The following table describes the configuration of the current directory used in the Web Service development.

Table 8-2: Configuration of the current directory (Starting from SEI)

Directory	Description
`c:\temp\jaxws\works\statelessjava`	This is the current directory.
`server\`	This is used in the Web Service development.
`META-INF\`	Corresponds to the META-INF directory of the EAR file.
`application.xml`	Created in *8.3.3 Creating application.xml*
`src\`	Stores the source files (*.java) of the Web Service. Used in *8.3.1 Creating Web Services Implementation Class* and *8.3.2 Compiling Web Services Implementation Class*.

Directory	Description
jar\	Stores the compiled class files (*.class). Used in *8.3.1 Creating Web Services Implementation Class* and *8.3.2 Compiling Web Services Implementation Class*.
WEB-INF\	Corresponds to the WEB-INF directory of the EJB JAR files.
wsdl\	Created in *8.3.4 Creating a WSDL file (optional)*.
statelessjava_dynamic_generate.ear	Created in *8.3.5 Creating EAR files*.
statelessjava_dynamic_generate.war	
client\	This is used in the Web Service client development.
src\	Stores the source files (*.java) of the Web Service client. Used in *8.5.1 Generating a service class* and *8.5.2 Creating an Implementation Class for the Web Service client*.
classes\	Stores the compiled class files (*.class). Used in *8.5.3 Compiling the Implementation Class for the Web Service client*.
usrconf.cfg	Created in *8.6.1 Creating an option definition file for Java applications*.
usrconf.properties	Created in *8.6.2 Creating a user property file for Java applications*.

Change the current directory path according to the development environment.

Note that the description below uses the directory and files names mentioned in the above table. The values in *bold* used in the command execution examples or Java source are the values that are specified or generated in the examples used within this document. Use appropriate values according to the environment to be built.

Furthermore, in the development example described in this chapter, the Web Service and Web Service client are developed in the same environment; however, they can be developed in different environments. To develop the Web Service and Web Service client in separate environments, you must change the current directory path according to the respective environment.

8.2 Procedure for the development examples (Starting from SEI and EJB Web Service)

In the development examples described in this chapter, the procedure for development and execution is as follows:

Developing a Web Service

1. Create Web Services Implementation Class (*8.3.1*)
2. Compiling Web Services Implementation Class (*8.3.2*)
3. Create an `application.xml` file (*8.3.3*)
4. Create a WSDL file (optional) (*8.3.4*)
5. Create an EAR file (*8.3.5*)

Deploying and starting the service

1. Deploy the EAR file (*8.4.1*)
2. Start the Web Service (*8.4.2*)

Developing the Web Service client

1. Execute the `cjwsimport` command and generate a service class (*8.5.1*)
2. Create an Implementation Class for Web Service client (*8.5.2*)
3. Compile the Implementation Class for Web Service client (*8.5.3*)

Executing the Web Service

1. Create an option definition file for Java applications (*8.6.1*)
2. Create a user property file for Java applications (*8.6.2*)
3. Execute the Web Service client (*8.6.3*)

8.3 Example of Web Service development (Starting from SEI and EJB Web Service)

This section describes an example of developing an EJB Web Service starting from SEI.

8.3.1 Creating Web Services Implementation Class(starting from SEI and EJB Web Service)

Create a Web Service Implementation Class that codes the processing of the Web Service.

In this subsection, calculate the contents of the received request message and create a Web Service Implementation Class that returns a response message. Save the created `AddNumbersImpl.java` to the `c:\temp\jaxws\works\statelessjava\server\src\com\sample\` directory in UTF-8 format.

```
package com.sample;

@javax.ejb.Stateless
@javax.jws.Web Service
public class AddNumbersImpl{

    public int add( int number1, int number2 ) throws AddNumbersFault{

        if( ( number1 < 0 ) || ( number2 < 0 ) ){
            throw new AddNumbersFault( "Negative number cannot be added!",
                "Numbers: " + number1 + ", " + number2 );
        }
        return number1 + number2;
    }

}
```

Also, create an exception class `com.sample.AddNumbersFault` thrown in `com.sample.AddNumbersImpl`. Usually, creating the exception class is optional. We will however, create the exception class in this example.

The following is an example of creating an exception class. Save the created `AddNumbersFault.java` in the `c:\temp\jaxws\works\statelessjava\server\src\com\sample\` directory in UTF-8 format.

```
package com.sample;

public class AddNumbersFault extends Exception {

    String detail;

    public AddNumbersFault( String message, String detail ){
        super( message );
        this.detail = detail;
    }

    public String getDetail(){
        return detail;
    }
}
```

8.3.2 Compiling Web Services Implementation Class

Execute the `javac` command to compile Web Services Implementation Class. For details on the `javac` command, see the *JDK documentation*.

The following is an example of the execution of the `javac` command.

```
> cd c:\temp\jaxws\works\statelessjava\server\
> mkdir .\WEB-INF\classes\
> javac -cp "%COSMINEXUS_HOME%\jaxws\lib\cjjaxws.jar;%COSMINEXUS_HOME%\CC\client\lib
\j2ee-javax.jar;%COSMINEXUS_HOME%\jaxp\lib\csmjaxb.jar;%COSMINEXUS_HOME%\jaxp\lib
```

```
\csmjaxp.jar;%COSMINEXUS_HOME%\jaxp\lib\csmstax.jar" -d jar\ -s src src\com\sample
\AddNumbersImpl.java src\com\sample\AddNumbersFault.java
```

On successful execution of the `javac` command, the compiled classes are output to the `c:\temp\jaxws\works\statelessjava\server\WEB-INF\classes\com\sample\` directory. When you execute the `cjwsgen` command for the compiled Web Services Implementation Class, you can check the errors in advance. For details on the `cjwsgen` command, see *14.3 cjwsgen command* and for the error check, see *10.23(1) Using the cjwsgen command for checking errors.*

8.3.3 Creating application.xml

Create the `application.xml` file that is required as an EAR file component.

The following is an example of creating an `application.xml` file. Note that there is no item to be set as Web Service in the `application.xml`.

```
<?xml version="1.0" encoding="UTF-8"?>
<application version="6" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/
javaee/application_6.xsd">

  <description>Sample application &quot;statelessjava_dynamic_generate &quot;</
description>
  <display-name>Sample_application_statelessjava_dynamic_generate </display-name>
  <module>
    <ejb>statelessjava_dynamic_generate.jar</ejb>
  </module>
```

When creating `web.xml` of version 5, specify `5` in the `version` attribute of the `application` element and specify `http://java.sun.com/xml/ns/javaee/application_5.xsd` as the second location information in the `xsd:schemaLocation` attribute.

Save the created `application.xml` in the `c:\temp\jaxws\works\statelessjava\server\META-INF\` directory in UTF-8 format. For notes on creating `application.xml`, see *5.2.2 Notes on editing application.xml* in the *uCosminexus Application Server Application Development Guide*.

8.3.4 Creating a WSDL file (optional)

Creation of a WSDL file during the development starting from SEI is optional, and if a file is created, include that file in an EAR file. This subsection describes an example of executing the WSDL generation functionality of the `cjwsgen` command to create a WSDL file from the compiled Java sources. For the `cjwsgen` command, see *14.3 cjwsgen command*.

The following is an example of executing the `cjwsgen` command:

```
> cd c:\temp\jaxws\works\statelessjava\server\
> mkdir .\jar\META-INF\wsdl\
> mkdir .\temporary
> "%COSMINEXUS_HOME%\jaxws\bin\cjwsgen.bat"
-d .\temporary -cp .\jar com.sample.AddNumbersImpl
> rmdir /S /Q .\temporary
```

On successful termination of the `cjwsgen` command, a resource file is generated in the `c:\temp\jaxws\works\statelessjava\META-INF\wsdl\` directory. The following table lists the generated products:

Table 8-3: Products generated when the `cjwsgen` command is executed

File name	Description
`AddNumbersImplService.wsdl`	This is a WSDL file corresponding to the specified Java source.
`AddNumbersImplService_schema1.xsd`	This is an XML Schema definition referenced from the WSDL file.

Delete the files generated in the `c:\temp\jaxws\works\statelessjava\temporary\` directory as the files are not required.

8.3.5 Creating EAR files

Use the `jar` command to create an EAR file that includes the files created until now.

The following is an example of creating an EAR file:

```
> cd c:\temp\jaxws\works\statelessjava\server\
> jar cvf statelessjava_dynamic_generate.jar -C jar com
> jar cvf statelessjava_dynamic_generate.ear .\statelessjava_dynamic_generate.jar .
\META-INF\application.xml
```

On successful termination of the `jar` command, `statelessjava_dynamic_generate.ear` is created in the `c:\temp\jaxws\works\statelessjava\server\` directory. For the `jar` command, see the *JDK documentation*.

8.4 Examples of deployment and startup (Starting from SEI and EJB Web Service)

This section describes the examples of deployment and startup of services for the development starting from SEI.

8.4.1 Deploying EAR files

Use the `cjimportapp` command to deploy the created EAR file to the J2EE server.

The following is an example of deployment:

```
> cd c:\temp\jaxws\works\statelessjava\server\
> "%COSMINEXUS_HOME%\CC\admin\bin\cjimportapp" jaxwsserver -nameserver
corbaname::testserver:900 -f statelessjava_dynamic_generate.ear
```

For the `cjimportapp` command, see *cjimportapp (Importing J2EE applications)* in the *uCosminexus Application Server Command Reference Guide*.

For the method to deploy (import) J2EE applications by using the management portal, see *12.3.3 Importing J2EE applications* in the *uCosminexus Application Server Management Portal User Guide*.

8.4.2 Starting Web Services

Use the `cjstartapp` command to start the Web Service.

The following is an example of starting the Web Service:

```
> cd c:\temp\jaxws\works\statelessjava\server\
> "%COSMINEXUS_HOME%\CC\admin\bin\cjstartapp" jaxwsserver -nameserver
corbaname::testserver:900 -name Sample_application_statelessjava_dynamic_generate
```

For the `cjstartapp` command, see *cjstartapp (Starting J2EE applications)* in the *uCosminexus Application Server Command Reference Guide*.

For the method to start J2EE applications by using the management portal, see *12.3.1 Starting J2EE applications* in the *uCosminexus Application Server Management Portal User Guide*.

8.5 Examples of Web Service client development (Starting from SEI and EJB Web Service)

This section describes examples of Web Service clients development starting from SEI.

8.5.1 Generating a service class

If you execute the `cjwsimport` command, the Java source, such as a service class that is required for Web Service client development, is generated. For the `cjwsimport` command, see *14.1 cjwsimport command*.

The following is an example of executing the `cjwsimport` command:

```
> cd c:\temp\jaxws\works\statelessjava\client\
> mkdir src\
> mkdir classes\
> "%COSMINEXUS_HOME%\jaxws\bin\cjwsimport.bat" -s src -d classes http://webhost:8085/
statelessjava_dynamic_generate /AddNumbersImplService?wsdl
```

On successful termination of the `cjwsimport` command, the Java source is generated in the `c:\temp\jaxws\works\statelessjava\client\src\com\sample\` directory. Note that the directory path *com\sample* (the directory path corresponding to the package) changes according to the coding of the namespace URI. For mapping between namespace URIs and packages, see *15.1.1 Mapping a namespace to a package name*.

The following table lists the generated products:

Table 8-4: Products generated when the service class is generated (starting from SEI)

File name	Description
`Add.java`	This is a `JavaBean` class corresponding to the type referenced by the `wrapper` element of the request message for "Operation" in the WSDL definition.
`AddResponse.java`	This is a `JavaBean` class corresponding to the type referenced by the `wrapper` element of the response message for "Operation" in the WSDL definition.
`ObjectFactory.java`	This is an `ObjectFactory` class of the JAXB 2.2 specifications.
`package-info.java`	This is the `package-info.java` file.
`AddNumbersImpl.java`	This is the SEI corresponding to the "Service" of the WSDL definition.
`AddNumbersImplService.java`	This is a service class.
`AddNumbersFault.java`	This is a `JavaBean` class corresponding to `AddNumbersFault`.
`AddNumbersFault_Exception.java`	This is a wrapper exception class of the fault bean.

The file names `Add`, `AddNumbersImpl`, and `AddNumbersImplService` change according to the coding of the local name for operation, local name for port type, and local name for services. For the mapping of a local name for operation, local name for port type, and local name for services see the following sections:

- *15.1.2 Mapping a port type to a SEI name*
- *15.1.3 Mapping an operation to a method name*
- *15.1.4 Mapping a message part to a parameter and return value (For wrapper style)*
- *15.1.5 Mapping a message part to a parameter and return value (For non-wrapper style)*

8.5.2 Creating an Implementation Class for the Web Services client

Create an Implementation Class for the Web Service client that uses Web Services.

The following is an example of creating Web Service clients that invoke Web Services once:

```
package com.sample.client;

import com.sample.AddNumbersImpl;
import com.sample.AddNumbersImplService;
import com.sample.AddNumbersFault_Exception;

public class TestClient {
    public static void main( String[] args ) {
        try {
            AddNumbersImplService service = new AddNumbersImplService();
            AddNumbersImpl port = service.getAddNumbersImplPort();

            int returnValue = port.add( 205, 103 );

            System.out.println( "[RESULT] " + returnValue );
        }
        catch( AddNumbersFault_Exception e ){
            e.printStackTrace();
        }
    }

}
```

Save the created `TestClient.java` in the `c:\temp\jaxws\works\statelessjava\client\src\com\sample\client\` directory in UTF-8 format.

Note that `com.sample`, `AddNumbersImpl`, `AddNumbersImplService`, `AddNumbersImplPort`, and `add` change according to the package name, class name, and method name in the class of the generated Java sources. When you want to develop a Web Service with a different configuration, you must review, and if necessary revise, the coding of the package name, class name, and the method name in the class.

8.5.3 Compiling the Implementation Class for the Web Services client

Use the `javac` command to compile the created Web Service client.

The following is an example of compilation:

```
> cd c:\temp\jaxws\works\statelessjava\client\
> javac -encoding UTF-8 -cp "%COSMINEXUS_HOME%\jaxws\lib\cjjaxws.jar;%COSMINEXUS_HOME%
\CC\client\lib\j2ee-javax.jar;%COSMINEXUS_HOME%\jaxp\lib\csmjaxb.jar;.\classes" -d .
\classes src\com\sample\client\TestClient.java
```

On successful termination of the `javac` command, the `TestClient.class` is generated in the `c:\temp\jaxws\works\statelessjava\client\classes\com\sample\client\` directory. For the `javac` command, see the *JDK documentation*.

8.6 Examples of executing Web Services (Starting from SEI and EJB Web Service)

This section describes examples of Web Service client execution starting from SEI.

8.6.1 Creating an option definition file for Java applications

Create an option definition file for Java applications (`usrconf.cfg`) required for executing the Web Service.

The following is an example of creating the option definition file for Java applications:

```
add.class.path=Cosminexus-Installation-directory \jaxws\lib\cjjaxws.jar
add.class.path=.\classes
ejb.client.log.directory=logs
add.jvm.arg=-Dcosminexus.home= Cosminexus-installation-directory
add.jvm.arg=-Dejbserver.server.prf.PRFID=<PRF ID>
```

For the *Cosminexus-installation-directory* part, use the absolute path to specify the path where *Cosminexus* is installed.

The created option definition file for Java applications is stored in the `c:\temp\jaxws\works\statelessjava\client\` directory. For the option definition file for Java applications, see *14.2 usrconf.cfg(Option definition file for Java applications)* in the *uCosminexus Application Server Definition Reference Guide*.

8.6.2 Creating a user property file for Java applications

Create a user property file for Java applications required for executing the Web Service.

As the settings are not particularly changed in this example, create a blank file named `usrconf.properties` in the `c:\temp\jaxws\works\statelessjava\client\` directory. For the user property file for Java applications, see *15.3 usrconf.properties (User property file for Java applications)* in the *uCosminexus Application Server Definition Reference Guide*.

8.6.3 Executing the Web Services clients

Use the `cjclstartap` command to execute the Web Service client.

The following is an example of executing the Web Service client:

```
> cd c:\temp\jaxws\works\statelessjava\client\
> "%COSMINEXUS_HOME%\CC\client\bin\cjclstartap"  com.sample.client.TestClient
```

On successful termination of the `cjclstartap` command, the execution results of the Web Service client are displayed. The following is an example of displaying the execution results:

```
KDJE40053-I The cjclstartap command will now start. (directory for the user
definition file = c:\temp\jaxws\works\statelessjava\client, PID = 2636)
[RESULT] 308
KDJE40054-I The cjclstartap command was stopped. (PID = 2636, exit status = 0)
```

The part in italics changes according to the execution timing and environment.

For the `cjclstartap` command, see *cjclstartap (Starting Java applications)* in the *uCosminexus Application Server Command Reference Guide*.

9 Examples of Development Starting from a Provider (using SAAJ)

This chapter describes the examples for developing Web Services using SAAJ, with a provider as the starting point.

9.1 Configuration examples for development (Starting from a provider and SAAJ)

With the development examples described in this chapter, use SAAJ for developing Web Services that send and receive SOAP Messages, with a dispatch or a provider.

Note that this section describes an example in which the information about an employee number and an employee face photographs is sent from a Web Service client. A Web Service registers the received information, and returns a registration confirmation message. The Java data type of the employee number and the registration confirmation message is `java.lang.String`, and the Java data type of the face photograph is `javax.activation.DataHandler`.

The following table describes the configuration of Web Services to be developed:

Table 9-1: Web Service configuration (Starting from a provider and SAAJ)

Item No.	Item		Value
1	Name of the J2EE server to be deployed		`jaxwsserver`
2	Host name and port number of the Web Server		`webhost:8085`
3	URL of the naming server		`corbaname::testserver:900`
4	Context root		`dispatch_provider`
5	Namespace URI		`http://sample.com`
6	Service	Number	`1`
7		Local name	`UserInfoService`
8	Port	Number	`1`
9		Local name	`UserInfoPort`

The following table describes the configuration of the current directory for developing Web Services:

Table 9-2: Configuration of the current directory (Starting from a provider and SAAJ)

Directory	Description
`c:\temp\jaxws\works\dispatch_provider`	This is the current directory.
`server\`	Used for developing a Web Service.
`META-INF\`	Corresponds to the `META-INF` directory of EAR files.
`application.xml`	Created in *9.3.4 Creating application.xml*.
`src\`	Saves the source file (`*.java`) of the Web Service. Used in *9.3.1 Creating Provider Implementation Classes*.
`WEB-INF\`	Corresponds to the `WEB-INF` directory of WAR files.
`web.xml`	Created in *9.3.3 Creating web.xml*.
`classes\`	Saves the compiled class file (`*.class`). Used in *9.3.2 Generating Java sources*.
`dispatch_provider.ear`	Created in *9.3.5 Creating EAR files*.
`dispatch_provider.war`	
`client\`	Used for developing a Web Service client.

Directory	Description
src\	Saves the source file (*.java) of the Web Service client.Created in *9.5.1 Creating an implementation class for the Web Service client*.
classes\	Saves the compiled class file (*.class). Created in *9.5.2 Compiling the Implementation Class for the Web Service client*.
usrconf.cfg	Created in *9.6.1 Creating option definition files for Java applications*.
usrconf.properties	Created in *9.6.2 Creating user property files for Java applications*.

Change the current directory path according to the environment to be developed.

Note that the directory and file names listed in the above table will be used in the description hereafter. The parts in the *Bold* font of the command execution examples and the Java source indicate the specified values and the generated values used in the examples. Substitute the values according to the environment you want to build.

Also, in the development examples described in this chapter, a Web Service and a Web Service client are developed in the same environment. However, you can also develop Web Services and Web Service clients in different environments. For developing Web Services and Web Service clients in different environments, substitute the current directory path corresponding to the respective environments.

9.2 Procedure for the development examples (Starting from a provider and SAAJ)

In the development examples described in this chapter, the procedure for development and execution is as follows:

Developing a Web Service

1. Creating a Provider Implementation Class (*9.3.1*)
2. Generating Java sources (*9.3.2*)
3. Creating `web.xml` (*9.3.3*)
4. Creating `application.xml` (*9.3.4*)
5. Creating an EAR file (*9.3.5*)

Deploying and starting

1. Deploying the EAR file (*9.4.1*)
2. Starting the Web Service (*9.4.2*)

Developing a Web Service client

1. Creating a Implementation Class for a Web Service client (*9.5.1*)
2. Compiling the Implementation Class for the Web Service client (*9.5.2*)

Executing the Web Service

1. Creating the option definition file for Java applications (*9.6.1*)
2. Creating the user property file for Java applications (*9.6.2*)
3. Executing the Web Service client (*9.6.3*)

9.3 Examples of developing Web Services (Starting from a provider and SAAJ)

This section describes the examples for developing Web Services using SAAJ, with a provider as the starting point.

9.3.1 Creating Provider Implementation Classes

The following is an example of creating a provider implementation class `com.sample.UserInfoImpl`. The created class `com.sample.UserInfoImpl` is saved in the `c:\temp\jaxws\works\dispatch_provider\server\src\com\sample\` directory with the UTF-8 format.

```
package com.sample;

import java.util.Iterator;
import javax.xml.namespace.QName;
import javax.xml.soap.AttachmentPart;
import javax.xml.soap.MessageFactory;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPBodyElement;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPException;
import javax.xml.soap.SOAPMessage;
import javax.xml.ws.Provider;
@javax.xml.ws.WebServiceProvider(serviceName="UserInfoService")
@javax.xml.ws.ServiceMode(value=javax.xml.ws.Service.Mode.MESSAGE)
public class UserInfoImpl implements Provider<SOAPMessage>{

    public SOAPMessage invoke( SOAPMessage request ){

        // Response message
        SOAPMessage response = null;
        // Attachment (face photograph)
        AttachmentPart attachment = null;

        try {
            // Data acquisition from the request message
            // Acquire the employee number
            SOAPBody soapBody = request.getSOAPBody();
            SOAPBodyElement reqRoot =
                (SOAPBodyElement)soapBody.getChildElements().next();
            Iterator number_iterator = reqRoot.getChildElements();
            String number =

((SOAPElement)number_iterator.next()).getFirstChild().getNodeValue();

            // Acquire the face photograph
            Iterator attachment_iterator = request.getAttachments();
            while(attachment_iterator.hasNext()){
                attachment = (AttachmentPart)attachment_iterator.next();
            }

            // If any other process, such as registering the face photograph, is to
be executed
            // for the acquired attachment, implement the process

            // Generating a response message
            response = MessageFactory.newInstance().createMessage();
            SOAPBody resSoapBody = response.getSOAPBody();
            SOAPBodyElement resRoot = resSoapBody.addBodyElement(
                new QName("http://sample.com", "result"));
            SOAPElement soapElement = resRoot.addChildElement(
                new QName("http://sample.com", "value"));
            // Set up a registration confirmation message
            if(null ==  attachment){
                soapElement.addTextNode("Failure(no image).");
            } else {
                soapElement.addTextNode("Success.");
            }

        } catch (SOAPException e) {
            e.printStackTrace();
        }

        return response;
```

```
    }
}
```

For SOAP 1.2, add the `javax.xml.ws.BindingType` annotation to the provider implementation class that you have created. Specify the value `http://www.w3.org/2003/05/soap/bindings/HTTP/` indicating the `SOAP 1.2/HTTP` binding. The following is an example of adding the `javax.xml.ws.BindingType` annotation:

```
package com.sample;

import java.util.Iterator;
import javax.xml.namespace.QName;
import javax.xml.soap.AttachmentPart;
import javax.xml.soap.MessageFactory;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPBodyElement;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPException;
import javax.xml.soap.SOAPMessage;
import javax.xml.ws.Provider;

@javax.xml.ws.BindingType("http://www.w3.org/2003/05/soap/bindings/HTTP/")
@javax.xml.ws.WebServiceProvider
@javax.xml.ws.ServiceMode(value=javax.xml.ws.Service.Mode.MESSAGE)
public class UserInfoImpl implements Provider<SOAPMessage>{

    public SOAPMessage invoke( SOAPMessage request ){

(Hereafter, same as for SOAP1.1)
```

Note that you can also specify the constant value field `javax.xml.ws.soap.SOAPBinding.SOAP12HTTP_BINDING` instead of `http://www.w3.org/2003/05/soap/bindings/HTTP/` in the value of the `javax.xml.ws.BindingType` annotation. For an example to specify constant value fields, see the section *5.3.1 Creating a Web Service Implementation Class*.

9.3.2 Generating Java sources

The following is an example of compiling the created provider implementation class `com.sample.UserInfoImpl`:

- In Windows(x86)

```
> set HNTRLIB2_HOME=HNTRLib2-installation-direcory
> cd c:\temp\jaxws\works\dispatch_provider\server\
> mkdir WEB-INF\classes\
> apt -factory com.cosminexus.istack.ws.AnnotationProcessorFactoryImpl -J-
Dcosminexus.home="%COSMINEXUS_HOME%" -cp "%COSMINEXUS_HOME%\jaxws\lib\cjjaxws.jar;
%COSMINEXUS_HOME%\CC\client\lib\j2ee-javax.jar;%COSMINEXUS_HOME%\jaxp\lib
\csmjaxb.jar;%COSMINEXUS_HOME%\jaxp\lib\csmjaxp.jar;%COSMINEXUS_HOME%\jaxp\lib
\csmstax.jar;%HNTRLIB2_HOME%\classes\hntrlib2j.jar;%HNTRLIB2_HOME%\classes
\hntrlibMj.jar" -d WEB-INF\classes\ -s src src\com\sample\UserInfoImpl.java
```

Specify the execution results of the following command in the *HNTRLib2-installation-directory*.

- In Windows(x64)

```
> set HNTRLIB2_HOME=HNTRLib2-installation-direcory
> cd c:\temp\jaxws\works\dispatch_provider\server\
> mkdir WEB-INF\classes\
> apt -factory com.cosminexus.istack.ws.AnnotationProcessorFactoryImpl
com.cosminexus.istack.ws.AnnotationProcessorFactoryImpl -J-
Dcosminexus.home="%COSMINEXUS_HOME%" -cp "%COSMINEXUS_HOME%\jaxws\lib\cjjaxws.jar;
%COSMINEXUS_HOME%\CC\client\lib\j2ee-javax.jar;%COSMINEXUS_HOME%\jaxp\lib
\csmjaxb.jar;%COSMINEXUS_HOME%\jaxp\lib\csmjaxp.jar;%COSMINEXUS_HOME%\jaxp\lib
\csmstax.jar;%HNTRLIB2_HOME%\classes\hntrlib2j64.jar;%HNTRLIB2_HOME%\classes
\hntrlibMj64.jar" -d WEB-INF\classes\ -s
src src\com\sample\UserInfoImpl.java
```

Specify the execution results of the following command in the *HNTRLib2-installation-directory*.

- In Windows(x86)

```
> "%COSMINEXUS_HOME%\common\bin\gethntr2conf.exe" HNTR2INSTDIR
```

- In Windows(x64)

```
> "%COSMINEXUS_HOME%\common\bin\gethntr2conf64.exe" HNTR2INSTDIR
```

If compilation is successful, `UserInfoImpl.class` is generated in the `c:\temp\jaxws\works\dispatch_provider\server\WEB-INF\classes\com\sample\` directory.

9.3.3 Creating web.xml

Create the `web.xml` file that is required as a component to configure WAR files.

The following is an example of creating `web.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/
javaee/web-app_3.0.xsd">
  <description>Sample web service &quot;dispatch_provider&quot;</description>
  <display-name>Sample_web_service_dispatch_provider</display-name>
  <listener>
    <listener-class>
      com.cosminexus.xml.ws.transport.http.servlet.WSServletContextListener
    </listener-class>
  </listener>
  <servlet>
    <description>Endpoint servlet for Cosminexus JAX-WS</description>
    <display-name>Endpoint_servlet_for_Cosminexus_JAX_WS</display-name>
    <servlet-name>CosminexusJaxwsServlet</servlet-name>
    <servlet-class>
      com.cosminexus.xml.ws.transport.http.servlet.WSServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>CosminexusJaxwsServlet</servlet-name>
    <url-pattern>/UserInfoService</url-pattern>
  </servlet-mapping>
  <session-config>
    <session-timeout>60</session-timeout>
  </session-config>
</web-app>
```

When creating `web.xml` of version 2.5, specify `2.5` in the `version` attribute of the `web-app` element and specify `http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd` as the second location information in the `xsd:schemaLocation` attribute..

Save the created `web.xml` in the `c:\temp\jaxws\works\dispatch_provider\server\WEB-INF` directory in UTF-8 format. For the `web.xml` settings, see the section *3.4 Creating web.xml*.

9.3.4 Creating application.xml

Create the `application.xml` file that is required as a component to configure EAR files. The following is an example of creating `application.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<application version="6" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/
javaee/application_6.xsd">

  <description>Sample application &quot;dispatch_provider&quot;</description>
  <display-name>Sample_application_dispatch_provider</display-name>
  <module>
    <web>
      <web-uri>dispatch_provider.war</web-uri>
```

```
      <context-root>dispatch_provider</context-root>
    </web>
  </module>
</application>
```

When creating `web.xml` of version 5, specify `5` in the `version` attribute of the `application` element and specify `http://java.sun.com/xml/ns/javaee/application_5.xsd` as the second location information in the `xsd:schemaLocation` attribute..

The created `application.xml` is saved in the `c:\temp\jaxws\works\dispatch_provider\server\META-INF` directory with the UTF-8 format. For notes on creating `application.xml`, see *5.2.2 Notes on editing application.xml* in the *uCosminexus Application Server Application Development Guide*.

9.3.5 Creating EAR files

Use the `jar` command to create EAR files containing the files created until now.

The following is an example of creating an EAR file:

```
> cd c:\temp\jaxws\works\dispatch_provider\server\
> jar cvf dispatch_provider.war .\WEB-INF
> jar cvf dispatch_provider.ear .\dispatch_provider.war .\META-INF\application.xml
```

If the `jar` command is terminated successfully, `dispatch_provider.ear` is created in the `c:\temp\jaxws\works\dispatch_provider\server\` directory.

For the `jar` command, see the *JDK documentation*.

9.4 Examples of deployment and startup (Starting from a provider and SAAJ)

This section describes the examples of deployment and startup using a provider as the starting point.

9.4.1 Deploying EAR files

Use the `cjimportapp` command to deploy the created EAR files on the J2EE server.

The following is an example of deployment:

```
> cd c:\temp\jaxws\works\dispatch_provider\server\
> "%COSMINEXUS_HOME%\CC\admin\bin\cjimportapp" jaxwsserver -nameserver
corbaname::testserver:900 -f dispatch_provider.ear
```

For the `cjimportapp` command, see *cjimportapp (Importing J2EE applications)* in the *uCosminexus Application Server Command Reference Guide*.

For the method to deploy (import) J2EE applications by using the management portal, see *12.3.3 Importing J2EE applications* in the *uCosminexus Application Server Management Portal User Guide*.

9.4.2 Starting Web Services

Use the `cjstartapp` command to start Web Services.

The following is an example of starting Web Services:

```
> cd c:\temp\jaxws\works\dispatch_provider\server\
> "%COSMINEXUS_HOME%\CC\admin\bin\cjstartapp" jaxwsserver -nameserver
corbaname::testserver:900 -name Sample_application_dispatch_provider
```

For the `cjstartapp` command, see *cjstartapp (Starting J2EE applications)* in the *uCosminexus Application Server Command Reference Guide*.

For the method to start J2EE applications by using the management portal, see *12.3.1 Starting J2EE applications* in the *uCosminexus Application Server Management Portal User Guide*.

9.5 Examples of Web Services client development (Starting from a provider and SAAJ)

This section describes an example of developing a Web Service client using SAAJ, with a provider as the starting point.

9.5.1 Creating Implementation Class for the Web Services client

Create an implementation class for the Web Service client that uses Web Services.

The following is an example of creating a dispatch-based Web Service client `com.sample.client.TestClient` that invokes a Web Service once.

Note that with a dispatch-based Web Service client, you must explicitly specify the version of the SOAP binding, so the `SOAP 1.1/HTTP` binding is specified in this example. For SOAP 1.2, substitute `SOAPBinding.SOAP11HTTP_BINDING` with `SOAPBinding.SOAP12HTTP_BINDING`.

```
package com.sample.client;

import java.io.File;
import java.util.Iterator;
import javax.activation.DataHandler;
import javax.activation.FileDataSource;
import javax.xml.namespace.QName;
import javax.xml.soap.MessageFactory;
import javax.xml.ws.soap.SOAPBinding;
import javax.xml.soap.AttachmentPart;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPBodyElement;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPException;
import javax.xml.soap.SOAPMessage;
import javax.xml.ws.Dispatch;
import javax.xml.ws.Service;
public class TestClient {
    public static void main( String[] args ) {
        // Generate service
        QName port = new QName( "http://sample.com", "UserInfoPort" );
        Service service = Service.create(
            new QName("http://sample.com", "UserInfoService"));
        String serviceURL = "http://webhost:8085/dispatch_provider/UserInfoService";

        // Add a port to the service
        service.addPort( port, SOAPBinding.SOAP11HTTP_BINDING, serviceURL );

        // Generate Dispatch object
        Dispatch<SOAPMessage> dispatch = service.createDispatch(
            port, SOAPMessage.class, Service.Mode.MESSAGE );

        // Request message
        SOAPMessage request = null;

        try{
            // Generate request message
            request = MessageFactory.newInstance().createMessage();
            SOAPBody reqSoapBody = request.getSOAPBody();
            // Set up employee number
            SOAPBodyElement requestRoot= soapBody.addBodyElement(
                new QName("http://sample.com", "number"));
            SOAPElement soapElement = requestRoot.addChildElement(
                new QName("http://sample.com", "value"));
            soapElement.addTextNode( "1234" );

            // Set up attachment (face photograph)
            String filePath = "C:\\attachment.jpg";
            FileDataSource fds = new FileDataSource(filePath);
            AttachmentPart apPart =
                request.createAttachmentPart(new DataHandler(fds));
            request.addAttachmentPart(apPart);

            // Sending and receiving SOAP Messages
            SOAPMessage response = dispatch.invoke( request );
            // Acquire data from the response message
            SOAPBody resSoapBody = response.getSOAPBody();
```

```
            SOAPBodyElement resRoot =
                (SOAPBodyElement)resSoapBody.getChildElements().next();
            Iterator iterator = resRoot.getChildElements();
            String result =
                ((SOAPElement)iterator.next()).getFirstChild().getNodeValue();

            // Display the registration confirmation message
            System.out.println( "[RESULT] " + result );
        } catch( SOAPException e ) {
            e.printStackTrace();
      }
    }
}
```

The created class `TestClient.java` is saved in the `c:\temp\jaxws\works\dispatch_provider\client\src\com\sample\client\` directory with the UTF-8 format.

9.5.2 Compiling Implementation Class for the Web Services client

Use the `javac` command to compile the created Web Service client.

The following is an example of compilation:

```
> cd c:\temp\jaxws\works\dispatch_provider\client\
> javac -encoding UTF-8 -cp "%COSMINEXUS_HOME%\jaxws\lib\cjjaxws.jar;%COSMINEXUS_HOME%
\CC\client\lib\j2ee-javax.jar;%COSMINEXUS_HOME%\jaxp\lib\csmjaxb.jar;.\classes" -d .
\classes src\com\sample\client\TestClient.java
```

If the `javac` command is terminated successfully, the `TestClient.class` is generated in the `c:\temp\jaxws\works\dispatch_provider\client\classes\com\sample\client\` directory.

For the `javac` command, see the *JDK documentation*.

9.6 Examples for executing Web Services (Starting from a provider and SAAJ)

This section describes examples for executing Web Services using SAAJ, with a provider as the starting point.

9.6.1 Creating option definition files for Java applications

Create option definition files for Java applications (`usrconf.cfg`) required for executing Web Services.

The following is an example of creating option definition files for Java applications:

```
add.class.path=Cosminexus-installation-directory\jaxws\lib\cjjaxws.jar
add.class.path=.\classes
ejb.client.log.directory=logs
add.jvm.arg=-Dcosminexus.home=Cosminexus-installation-directory
add.jvm.arg=-Dejbserver.server.prf.PRFID=PRF-ID
```

For *Cosminexus-installation-directory*, use an absolute path to specify the path where Cosminexus is installed. For *PRF-ID*, specify an identifier of the PRF daemon.

The created option definition file for the Java application is saved in the `c:\temp\jaxws\works\dispatch_provider\client\` directory. For the option definition files for Java applications, see *14.2 usrconf.cfg (Option definition file for Java applications)* in the *uCosminexus Application Server Definition Reference Guide*.

9.6.2 Creating user property files for Java applications

Create user property files for the Java applications required to execute Web Services.

Since the settings are not specially changed, create an empty file *usrconf.properties* in the `c:\temp\jaxws\works\dispatch_provider\client\` directory. For the user property files for Java applications, see *14.3 usrconf.cfg (User property file for Java applications)* in the *uCosminexus Application Server Definition Reference Guide*.

9.6.3 Executing Web Services clients

Use the `cjclstartap` command to execute a Web Service client.

The following is an example of executing a Web Service client:

```
> cd c:\temp\jaxws\works\dispatch_provider\client\
> "%COSMINEXUS_HOME%\CC\client\bin\cjclstartap"  com.sample.client.TestClient
```

If the `cjclstartap` command is terminated successfully, the results of the Web Service client execution will be displayed. The following is an example of displaying the execution results:

```
KDJE40053-I The cjclstartap command will now start. (directory for the user
definition file = c:\temp\jaxws\works\dispatch_provider\client, PID = 2636)
[RESULT] Success.
KDJE40054-I The cjclstartap command was stopped. (PID = 2636, exit status = 0)
```

The part in italics changes according to the execution timing and the environment.

For the `cjclstartap` command, see *cjclstartap (Starting Java applications)* in the *uCosminexus Application Server Command Reference Guide*.

10 Settings and Operations of the JAX-WS Functionality

This chapter describes the operations of the JAX-WS engine that you need to understand for the various settings and operations of the JAX-WS functionality used when developing or operating SOAP Web Services.

10.1 Action definition file

The settings such as the log and timeout settings are coded in the action definition file. There are two types of action definition files:

- **Common definition file**
  This definition file is used for setting up common system operations. There is only one common definition file.
- **Process-wise definition file**
  This definition file is created for setting up the process-specific operations. You create specific settings for each required process. For example, you create a process-wise definition file when you want to change the settings for each J2EE server and when you want to change the settings for the Web Service client.

You can also acquire the message context in the Web Service client for specifying some of the definitions. For the definitions that can be specified as the message context, see *19.2.5(1) Support range of the message context properties*.

This section describes the coding rules for the action definition file and the settings for the definition files.

10.1.1 Coding rules for the action definition file

This subsection describes the coding format and coding rules for the action definition file. This subsection also describes the priority for the action definition.

(1) Coding format

In the action definition file, specify the keys as follows:

key-name=value

(2) Coding rules

Code the action definition file according to the following rules:

- The string up to the linefeed is a *value*.
- A line beginning with a hash mark (#) is a comment.
- You cannot add a comment after the *value*. If you add such a string, the comment is interpreted as a value.
- Use the ISO 8859-1 character encoding in compliance with Java specifications for the characters to be described. Characters such as two-byte characters are interpreted as invalid strings, so convert such characters using the `native2ascii` command. For the `native2ascii` command, see the *JDK documentation*.
- You can also specify a space in the *value*.
- If you enter a space between the *key-name* and = (equal sign) and between = (equal sign) and *value*, the space is removed to interpret the string.
- If you specify a *key-name* other than the *key-name* that can be specified, that *key-name* is not used (warning and error are not displayed).
- If you specify a line without a value, the default value is assumed.
- The key name is case sensitive.

(3) Priority of action definition

The priority order for the action definition is as follows:

1. Defining in the Web Service client (message context)
2. Process-wise definition file
3. Common definition file

The following example of specifying a definition in the Web Service client describes the code for setting up the connection timeout value in the Web Service client:

```
//setConnectTimeout()
int timeout = 60000;
Map<String, Object> ctxt = ((BindingProvider)port).getRequestContext();
ctxt.put("com.cosminexus.jaxws.connect.timeout", timeout);
```

10.1.2 Settings for the common definition file

Use the common definition file to set up the action definition common to the system. This subsection describes the file name, storage directory name, and settings for the common definition file.

(1) File name

The following is the file name for the common definition file:

`cjwconf.properties`

(2) Storage destination directory

The following is the storage destination directory for the common definition file. The storage destination directory is fixed.

Cosminexus-installation-directory`\jaxws\conf`

(3) Settings

The following table lists the key names to be set up and the specified contents:

Table 10-1: Settings for the common definition file

No.	Settings	Key name	Specified contents	Default value	Context
1	Log output level of operation log	`com.cosminexus.jaxws.logger.runtime.message.level`	Specifies the log output level of operation log. Specify `ERROR`, `WARN`, `INFO`, `DEBUG`, or `NONE`. For the output contents corresponding to each specified value, see *39.3.4 Importance level and output conditions of log*.	`INFO`	--
2	Number of operation log files	`com.cosminexus.jaxws.logger.runtime.message.file_num`	Specifies the number of operation log files. Specify a number (1 to 16).	`2`	--
3	Operation file size	`com.cosminexus.jaxws.logger.runtime.message.file_size`	Specifies the operation file size. Specify a number from 4096 to 16777216 (unit: bytes).	`2097152`	--
4	Output of maintenance log	`com.cosminexus.jaxws.logger.runtime.maintenance.level`	Specifies whether to output the maintenance log. If `ALL` is specified The maintenance log is output. If `NONE` is specified The maintenance log is not output.	`ALL`	--
5	Number of maintenance log files	`com.cosminexus.jaxws.logger.runtim`	Specifies the number of maintenance log files. Specify a number from 1 to 16.	`2`	--

No.	Settings	Key name	Specified contents	Default value	Context
5	Number of maintenance log files	e.maintenance.file_num	Specifies the number of maintenance log files. Specify a number from 1 to 16.	2	--
6	Maintenance log capacity	com.cosminexus.jaxws.logger.runtime.maintenance.file_size	Specifies the maintenance log capacity. Specify a number from 4096 to 16777216 (unit: bytes).	16777216	--
7	Log output level of exception log	com.cosminexus.jaxws.logger.runtime.exception.level	Specifies the log output level of exception log. Specify ERROR, WARN, INFO, DEBUG, or NONE. For the output contents corresponding to each specified value, see *39.3.4 Importance level and output conditions of log*.	INFO	--
8	Number of exception log files	com.cosminexus.jaxws.logger.runtime.exception.file_num	Specifies the number of exception log files. Specify a number from 1 to 16.	2	--
9	Exception log capacity	com.cosminexus.jaxws.logger.runtime.exception.file_size	Specifies the exception log capacity. Specify a number from 4096 to 16777216 (unit: bytes).	16777216	--
10	Output level of communication log (In the Web Services client)	com.cosminexus.jaxws.logger.runtime.transport.client_dump	Specifies the output level of the communication log in the Web Services client. If NONE is specified The communication log is not output in the Web Service client. If ALL is specified The messages sent and received with the Web Service client are always output in the communication log. If HEADER is specified The HTTP header of the messages received with the Web Service client is always output in the communication log. If ERROR_HEADER is specified The HTTP header of the message received when SOAPFault is received is output in the communication log. **Notes** When ALL is specified, java.lang.OutOfMemoryError exception might occur depending on the length of the sent and received message. In such a case, adjust the JVM heap size.	ERROR_HEADER	--
11	Output level of communication log (In Web Services)	com.cosminexus.jaxws.logger.runtime.transport.server_dump	Specifies the output level of the communication log in Web Services. If NONE is specified The communication log is not output in the Web Service.	ERROR_HEADER	--

No.	Settings	Key name	Specified contents	Default value	Context
11	Output level of communication log (In Web Services)	com.cosminexus.jaxws.logger.runtime.transport.server_dump	If ALL is specified The messages sent and received in the Web Service are always output in the communication log. If HEADER is specified The HTTP header of the messages received in the Web Service is always output in the communication log. ERROR_HEADER The HTTP header of the message received when SOAPFault is sent is output in the communication log. Note that when a received message is output, the HTTP request information is also output. **Notes** When ALL is specified, java.lang.OutOfMemoryError exception might occur depending on the length of the sent and received message. In such a case, adjust the JavaVM heap size.	ERROR_HEADER	--
12	Number of communication log files	com.cosminexus.jaxws.logger.runtime.transport.file_num	Specifies the number of communication log files. Specify a number from 1 to 16.	2	--
13	Communication log capacity	com.cosminexus.jaxws.logger.runtime.transport.file_size	Specifies the communication log capacity. Specify a number from 4096 to 16777216 (unit: bytes).	16777216	--
14	Character encoding for communication logs	com.cosminexus.jaxws.logger.runtime.transport.encoding	Specifies the character encoding for communication logs. For the character encoding supported by J2SE 6.0, see the *J2SE 6.0 documentation*. If you specify DEFAULT, the character encoding for the communication logs changes to the default platform encoding.	DEFAULT	--
15	Log output level of operation log (cjwsimport)	com.cosminexus.jaxws.logger.cjwsimport.message.level	Specifies the log output level of operation log for the cjwsimport command. Specify ERROR, WARN, INFO, DEBUG, or NONE. For the output contents corresponding to each specified value, see *39.3.4Importance level and output conditions of log*.	INFO	--
16	Number of operation log files (cjwsimport)	com.cosminexus.jaxws.logger.cjwsimport.message.file_num	Specifies the number of operation log files for the cjwsimport command. Specify a number from 1 to 16.	2	--
17	Operation file size (cjwsimport)	com.cosminexus.jaxws.logger.cjwsimport.message.file_size	Specifies the operation file size for the cjwsimport command. Specify a number from 4096 to 16777216 (unit: bytes).	2097152	--
18	Log output level of exception log (cjwsimport)	com.cosminexus.jaxws.logger.cjwsim	Specifies the log output level of exception log for the cjwsimport command. Specify ERROR, WARN, INFO, DEBUG, or	INFO	--

No.	Settings	Key name	Specified contents	Default value	Context
18	Log output level of exception log (cjwsimport)	port.exception.level	NONE. For the output contents corresponding to each specified value, see *39.3.4 Importance level and output conditions of log*.	INFO	--
19	Number of exception log files (cjwsimport)	com.cosminexus.jaxws.logger.cjwsimport.exception.file_num	Specifies the number of exception log files for the cjwsimport command. Specify a number from 1 to 16.	2	--
20	Exception log capacity (cjwsimport)	com.cosminexus.jaxws.logger.cjwsimport.exception.file_size	Specifies the exception log capacity for the cjwsimport command. Specify a number from 4096 to 16777216 (unit: bytes).	16777216	--
21	Output of maintenance log (cjwsimport)	com.cosminexus.jaxws.logger.cjwsimport.maintenance.level	Specifies whether to output the maintenance log for the cjwsimport command. If ALL is specified The maintenance log is output. If NONE is specified The maintenance log is not output.	ALL	--
22	Number of maintenance log files (cjwsimport)	com.cosminexus.jaxws.logger.cjwsimport.maintenance.file_num	Specifies the number of maintenance log files for the cjwsimport command. Specify a number from 1 to 16.	2	--
23	Maintenance log capacity (cjwsimport)	com.cosminexus.jaxws.logger.cjwsimport.maintenance.file_size	Specifies the maintenance log capacity for the cjwsimport command. Specify a number from 4096 to 16777216 (unit: bytes).	16777216	--
24	Log output level of operation log (apt)	com.cosminexus.jaxws.logger.apt.message.level	Specifies the log output level of operation log for the apt command. Specify ERROR, WARN, INFO, DEBUG, or NONE. For the output contents corresponding to each specified value, see *39.3.4Importance level and output conditions of log*.	INFO	--
25	Number of operation log files (apt)	com.cosminexus.jaxws.logger.apt.message.file_num	Specifies the number of operation log files for the apt command. Specify a number from 1 to 16.	2	--
26	Operation file size (apt)	com.cosminexus.jaxws.logger.apt.message.file_size	Specifies the operation file size for the apt command. Specify a number from 4096 to 16777216 (unit: bytes).	2097152	--
27	Log output level of exception log (apt)	com.cosminexus.jaxws.logger.apt.exception.level	Specifies the log output level of exception log for the apt command. Specify ERROR, WARN, INFO, DEBUG, or NONE. For the output contents corresponding to each specified value, see *39.3.4Importance level and output conditions of log*.	INFO	--
28	Number of exception log files (apt)	com.cosminexus.jaxws.logger.apt.exception.file_num	Specifies the number of exception log files for the apt command. Specify a number from 1 to 16.	2	--
29	Exception log capacity (apt)	com.cosminexus.jaxws.logger.apt.exception.file_size	Specifies the exception log capacity for the apt command. Specify a number from 4096 to 16777216 (unit: bytes).	16777216	--

No.	Settings	Key name	Specified contents	Default value	Context
30	Output of maintenance log (apt)	com.cosminexus.jaxws.logger.apt.maintenance.level	Specifies whether to output the maintenance log for the apt command. If ALL is specified The maintenance log is output. If NONE is specified The maintenance log is not output.	ALL	--
31	Number of maintenance log files (apt)	com.cosminexus.jaxws.logger.apt.maintenance.file_num	Specifies the number of maintenance log files for the apt command. Specify a number from 1 to 16.	2	--
32	Maintenance log capacity (apt)	com.cosminexus.jaxws.logger.apt.maintenance.file_size	Specifies the maintenance log capacity for the apt command. Specify a number from 4096 to 16777216 (unit: bytes).	16777216	--
33	Operation option when SOAPAction is absent	com.cosminexus.jaxws.fault_omit_soapaction	Specifies the operation option when SOAPAction is absent. If true is specified If SOAPAction header is absent, returns the SOAP Fault message. If false is specified If SOAPAction header is absent, operates the value of SOAPAction header as a null character.	true	--
34	Connection timeout value of the client socket	com.cosminexus.jaxws.connect.timeout	Specifies the connection timeout value of the client socket. The timeout value specified in this property is applied when the Web Service is invoked and during Meta data (WSDL) acquisition that occurs in the generation of the javax.xml.ws.Service class before the Web Service is invoked. Specify a number from 0 to 2147483647 (unit: milliseconds). If 0 is specified, a timeout does not occur. If you change the settings related to the TCP connection of the OS, the values of OS settings will be given the priority.	60000	Y#
35	Read timeout value of client socket	com.cosminexus.jaxws.request.timeout	Specifies the read timeout value for the client socket. The timeout value specified in this property is applied when the Web Service is invoked and during Meta data (WSDL) acquisition that occurs in the generation of the javax.xml.ws.Service class before the Web Service is invoked. Specify a number from 0 to 2147483647 (unit: milliseconds). If 0 is specified, a timeout does not occur. If you change the settings related to the TCP connection of the OS, the values of OS settings will be given the priority.	300000	Y#
36	User ID for basic authentication	javax.xml.ws.security.auth.username	Specifies the user ID for basic authentication used in HTTP connection.	Nill	Y#

No.	Settings	Key name	Specified contents	Default value	Context
36	User ID for basic authentication	`javax.xml.ws.security.auth.username`	The user ID specified in this property is applied when the Web Service is invoked and during Meta data (WSDL) acquisition that occurs in the generation of the `javax.xml.ws.Service` class before the Web Service is invoked.	Nill	Y#
37	Password for basic authentication	`javax.xml.ws.security.auth.password`	Specifies the password for basic authentication used in HTTP connection. The password specified in this property is applied when the Web Service is invoked and during Meta data (WSDL) acquisition that occurs in the generation of the `javax.xml.ws.Service` class before the Web Service is invoked. If `javax.xml.ws.security.auth.username` is not specified, password is not enabled.	Nil	Y#
38	Presence or absence of HTTP session maintenance	`javax.xml.ws.session.maintain`	Specifies the presence or absence of HTTP session maintenance. An HTTP session is maintained until the same port object is used. If true The session is maintained. If false The session is not maintained.	`false`	Y#
39	Strict validation of handler chain setup file	`com.cosminexus.jaxws.validation.handlerchain.strict`	Specifies whether to validate the handler chain setup file strictly. If `javaee:handler-name` element is not coded as the child element of the `javaee:handler` element in the handler chain setup file, an error occurs.	`false`	--
40	Issuing of WSDL for HTTP request	`com.cosminexus.jaxws.security.publish_wsdl`	Specifies whether to issue WSDL for the Web Service where the HTTP GET request with a query string ?wsdl or ?WSDL is sent to the Web Service URL. If `true` is specified WSDL is issued. If `false` is specified WSDL is not issued and 405 Method Not Allowed is returned.	`true`	--
41	Displaying Web Service information for HTTP request	`com.cosminexus.jaxws.security.display_webservice_info`	Specifies whether to display the Web Service information as the response when an HTTP request by GET arrives for the Web Service. If `true` is specified The Web Service information corresponding to the request URL is displayed. If `false` is specified 405 Method Not Allowed is returned.	`false`	--
42	Propagation of Java exception occurring in the Web Service	`com.cosminexus.jaxws.fault.SOAPFau`	Specifies whether to propagate the Java exception occurring in the Web Service to the Web Service client.	`false`	--

No.	Settings	Key name	Specified contents	Default value	Context
42	Propagation of Java exception occurring in the Web Service	`ltBuilder.captureStackTrace`	If `true` is specified The exception is propagated. If `false` is specified The exception is not propagated.	`false`	--
43	Log output destination directory	`com.cosminexus.jaxws.tool.log.directory`	Specifies the log output destination directory output by the `cjwsimport` command, `apt` command, and `cjwsgen` command.	*Cosminexus-installation-directory*`/jaxws/logs`	--
44	Authenticated user ID of the proxy server	`com.cosminexus.jaxws.http.proxyUser`	Specifies an authenticated user ID of the proxy server. As and when required, specify the user ID when the Web Service client invokes Web Services in an external network using the proxy server. The user ID that you specify in this property is applied during the invocation of Web Services and during the acquisition of Meta data (WSDL) that is created while generating the `javax.xml.ws.Service` class before the invocation of the Web Services.	None	--
45	Password corresponding to the authenticated user ID of the proxy server	`com.cosminexus.jaxws.http.proxyPassword`	Specifies a password corresponding to the authenticated user ID of the proxy server. As and when required, specify this password when the Web Service client invokes Web Services in an external network using the proxy server. The password that you specify in this property is applied during the invocation of Web Services and during the acquisition of Meta data (WSDL) that is created while generating the `javax.xml.ws.Service` class before the invocation of the Web Services.	None	--
46	Authenticated user ID of the proxy server used for establishing a connection through the SSL protocol	`com.cosminexus.jaxws.https.proxyUser`	Specifies an authenticated user ID of the proxy server used when establishing a connection through the SSL protocol. As and when required, specify this user ID during the establishment of a connection through the SSL protocol, when the Web Service client invokes Web Services in an external network using the proxy server. The user ID that you specify in this property is applied during the invocation of Web Services and during the acquisition of Meta data (WSDL) that is created while generating the `javax.xml.ws.Service` class before the invocation of the Web Services.	None	--
47	Password corresponding to the authenticated user ID of the proxy server used for establishing a connection through the SSL protocol	`com.cosminexus.jaxws.https.proxyPassword`	Specifies a password corresponding to the authenticated user ID of the proxy server used when establishing a connection through the SSL protocol. As and when required, specify this password during the establishment of a connection through the SSL protocol, when the Web Service client invokes Web Services in an external network using the proxy server. The user ID that you specify in this property is	None	--

No.	Settings	Key name	Specified contents	Default value	Context
47	Password corresponding to the authenticated user ID of the proxy server used for establishing a connection through the SSL protocol	com.cosminexus.jaxws.https.proxyPassword	applied during the invocation of the Web Services and during the acquisition of Meta data (WSDL) that is created while generating the javax.xml.ws.Service class before the invocation of the Web Services.	None	--
48	Log output level of operation logs (cjwsgen)	com.cosminexus.jaxws.logger.cjwsgen.message.level	Specifies the log output level of operation logs for the cjwsgen command. Specify ERROR, WARN, INFO, DEBUG, or NONE. For the output contents corresponding to each specified value, see *39.3.4 Importance level and output conditions of log*.	INFO	--
49	Number of operation log files (cjwsgen)	com.cosminexus.jaxws.logger.cjwsgen.message.file_num	Specifies the number of operation log files for the cjwsgen command. Specify a number from 1 to 16.	2	--
50	Operation file size (cjwsgen)	com.cosminexus.jaxws.logger.cjwsgen.message.file_size	Specifies the operation file size for the cjwsgen command. Specify a number from 4096 to 16777216 (unit: bytes).	2097152	--
51	Log output level of exception logs (cjwsgen)	com.cosminexus.jaxws.logger.cjwsgen.exception.level	Specifies the log output level of exception logs for the cjwsgen command. Specify ERROR, WARN, INFO, DEBUG, or NONE. For the output contents corresponding to each specified value, see *39.3.4 Importance level and output conditions of log*.	INFO	--
52	Number of exception log files (cjwsgen)	com.cosminexus.jaxws.logger.cjwsgen.exception.file_num	Specifies the number of exception log files for the cjwsgen command. Specify a number from 1 to 16.	2	--
53	Exception log capacity (cjwsgen)	com.cosminexus.jaxws.logger.cjwsgen.exception.file_size	Specifies the exception log capacity for the cjwsgen command. Specify a number from 4096 to 16777216 (unit: bytes).	16777216	--
54	Output of maintenance logs (cjwsgen)	com.cosminexus.jaxws.logger.cjwsgen.maintenance.level	Specifies whether to output maintenance logs for the cjwsgen command. If you specify ALL The maintenance log is output. If you specify NONE The maintenance log is not output.	ALL	--
55	Number of maintenance log files (cjwsgen)	com.cosminexus.jaxws.logger.cjwsgen.maintenance.file_num	Specifies the number of maintenance log files for the cjwsgen command. Specify a number from 1 to 16.	2	--
56	Maintenance log capacity (cjwsgen)	com.cosminexus.jaxws.logger.cjwsgen.maintenance.file_size	Specifies the maintenance log capacity for the cjwsgen command. Specify a number from 4096 to 16777216 (unit: bytes).	16777216	--
57	Specify transport attribute of the soap12:binding element	com.cosminexus.jaxws.publish_wsdl.soap12binding	For SOAP1.2 specify whether to set http://schemas.xmlsoap.org/soap/http in transport attribute of soap12:binding element that is a	DEFAULT	--

No.	Settings	Key name	Specified contents	Default value	Context
57	Specify `transport` attribute of the `soap12:binding` element	`com.cosminexus.jaxws.publish_wsdl.soap12binding`	child element of `wsdl:binding` element in the WSDL issued by using HTTP GET request. The query string of the HTTP GET request must be `?wsdl` or `?WSDL`. DEFAULT Set "`http://www.w3.org/2003/05/ssoa/bindings/HTTP/`" in the `soap12:binding/@transport` attribute value. WSI_BP20_TRANSPORT Set "`http://schemas.xmlsoap.org/ssoa/http`" in the `soap12:binding/@transport` attribute value.	DEFAULT	--

58	Application of the validation of the host name in an SSL connection	`com.cosminexus.xml.ws.client.http.HostnameVerificationProperty`	Specifies whether to omit the following validation when a Web service client connects according to the SSL protocol.
			Validation content: Whether the host name to be included in the end point address matches with the host name given in the certificate.
			`true`
			Omit validation.
			`false`
			Do not omit validation.
			Validation of the host name in the SSL is applied when calling the Web Service, and when acquiring the meta data (WSDL) generated when creating the `javax.xml.ws.Service` class before calling the Web Service.
				false	Y#

Legend:

Y: Indicates that this item can be set as a message context.

--: Indicates that this item cannot be set as a message context.

#

The specification to message context is enabled only when the Web Service is invoked and is not applied during Meta data (WSDL) acquisition that occurs in the generation of the `javax.xml.ws.Service` class before the Web Service is invoked.

To set up the basic authentication information during Meta data acquisition, or the requirement for validation of the host name in an SSL connection, either code in the common definition file or the process-wise definition file or separately download WSDL on the local machine and use it (when using the WSDL that exists on the local machine, connection with the remote machine does not occur while acquiring the meta data). If WSDL imported separately from WSDL exists, download the imported WSDL on the local machine.

Also, when setting `true` or `false` in the property by using the message context, set with a string having a `java.lang.String` type. The operation is not guaranteed if you set `true` or `false` by using a string other than the `java.lang.String` type. For details, see *19.2.5 (2)(k) How to set the com.cosminexus.xml.ws.client.http.HostnameVerificationProperty property*.

(4) Changing settings

Changing settings at the Web Service machine

Stop all the J2EE servers that are not using a process-wise definition file, and then change the settings of the common definition file. For the process-wise definition files, see the section *10.1.3 Settings for the process-wise definition file*.

For changing log-related settings, save a log as and when required, and then change the settings.

Changing settings at the Web Service client machine

Stop all the J2EE servers (when assuming a J2EE application as a Web Service client) or all the Java applications, and then change the settings of the common definition file.

For changing log-related settings, save a log as and when required, and then change the settings.

10.1.3 Settings for the process-wise definition file

Create the process-wise definition file to specify a process-wise specific definition.

The file name and the storage destination directory name of the process-wise definition file are optional. By specifying the storage destination path using the system properties, the process-wise definition is enabled. The following is an example of the specification of the process-wise definition file:

```
com.cosminexus.jaxws.confpath=d:/tmp/example.properties
```

When changing process-wise definitions, stop the target processes (J2EE applications or Java applications), and then change the definitions of the process-wise definition file.

For changing log-related definitions, save a log as and when required, and then change the definitions.

10.2 Operations of the JAX-WS engine

This section describes the operations and support range of the Cosminexus JAX-WS engine.

10.2.1 Operations and support range of the JAX-WS engine

The SOAP Messages exchanged between the Web Service and the Web Service client are marshalled or unmarshalled using the JAX-WS engine operations.

This subsection describes the JAX-WS engine and support range in the Web Service and Web Service client.

(1) Operations and support range of the JAX-WS engine on the Web Service

This point describes operations of the JAX-WS engine on the Web Service and shows the support range of the Web Service client that uses the Web Service.

(a) JAX-WS engine operations on the Web Service

The JAX-WS engine on the Web Service operates with the following procedure:

- Receives the SOAP request message using the `POST HTTP` method from the Web Service client, un-marshals the message, and converts it into a Java object.
- Discovers the target Web Service Implementation Class or the Provider Implementation Class (discovery) and invokes the method corresponding to the operation (dispatch).
- Receives the Java object expressing the SOAP response message and fault message from the target Web Service Implementation Class or the Provider Implementation Class, marshals the object, and returns it to the invocation source as a SOAP response message or fault message.

For discovery and dispatch, see *10.2.2 Discovery and dispatch*.

Also, if the JAX-WS engine at the Web Service machine uses the HTTP GET method to request a WSDL that is the Meta data of the Web Service, and the WSDL does not exist in the WAR file, the WSDL will be automatically generated and returned. For issuing the Meta data, see *10.6 Issuing the Meta data*.

Note that if the JAX-WS engine on the Web Service is invoked using an HTTP method that is neither `POST` nor `GET`, the HTTP status code 405 Method Not Allowed is returned.

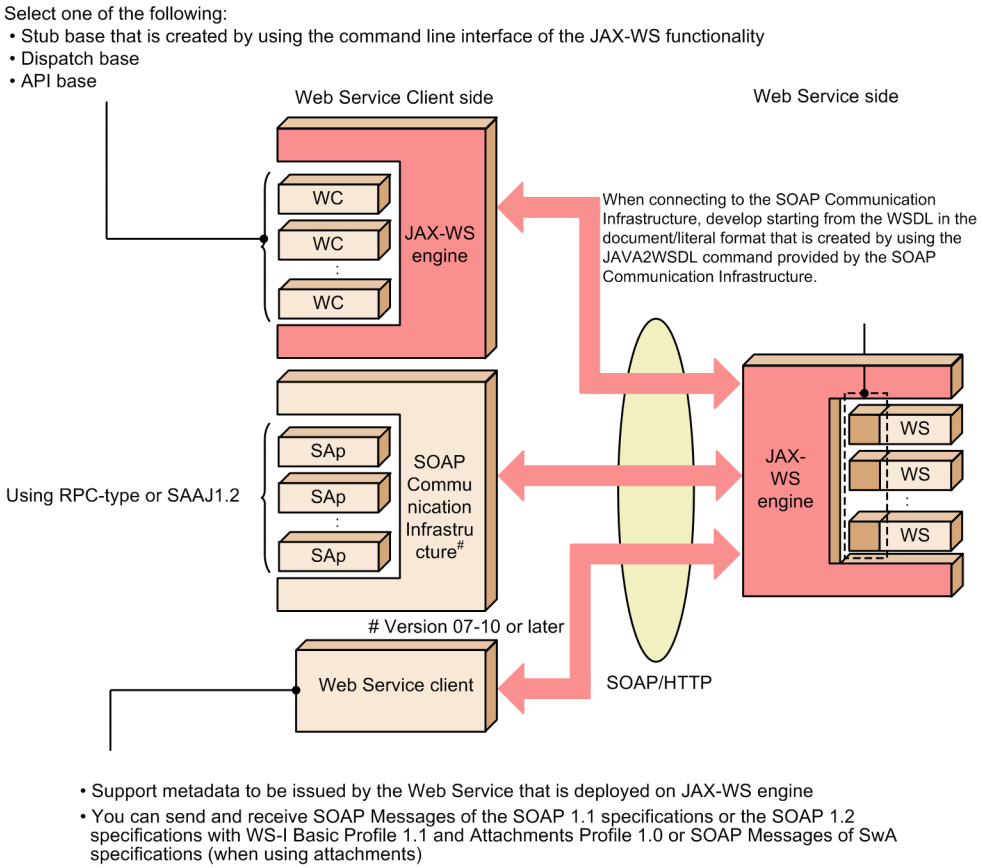
(b) Support range of the JAX-WS engine on the Web Service

This section describes the relationship of the JAX-WS engine on the Web Service and the Web Service client.

For a Web Service Implementation Class

The following figure shows the relationship between a JAX-WS engine on the Web Service machine and a Web Service client, in the case of a Web Service Implementation Class:

Figure 10－1: Relationship of the JAX-WS engine on the Web Service and the Web Service client (For a Web Service Implementation Class)



Legend:
WC: Web Service client
SAp: SOAP application
WS: Web Service Implementation Class

The messages that can be received by the JAX-WS engine on the Web Service and the conditions of the Web Service client at the connection source are as follows:

- Web Service client operated using the Cosminexus JAX-WS functionality

  The Web Service client developed using the commands provided in the Cosminexus JAX-WS functionality and operated using the Cosminexus JAX-WS functionality can be used. If the Cosminexus JAX-WS functionality at the connection destination has an earlier version, you can only use the functionality that is supported by that version.

- RPC-type SOAP application client operated using the SOAP Communication Infrastructure

  The RPC-type SOAP application client developed using the SOAP application development support function and operated using the SOAP Communication Infrastructure can be used.

  The version of SOAP application development support function and SOAP Communication Infrastructure used for development must be 07-10 or a later version. Also, the Web Service in this case must be developed starting from a document/literal-type WSDL generated using the `Java2WSDL` command of the SOAP Communication Infrastructure.

- Other Web Service clients

  You can use a Web Service client that supports the Meta data (WSDL) issued by the Web Service operated using the Cosminexus JAX-WS functionality and can send and receive SOAP Messages of any of the following specifications# with applied WS-I Basic Profile 1.1 and Attachments Profile 1.0:

  - SOAP 1.1 specifications
  - SOAP 1.2 specifications

- SwA specifications (when using attachments in `wsi:swaRef` format)
- MTOM/XOP specifications (when using attachments in the MTOM/XOP specification format).

For the WSDL support range, see *20.1 Support range of the WSDL 1.1 specifications*.

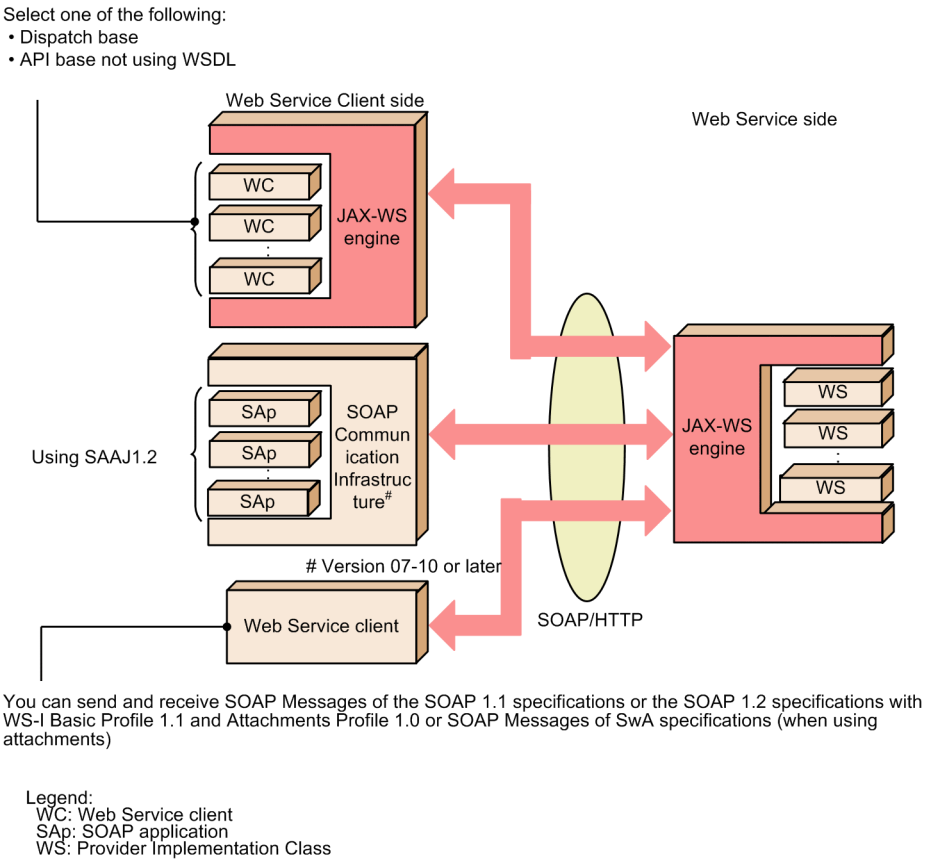
#

From the nature of standard specifications, some ambiguous parts still remain even within the SOAP 1.1 specifications, SOAP 1.2 specifications, SwA specifications or MTOM/XOP specifications having the WS-I Basic Profile 1.1 and the Attachments Profile 1.0. Therefore, perform the operations after studying interconnectivity properly.

For a Provider Implementation Class

The following figure shows the relationship between a JAX-WS engine at the Web Service machine and a Web Service client, in the case of a Provider Implementation Class:

Figure 10–2: Relationship between a JAX-WS engine at the Web service machine and the Web Service client (For a Provider Implementation Class)



The following are the messages that the JAX-WS engine can receive at the Web Service machine and the conditions for the Web Service client at the connection source machine:

- A Web Service client operating with the Cosminexus JAX-WS functionality

  You can use a Web Service client that is developed using the APIs provided by the Cosminexus JAX-WS functionality and operated using the Cosminexus JAX-WS functionality. If the Cosminexus JAX-WS functionality at the connection destination machine has an earlier version, you can only use the functionality that is supported by that version.
- A dispatch-based Web Service client operating with the Cosminexus JAX-WS functionality
- A SOAP application client using the SAAJ 1.2 specifications

  You can use a SOAP application client that is developed using the SOAP application development support functionality and operated with the SOAP Communication Infrastructure using the SAAJ 1.2 specifications.

The SOAP application development support function used during the development and the SOAP Communication Infrastructure must have 07-10 or later versions. Also, the SOAP application must send and receive the SOAP Messages supported by both the SOAP application development support functionality and the Cosminexus JAX-WS functionality.

- Other Web Service clients

  You can use a Web Service client that can send and receive the SOAP Messages of any of the following specifications# with applied WS-I Basic Profile 1.1 and Attachments Profile 1.0:

  - SOAP 1.1 specifications
  - SOAP 1.2 specifications
  - SwA specifications (when using attachments in `wsi:swaRef` format)

  #

  Due to the nature of standard specifications, some ambiguous parts still remain even with SOAP 1.1 specifications, SOAP 1.2 specifications, or SwA specifications having WS-I Basic Profile 1.1 and Attachments Profile 1.0. Therefore, perform the operations after studying interconnectivity properly. Specially note that the variance of SOAP Messages that you can send and receive becomes large for Provider Implementation Classes.

(2) Operations and support range of the JAX-WS engine on the Web Service client

This point describes the operations of the JAX-WS engine on the Web Service client and shows the support range of the Web Service available from the Web Service client.

(a) JAX-WS engine operations on the Web Service client

The JAX-WS engine on the Web Service client operates with the following procedure:

- Receives the Java object expressing the SOAP request message through the JAX-WS API from the Web Service client.
- Marshals the received Java object, and sends as a SOAP request message.
- Receives the SOAP response message and fault message from the invocation destination, un-marshals the message, and returns it to the Web Service client.

The Web Service client accesses the JAX-WS engine through the generated class or JAX-WS API; hence, the JAX-WS engine need not be considered.

Also, since the generated class and JAX-WS API are based on the JAX-WS 2.2 specifications, the implementer of the Web Service client need not consider an interface other than the standard specifications. The invocation of the Web Service (sending of the SOAP request message) and the receipt of the SOAP response message and fault message are performed within the support range for the interface in standard specifications.

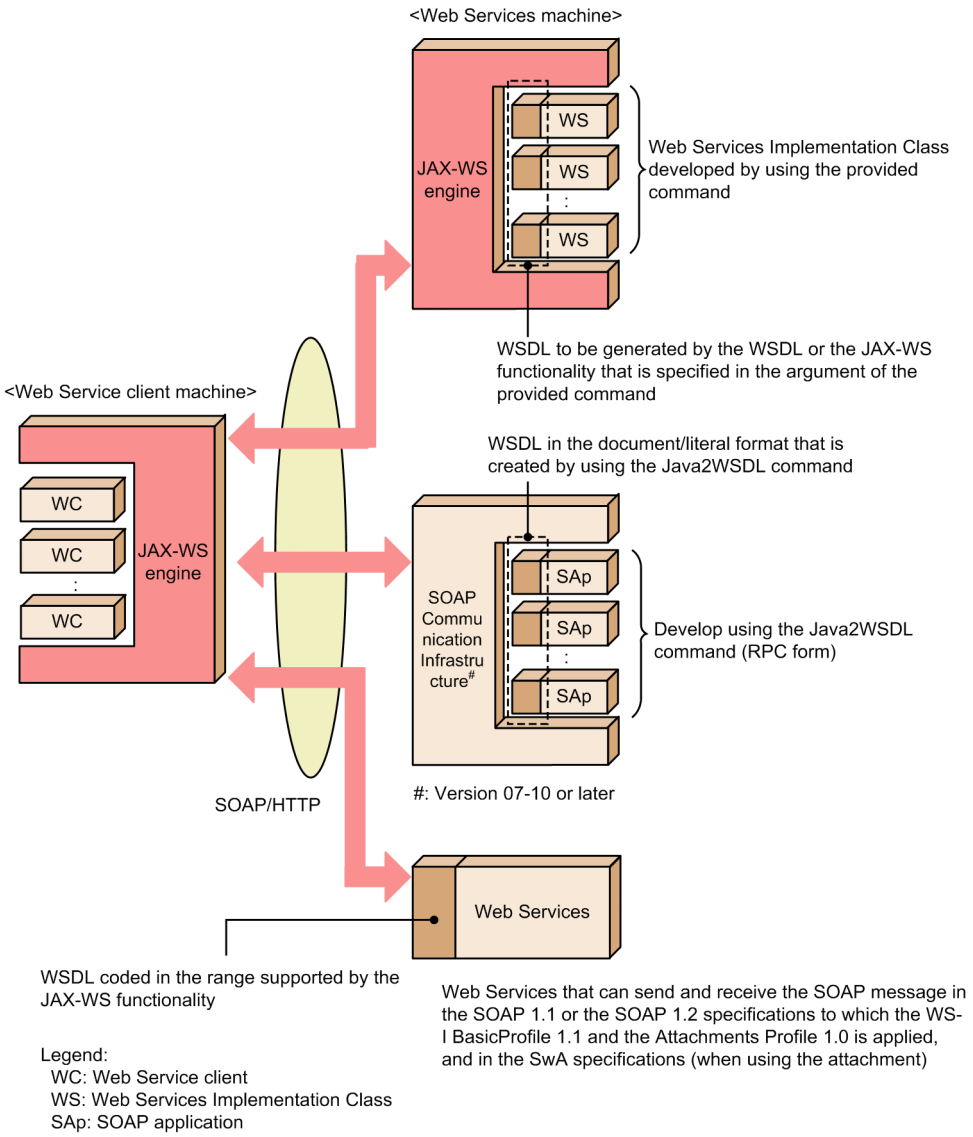
(b) Support range of JAX-WS engine on the Web Service client

This section describes the relationship of the JAX-WS engine on the Web Service client and the Web Service.

For a stub-based Web Service client

The following figure shows the relationship between a JAX-WS engine at the Web Service client machine and a Web Service, in the case of a stub-based Web Service client:

Figure 10－3: Relationship of the JAX-WS engine on the Web Service client and the Web Service (For a stub-based)



The conditions for the Web Service that can be invoked by the JAX-WS engine on the Web Service client are as follows:

- A Web Service Implementation Class developed using the commands provided in the Cosminexus JAX-WS functionality

  The Web Service Implementation Class developed using the commands provided in the Cosminexus JAX-WS functionality and deployed on the JAX-WS engine can be invoked. If the Cosminexus JAX-WS functionality at the connection destination machine has an earlier version, you can only use the functionality that is supported by that version.

- RPC-type SOAP application operated using the SOAP Communication Infrastructure

  The RPC-type SOAP application developed using the SOAP application development support function and deployed on the SOAP Communication Infrastructure can be invoked.

  The version of SOAP application development support function and SOAP Communication Infrastructure used for development must be 07-10 or a later version. Also, the SOAP application must be of the document/literal-type generated using the `Java2WSDL` command of the SOAP Communication Infrastructure.

- Other Web Services

The Web Service that publishes WSDL, coded in the range supported by the Cosminexus JAX-WS functionality, as Meta data and can send and receive SOAP Messages of any of the following specifications# with applied WS-I Basic Profile 1.1 and Attachments Profile 1.0 can be used:

- SOAP 1.1 specifications
- SOAP 1.2 specifications
- SwA specifications (when using attachments in `wsi:swaRef` format)
- MTOM/XOP specifications (when using attachments in the MTOM/XOP specification format).

For the WSDL support range, see *20.1 Support range of the WSDL 1.1 specifications*.

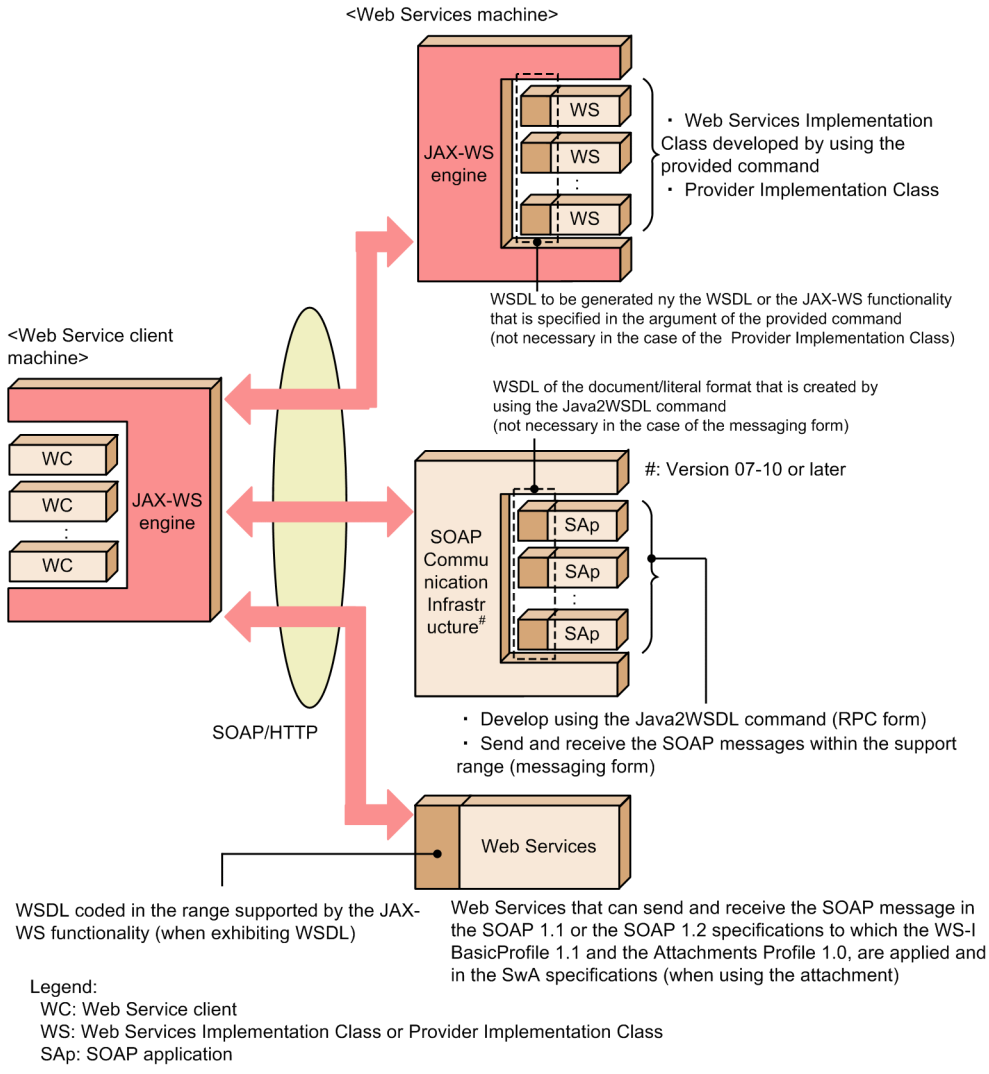
#

From the nature of standard specifications, some ambiguous parts still remain even with SOAP 1.1 specifications, SOAP 1.2 specifications, SwA specifications, or MTOM/XOP specifications having WS-I Basic Profile 1.1 and Attachments Profile 1.0. Therefore, perform the operations after studying interconnectivity properly.

For a dispatch-based Web Service client

The following figure shows the relationship between a JAX-WS engine at the Web Service client machine and a Web Service, for a dispatch-based Web Service client:

Figure 10–4: Relationship between a JAX-WS engine at the Web Service client machine and a Web Service (For a dispatch-based)



The following are the conditions for a Web Service that the JAX-WS engine can invoke at the Web Service client machine:

- A Web Service Implementation Class developed using the commands provided by the Cosminexus JAX-WS functionality

  You can invoke a Web Service Implementation Class developed using the commands provided by the Cosminexus JAX-WS functionality and deployed on the JAX-WS engine. If the Cosminexus JAX-WS functionality has an earlier version, you can only use the functionality that is supported by that version.
- A Provider Implementation Class developed using the commands provided by the Cosminexus JAX-WS functionality

  You can invoke a Provider Implementation Class developed using the commands provided by the Cosminexus JAX-WS functionality and deployed on the JAX-WS engine.
- An RPC-type SOAP application running on the SOAP Communication Infrastructure

  You can invoke an RPC-type SOAP application developed using the SOAP application development support functionality and deployed on the SOAP Communication Infrastructure.

  The SOAP application development support functionality used during the development and the SOAP Communication Infrastructure must have 07-10 or later versions. Also, the SOAP application must have the `document/literal` format that is generated using the `Java2WSDL` command of the SOAP Communication Infrastructure.
- A messaging-type SOAP application running on the SOAP Communication Infrastructure

  You can use a messaging-type SOAP application client developed using the SOAP application development support functionality and deployed on the SOAP Communication Infrastructure of 07-10 or later versions.

  The SOAP application must be able to send and receive the SOAP Messages supported by both the SOAP application development support functionality and the Cosminexus JAX-WS functionality.
- Other Web Services

  You can use a Web Service client that can publish the WSDL coded within the support range of the Cosminexus JAX-WS functionality, as Meta data and can send and receive the SOAP Messages of any of the following specifications# having WS-I Basic Profile 1.1 and Attachments Profile 1.0:

  - SOAP 1.1 specifications
  - SOAP 1.2 specifications
  - SwA specifications (when using attachments in `wsi:swaRef` format)

  For the WSDL support range, see *20.1 Support range of the WSDL 1.1 specifications*.

  #

  Because of the nature of standard specifications, some ambiguous parts still remain even with the SOAP 1.1 specifications, SOAP 1.2 specifications, or SwA specifications having WS-I Basic Profile 1.1 and Attachments Profile 1.0. Therefore, perform operations after properly studying the interconnectivity. Specially note that the variance of the SOAP Messages that you can send and receive becomes large when the WSDL is not published as the Meta data.

(3) Notes for using arrays and java.util.List

Note that when sending a SOAP Message, the status in which element count of *array* and the `java.util.List` object are 0 indicating the empty status and the `null` status cannot be differentiated.

The sections *10.2.1(3)(a) Sending SOAP Messages* and *10.2.1(3)(b) Receiving SOAP Messages* describe the operations of the JAX-WS engine. Also, for the operations when both the Web Service machine and the Web Service client machine use the Cosminexus JAX-WS functionality, see the section *10.2.1(3)(c) When both the Web Service machine and the Web Service client machine use the Cosminexus JAX-WS functionality*.

```
The following methods are used as examples in the description:
@WebMethod
public List<String> test( List<Integer> param )
```

(a) Sending SOAP Messages

This subsection describes a request message when an implementation class of the Web Service client invokes the methods with the following procedure:

- Invoking the method by passing `null` to the first argument
- Invoking the method by passing an object of a concrete class of `java.util.List` with zero number of elements to the first argument

The following is an excerpt of request messages sent by the JAX-WS engine at the Web Service client machine:

```
<test/>
```

Even if you invoke the method with any condition, the elements corresponding to the `param` parameter do not appear in request messages. The same holds true for response messages when the implementation class of the Web Service returns a return value.

(b) Receiving SOAP Messages

When any of the conditions described in the following table is fulfilled, the implementation class of the Web Service client and the Web Service that receives the messages of the section *10.2.1(3)(a) Sending SOAP Messages* accept arrays and objects of concrete classes of `java.util.List` in an empty state without elements:

Table 10-2: Conditions for accepting an empty array or an object of a concrete class of java.util.List

Item no.	Condition
1	When operating the `java.util.List` object with an implementation class of the Web Service client
2	When operating the `java.util.List` object with an implementation class of the Web Service developed with a WSDL as the starting point
3	When operating an array and `java.util.List` that appears directly in the arguments of the method corresponding to the WSDL operation (method annotated with the `WebMethod` annotation), with an implementation class of the Web Service developed with SEI as the starting point

Note that with the Web Service Implementation Class developed with SEI as the starting point, the properties of the arrays and `java.util.List`, included in the JavaBeans class that appear with the arguments of the method corresponding to the WSDL operation, depend on the implementation of the JavaBeans class. If an element corresponding to a property does not exist in the request message, the JavaBeans class accepts the arrays and objects of concrete classes of `java.util.List` with the `null` value.

(c) When both the Web Service machine and the Web Service client machine use the Cosminexus JAX-WS functionality

The operations of the Cosminexus JAX-WS engine are in accordance with the sections *10.2.1(3)(a) Sending SOAP Messages* and *10.2.1(3)(b) Receiving SOAP Messages*. Particularly when both; a Web Service client and a Web Service exist on the Cosminexus server with the implementation class of the Web Service client and the implementation class of the Web Service satisfying the conditions shown in *Table 10-2*, if one of the Web Service client or the Web Service sends `null` as an array and the `java.util.List` object, the other machine receives the array and the concrete class of `java.util.List` with zero number of elements.

10.2.2 Discovery and dispatch

On the JAX-WS engine, discovery of Web Service Implementation Classes and dispatch of SOAP Messages is performed in order to send and receive SOAP Messages.

This subsection describes the discovery of a Web Service Implementation Class, dispatch of SOAP Messages, and the mapping between fault and exception classes.

This subsection also describes the transparency of interfaces.

(1) Discovery

The requested Web Service Implementation Class or the requested Provider Implementation Class is discovered from the Web Service client using the JAX-WS engine on the Web Service. This is called *discovery*.

In discovery, the processing for mapping the appropriate Web Service Implementation Class is performed from the URL requested in the SOAP request message. This point describes the mapping when the following URL is requested:

`http://example.org/fromwsdl/TestJaxWsService`

If the context root is assumed to be `fromwsdl`, `/TestJaxWsService` after the context root (underlined part) indicates the path information. The Web Service Implementation Class or the requested Provider Implementation Class is mapped on the basis of this path information.

The following figure shows an example of mapping between the path information and the Web Service Implementation Class:

Figure 10-5: Discovery of the Web Service Implementation Class (POJO Web Service)

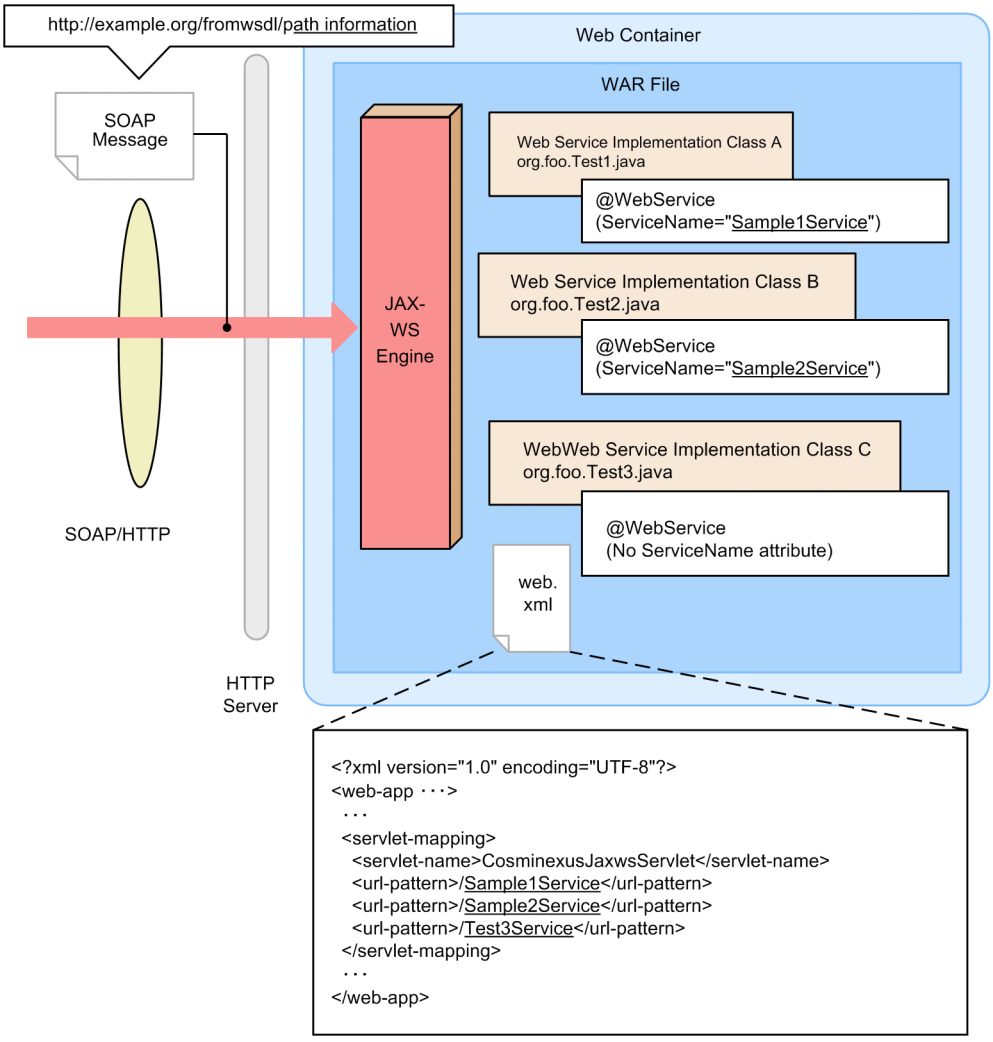
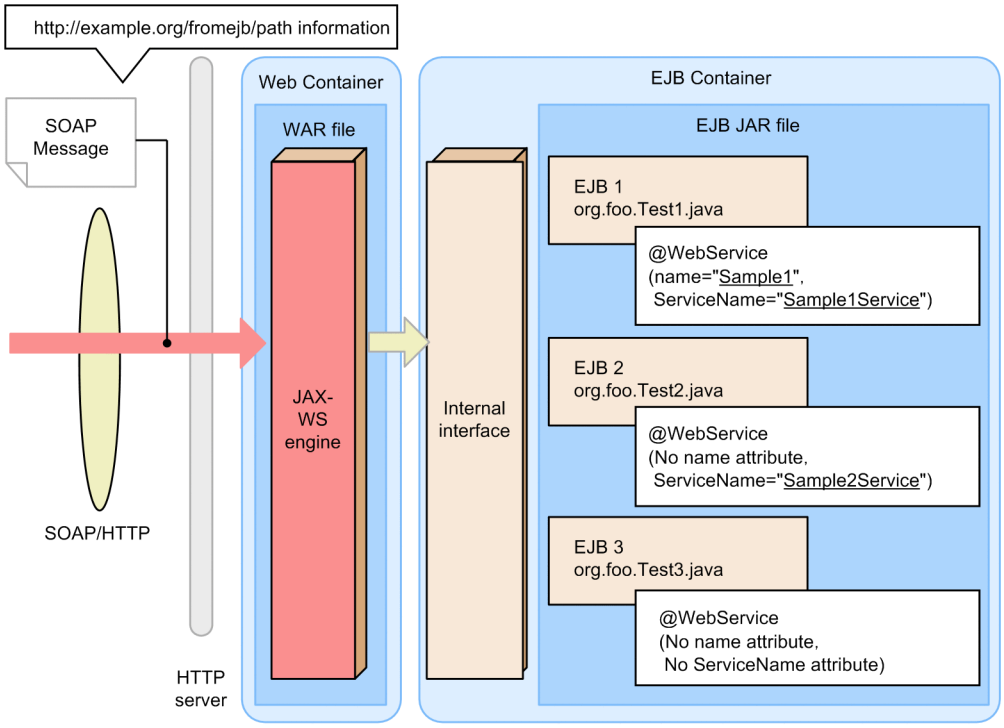


Figure 10-6: Discovery of the Web Service Implementation Class (EJB Web Service)



Among the deployed Web Service Implementation Classes or Provider Implementation Classes, the JAX-WS engine on the Web Service invokes the class corresponding to the path information. In POJO Web Services, the `serviceName` attribute of the `javax.jws.WebService` or the `javax.xml.ws.WebServiceProvider` annotation invokes the string matching the one with / (forward slash) at the beginning removed from the path information.

In EJB Web Services, the `serviceName` attribute of the `javax.jws.WebService` annotation invokes the string matching the one between the / (forward slash) at the beginning and the second / (forward slash) in the path information, and the `name` attribute value of the `javax.jws.WebService` annotation invokes the string matching the one after the second / (forward slash) in the path information.

You can omit the `serviceName` attribute of the `javax.jws.WebService` annotation and `javax.xml.ws.WebServiceProvider` annotation. If you omit the `serviceName` attribute, based on the JSR-181 specifications, a string with `Service` added as a suffix to the class name (simple name) of the Web Service Implementation Class or the Provider Implementation Class is considered as the value of the `serviceName` attribute. Also, you can omit the `name` attribute of the `javax.jws.WebService` annotation. If you omit the `name` attribute, based on JSR-181 specifications, the class name (simple name) of the Web Service Implementation Class is considered as the value of the `name` attribute.

The following is the correspondence between the path information in POJO Web Service example and the invoked Web Service Implementation Class:

- If the path information is `Sample1Service`
  The Web Service Implementation Class A (`org.foo.Test1.java`) is invoked.
- If the path information is `Sample2Service`
  The Web Service Implementation Class B (`org.foo.Test2.java`) is invoked.
- If the path information is `Test3Service`
  The Web Service Implementation Class C (`org.foo.Test3.java`) is invoked.

The following is the correspondence between the path information in the EJB Web Service example and the invoked Web Service Implementation Class:

- If the path information is`/Sample1Service/Sample1`

The EJB 1 (`org.foo.Test1.java`) is invoked.

- If the path information is `/Sample2Service/Test2`
  The EJB 2 (`org.foo.Test2.java`) is invoked.
- If the path information is `/Test3Service/Test3`
  The EJB 3 (`org.foo.Test3.java`) is invoked.

You can customize this mapping by coding `cosminexus-jaxws.xml`. The following is the customization of mapping with the coding example of `cosminexus-jaxws.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<endpoints xmlns='http://java.sun.com/xml/ns/jax-ws/ri/runtime'>
 <endpoint
 name="test1"
 implementation="org.foo.Test1"
 url-pattern="/test1"
 />
 <endpoint
 name="test2"
 implementation="org.foo.Test2"
 url-pattern="/test2"
 />
 <endpoint
 name="test3"
 implementation="org.foo.Test3"
 url-pattern="/test3"
 />
</endpoints>
```

The following is the correspondence between the path information in this example and the invoked Web Service Implementation Class:

- If the path information is `test1`
  The Web Service Implementation Class A (`org.foo.Test1.java`) is invoked.
- If the path information is `test2`
  The Web Service Implementation Class B(`org.foo.Test2.java`) is invoked.
- If the path information is `test3`
  The Web Service Implementation Class C(`org.foo.Test3.java`) is invoked.

However, the `url-pattern` attribute of `cosminexus-jaxws.xml` and the `url-pattern` element of `web.xml` must have one to one correspondence. Therefore, if you customize the mapping in this example, you must also change the coding in `web.xml`. The following is a coding example of `web.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app ...>
 ...
 <servlet-mapping>
 <servlet-name>CosminexusJaxwsServlet</servlet-name>
 <url-pattern>/test1</url-pattern>
 <url-pattern>/test2</url-pattern>
 <url-pattern>/test3</url-pattern>
 </servlet-mapping>
 ...
</web-app>
```

For customizing `cosminexus-jaxws.xml`, see *10.3 Customization using cosminexus-jaxws.xml*.

(2) Dispatch of SOAP Messages

With the JAX-WS engine at the Web Service machine, if the discovered target is a Web Service Implementation Class, a method corresponding to the operation is invoked and executed in accordance with the contents of the received SOAP Message. If the discovered target is a Provider Implementation Class, the unmarshalled SOAP Message is converted to the object specified by the Provider Implementation Class, and the `invoke()` method is invoked.

This is called *dispatch* of SOAP Messages.

A SOAP Message must conform to any of the following specifications# having WS-I Basic Profile 1.1 and Attachments Profile 1.0:

- SOAP 1.1 specifications
- SOAP 1.2 specifications
- SwA specifications (when using an attachment in `wsi:swaRef` format)
- MTOM/XOP specifications (when using attachments in the MTOM/XOP specification format).

The following is an example of a SOAP Message for the SOAP 1.1 specifications. Note that this is the example of the SOAP Message without attachment. For the SOAP Messages with attachments, see the section *28.4 SOAP Messages with attachments (wsi:swaRef format)*.

```
POST http://sample.org/fromjava/AddNumbersImplService HTTP/1.1
SOAPAction: ""
Content-Type: text/xml;charset="utf-8"
Accept: text/xml, multipart/related, text/html, image/gif, image/jpeg, *; q=.2, */*;
q=.2

<?xml version="1.0" ?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
 <S:Body>
 <ns2:add xmlns:ns2="http://sample.com/">
 <arg0>256</arg0>
 <arg1>103</arg1>
 </ns2:add>
 </S:Body>
</S:Envelope>
```

At the beginning of the message are the HTTP request line (POST line) followed by the HTTP request header fields (SOAPAction, Content-Type, and Accept lines). After this, keep a blank line and then enter the SOAP Message. Based on the contents of this SOAP Message, appropriate SEI is invoked and processed.

In the header field of the HTTP request, a SOAPAction header is required, and the value of the header must be set as a null character (""). Even if a value is set in the SOAPAction header, the value is ignored by the JAX-WS engine. If a value of a SOAPAction header is not enclosed within quotation marks ["], an error message (KDJW3022-W) is displayed. Also, the operations when the SOAPAction header is not included in the HTTP request differ according to the settings in the action definition file. For the settings in the action definition file, see the section *10.1 Action definition file*.

The following is an example of a SOAP Message for the SOAP 1.2 specifications. Note that this is the example of the SOAP Message without attachment. For the SOAP Messages with attachments, see the section *28.4 SOAP Messages with attachments (wsi:swaRef format)*.

```
POST http://sample.org/fromjava/AddNumbersImplService HTTP/1.1
Content-Type: application/soap+xml;charset="utf-8";action=""
Accept: application/soap+xml, multipart/related, text/html, image/gif, image/jpeg, *;
q=.2, */*; q=.2

<?xml version="1.0" ?>
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope">
  <S:Body>
    <ns2:add xmlns:ns2="http://sample.com/">
      <arg0>256</arg0>
      <arg1>103</arg1>
    </ns2:add>
  </S:Body>
</S:Envelope>
```

In the case of the SOAP 1.2 specifications, the `SOAPAction` header is ignored even if the `SOAPAction` header appears in the header field of the HTTP request. Also, you can omit the `action` parameter.

10.3 Customization using cosminexus-jaxws.xml

The `cosminexus-jaxws.xml` is the DD used for customizing the mapping between the POJO Web Service URL and the Web Service Implementation Class or the Provider Implementation Class. This file is created when you want to customize the processing for discovering the Web Service Implementation Class or the Provider Implementation Class (discovery) or when you want to assign a single Web Service Implementation Class and a Provider Implementation Class for multiple URLs.

If you do not want to customize mapping, `cosminexus-jaxws.xml` is not required. Note that even if you include `cosminexus-jaxws.xml` in the EJB JAR file, the `cosminexus-jaxws.xml` is not applied in the EJB Web Service.

For customization of mapping, see the subsection *10.2.2(1) Discovery*.

Confirm that `cosminexus-jaxws.xml` is coded correctly, when you want to customize mapping. The `cosminexus-jaxws.xml` is read when the Web Service is initialized. Therefore, if an error occurs while `cosminexus-jaxws.xml` is being read, the initialization of the Web Service fails. For Web Service initialization, see the section *10.9(1) Initializing the Web Service*.

This section describes the file name and storage destination of the `cosminexus-jaxws.xml` and the format:

10.3.1 File name and storage destination of cosminexus-jaxws.xml

The `cosminexus-jaxws.xml` is included in the WAR file. To include the `cosminexus-jaxws.xml`, include the file directly under the WEB-INF directory with the name `cosminexus-jaxws.xml`. The file name and storage destination are as follows:

WAR-root`/WEB-INF/cosminexus-jaxws.xml`

10.3.2 Format of cosminexus-jaxws.xml

The following is the format and encoding of `cosminexus-jaxws.xml`. The operations might not function properly if a format and encoding other than the following are coded:

- Format: XML version 1.0
- Encoding: UTF-8

The following table lists the elements that can be specified in `cosminexus-jaxws.xml`:

Table 10-3: List of elements in cosminexus-jaxws.xml

Element name		Specified number	Description
`jaxwsdd:endpoints` element		1	This is the root element.
	`jaxwsdd:endpoint` element	1 or more	Specifies the mapping of the URL to the Web Service Implementation Class or the Provider Implementation Class.

The following is a description on each element and attribute:

(1) jaxwsdd:endpoints element (cosminexus-jaxws.xml)

The `jaxwsdd:endpoints` element is the root element of the `cosminexus-jaxws.xml`. This element does not have attributes.

(2) jaxwsdd:endpoint element (cosminexus-jaxws.xml)

The `jaxwsdd:endpoint` element codes the mapping between the URL and the Web Service Implementation Class or the Provider Implementation Class. One mapping is coded for one `jaxwsdd:endpoint` element. If the Web Service has multiple ports (when one Web Service is provided in multiple URLs) and if multiple Web Services exist

in the same WAR file, you need to code the number of `jaxwsdd:endpoint` elements corresponding to the port and Web Service.

The following table lists and describes the attributes for the `jaxwsdd:endpoint` element:

Table 10-4: List of attributes for the jaxwsdd:endpoint element

No.	Attribute name	Required	Description
1	`name`	Y	Specify the name for distinguishing the `jaxwsdd:endpoint` element.
2	`implementation`	Y	Specify the Web Service Implementation Class or the Provider Implementation Class.
3	`port`	N	Specify the port associated with the Web Service Implementation Class or the Provider Implementation Class. The settings for this attribute have a higher priority than the settings for the `javax.jws.WebService` annotation or the `javax.xml.ws.WebServiceProvider` annotation. Specify this attribute to map one Web Service Implementation Class or a Provider Implementation Class to multiple URLs.
4	`url-pattern`	Y	Specify the URL associated with the Web Service Implementation Class or the Provider Implementation Class. Corresponds to the `url-pattern` element of `web.xml`.

Legend:

Y: Indicates that the specification is required.

N: Indicates that the specification is not required.

(a) name attribute (cosminexus-jaxws.xml)

Specify the name for distinguishing the `jaxwsdd:endpoint` element with a string (string that can be handled in Java) that is not a null character string.

If you specify a null character string, the KDJW20031-E message is displayed during deployment. The value must be unique in the same `cosminexus-jaxws.xml`. If there are multiple `jaxwsdd:endpoint` elements with the same name attribute value, the KDJW40007-W message is displayed.

(b) implementation attribute (cosminexus-jaxws.xml)

Specify the class name of the class that has the `javax.jws.WebService` annotation.

If you specify a null character string, the KDJW20031-E message is displayed during deployment. Also, if you specify a non-existent class, the KDJW20014-E message is displayed during deployment.

When you want to map the same Web Service Implementation Class or the Provider Implementation Class for multiple URLs, specify the coding such that the port attribute is unique. Particularly for the Web Service Implementation Class, if the `port` attribute is not coded or if the `port` attribute value is not unique, an invalid WSDL is issued as the Meta data.

(c) port attribute (cosminexus-jaxws.xml)

Specify `Qname` of the port name (the name attribute value of the `wsdl:port` element) for WSDL issued as Meta data. For issuing the Meta data, see the section *10.6 Issuing the Meta data*.

This attribute can be omitted. If the attribute is omitted or if a null character string is specified, the value of the `portName` attribute of the `javax.jws.WebService` annotation or the `javax.xml.ws.WebServiceProvider` annotation for the class specified in the `implementation` attribute is used. If the `portName` attribute is omitted, a string with `Port` added as the suffix to the simple name of the Web Service Implementation Class or the Provider Implementation Class is used according to JSR-181 specifications.

When you want to map the same Web Service Implementation Class for multiple URLs, if the port attribute is not specified, the Meta data is not issued normally.

(d) url-pattern attribute (cosminexus-jaxws.xml)

Specify the path information associated with the Web Service Implementation Class or the Provider Implementation Class specified in the `implementation` attribute. The discovery is performed on the basis of the value specified in the `url-pattern` attribute. For the discovery, see the subsection *10.2.2(1) Discovery*.

The path information must be a clearly specified value (wild cards such as asterisk cannot be used). Also, the path information must form a one-to-one correspondence with the value of the `url-pattern` element of `web.xml`.

For example, if "`/path1`" is specified in the `url-pattern` attribute, the class specified in the `implementation` attribute is mapped for the request to "`/path1`".

The value must be unique in the same `cosminexus-jaxws.xml`. If there are multiple `jaxwsdd:endpoint` elements with the same `url-pattern` attribute value, the KDJW40009-W message is displayed. In this case, among the `jaxwsdd:endpoint` elements with the same `url-pattern` attribute value, only the mapping coded in the first `jaxwsdd:endpoint` element is enabled.

(3) Example of settings when cosminexus-jaxws.xml is used

The following table describes the coding examples for DD corresponding to the below Web Services:

Table 10-5: Examples of URLs corresponding to the Web Service Implementation Classes

No.	Web Service Implementation Classes	URL
1	`com.sample.AddNumbersImplA`	`/test1`
2	`com.sample.AddNumbersImplB`	`/test2`, `/test3`

The following is an example of `web.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/
javaee/web-app_3.0.xsd">
 <description>Sample web service &amp;quot;fromwsdl&amp;quot;</description>
 <display-name>Sample_web_service_fromwsdl</display-name>
 <listener>
 <listener-class>
 com.cosminexus.xml.ws.transport.http.servlet.WSServletContextListener
 </listener-class>
 </listener>
 <servlet>
 <description>Endpoint servlet for Cosminexus JAX-WS</description>
 <display-name>Endpoint_servlet_for_Cosminexus_JAX_WS</display-name>
 <servlet-name>CosminexusJaxwsServlet</servlet-name>
 <servlet-class>
 com.cosminexus.xml.ws.transport.http.servlet.WSServlet
 </servlet-class>
 <load-on-startup>1</load-on-startup>
 </servlet>
 <servlet-mapping>
 <servlet-name>CosminexusJaxwsServlet</servlet-name>
 <url-pattern>/test1</url-pattern>
 <url-pattern>/test2</url-pattern>
 <url-pattern>/test3</url-pattern>
 </servlet-mapping>
</web-app>
```

When creating version 2.5 of `web.xml`, set the version attribute of the web-app element as `2.5` and the second location information of the `xsd:schemaLocation` attribute as `http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd`.

The following is an example of `cosminexus-jaxws.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<endpoints xmlns='http://java.sun.com/xml/ns/jax-ws/ri/runtime'>
 <endpoint
 name="test1"
 implementation="com.sample.AddNumbersImplA"
 url-pattern="/test1"
```

```
  />
  <endpoint
  name="test2"
  implementation="com.sample.AddNumbersImplB"
  url-pattern="/test2"
  port="{http://sample.com}port1"
  />
  <endpoint
  name="test3"
  implementation="com.sample.AddNumbersImplB"
  url-pattern="/test3"
  port="{http://sample.com}port2"
  />
 </endpoints>
```

When creating version 2.5 of `web.xml`, set the version attribute of the web-app element as `2.5` and the second location information of the `xsd:schemaLocation` attribute as `http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd`.

10.4 Fault and exception processing

The JAX-WS engine binds the faults and exceptions to the Web Service and Web Service client based on the JAX-WS 2.2 specifications. This is applicable to both, POJO and EJB Web Services.

This section describes the fault and exception processing in the JAX-WS engine.

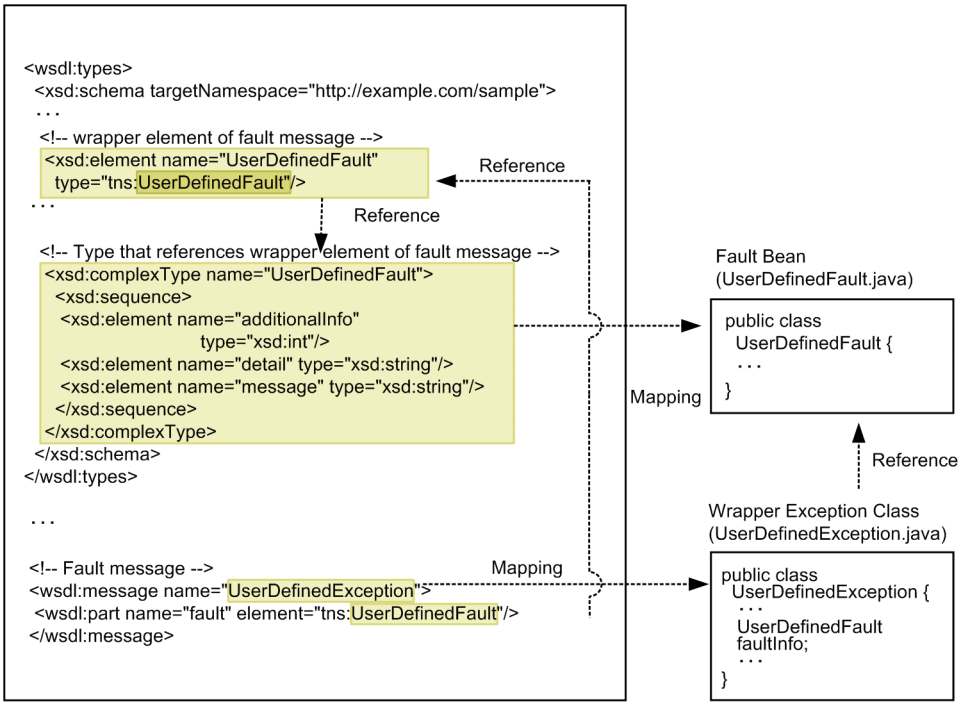
10.4.1 Fault and exception processing on the Web Service

This subsection describes the fault and exception processing in the JAX-WS engine on the Web Service. Note that if the Web Services are implemented with the Provider Implementation Class, this processing will not be executed.

(1) Processing of service-specific exceptions

The WSDL faults and Java exceptions are mapped according to the JAX-WS 2.2 specifications. The following figure shows an example of mapping between the WSDL faults and Java exception classes.

Figure 10–7: Example of mapping between WSDL faults and Java exception classes



In the mapping example, you learn that the `UserDefinedFault` fault is mapped to the fault bean (`com.example.sample.UserDefinedFault`) and the wrapper exception class (`com.example.sample.UserDefinedException`).

For the mapping between the fault and exception classes, see *15.1.7 Mapping the fault to the exception class* and *16.1.7 Mapping the Java wrapper exception class to the fault*.

Using the JAX-WS engine on the Web Service, the wrapper exception class is bound to the SOAP fault as described in the following table:

Table 10-6: Wrapper exception class binding

No.	Child element of the SOAP fault		Contents
	SOAP 1.1 specifications	SOAP 1.2 specifications	
1	faultcode	soapenv12:Code	SOAP 1.1 specifications Fixed to QName soapenv:server. SOAP 1.2 specifications Fixed to QName soapenv 12:Receiver
2	faultstring	soapenv12:Reason	Results in the execution of the getMessage method for the wrapper exception class.
3	faultactor	soapenv12:Role	Does not exist.
4	detail	soapenv12:Detail	Results in the marshalling of the fault bean.

The following is an example of a wrapper exception class in the Web Service Implementation Class:

```
//Generate the fault bean and specify the information you want marshlling
UserDefinedFault fault = new UserDefinedFault();
fault.additionalInfo = 257;
fault.detail = "Failed by some reason.";
fault.message = "Contact your administrator.";

//wrapper exception class is thrown
throw new UserDefinedException(
 "Something happens.", fault );
```

The following is an example of a SOAP fault message of the SOAP 1.1 specifications that will be sent (actually, there is no linefeed and indent):

```
<?xml version="1.0" ?>
<S:Envelope xmlns:S= "http://schemas.xmlsoap.org/soap/envelope/">
 <S:Body>
 <ns2:Fault xmlns:ns2="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns3="http://www.w3.org/2003/05/soap-envelope">
<faultcode>ns2:Server</faultcode>
 <faultstring>Something happens.</faultstring>
 <detail>
 <ns2:UserDefinedFault xmlns:ns2="http://example.com/sample">
 <additionalInfo>257</additionalInfo>
 <detail>Failed by some reason.</detail>
 <message>Contact your administrator.</message>
 </ns2:UserDefinedFault>
 </detail>
 </ns2:Fault>
 </S:Body>
</S:Envelope>
```

#

A SOAP fault message always includes the namespace definition of the SOAP 1.1 and SOAP 1.2 specifications.

The following is an example of a SOAP fault message of the SOAP 1.2 specifications that will be sent (actually, there is no linefeed and indent):

```
<?xml version="1.0" ?>
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope">
  <S:Body>
    <ns3:Fault xmlns:ns2="http://schemas.xmlsoap.org/soap/envelope/"
               xmlns:ns3="http://www.w3.org/2003/05/soap-envelope">
      <ns3:Code>
        <ns3:Value>ns3:Receiver</ns3:Value>
      </ns3:Code>
      <ns3:Reason>
        <ns3:Text xml:lang="ja">Something happens.</ns3:Text>
      </ns3:Reason>
      <ns3:Detail>
```

```
        <env:UserDefinedFault xmlns:env="http://example.com/sample">
          <additionalInfo>257</additionalInfo>
          <detail>Failed by some reason.</detail>
          <message>Contact your administrator.</message>
        </env:UserDefinedFault>
      </ns3:Detail>
    </ns3:Fault>
  </S:Body>
</S:Envelope>
```

#

A SOAP fault message always includes the namespace definition of the SOAP 1.1 and SOAP 1.2 specifications.

(2) Runtime exception binding

If a runtime exception other than `javax.xml.ws.WebServiceException` is thrown in the Web Service Implementation Class, the runtime exception is bound to the SOAP fault by the JAX-WS engine on the Web Service (binding based on the JAX-WS 2.2 specifications).

The following table describes an example of runtime exception binding:

Table 10-7: Runtime exception binding

No.	Child element of the SOAP fault		Contents
	SOAP 1.1 specifications	SOAP 1.2 specifications	
1	`faultcode`	`soapenv12:Code`	SOAP 1.1 specifications Fixed to `QName soapenv:server`. SOAP 1.2 specifications . Fixed to `QName soapenv 12:Receiver`
2	`faultstring`	`soapenv12:Reason`	Results in the execution of the `getMessage` method for the wrapper exception class.
3	`faultactor`	`soapenv12:Role`	Does not exist.
4	`detail`	`soapenv12:Detail`	Results in the marshalling of the fault bean.

The following is an example of a runtime exception:

```
//runtime exception is thrown
throw new IllegalArgumentException( "Something illegal." );
```

The following is an example of a SOAP fault message of the SOAP 1.1 specifications that will be sent (actually, there is no linefeed and indent):

```
<?xml version="1.0" ?>
<S:Envelope xmlns:S= "http://schemas.xmlsoap.org/soap/envelope/">
 <S:Body>
 <ns2:Fault xmlns:ns2="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns3="http://www.w3.org/2003/05/soap-envelope">
<faultcode>ns2:Server</faultcode>
 <faultstring>Something illegal.</faultstring>
 </ns2:Fault>
 </S:Body>
</S:Envelope>
```

#

A SOAP fault message always includes the namespace definition of the SOAP 1.1 and SOAP 1.2 specifications.

The following is an example of a SOAP fault message of the SOAP 1.2 specifications that will be sent (actually, there is no linefeed and indent):

```
<?xml version="1.0" ?>
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope">
  <S:Body>
    <ns3:Fault xmlns:ns2="http://schemas.xmlsoap.org/soap/envelope/"
               xmlns:ns3="http://www.w3.org/2003/05/soap-envelope">
      <ns3:Code>
        <ns3:Value>ns3:Receiver</ns3:Value>
      </ns3:Code>
      <ns3:Reason>
        <ns3:Text xml:lang="ja">Something illegal.</ns3:Text>
      </ns3:Reason>
    </ns3:Fault>
  </S:Body>
</S:Envelope>
```

#

The SOAP fault message always includes the namespace definition of the SOAP 1.1 and SOAP 1.2 specifications.

(3) javax.xml.ws.WebServiceException binding

If `javax.xml.ws.WebServiceException` other than `javax.xml.ws.soap.SOAPFaultException` is thrown in the Web Service Implementation Class or the Provider Implementation Class, `javax.xml.ws.WebServiceException` is bound to the SOAP fault by the JAX-WS engine on the Web Service (binding based on the JAX-WS 2.2 specifications).

The following table describes an example of `javax.xml.ws.WebServiceException` binding:

Table 10-8: javax.xml.ws.WebServiceException binding

No.	Child element of the SOAP fault		Contents
	SOAP 1.1 specifications	SOAP 1.2 specifications	
1	`faultcode`	`soapenv12:Code`	SOAP 1.1 specifications Fixed to `QName soapenv:server`. SOAP 1.2 specifications Fixed to `QName soapenv12:Receiver`.
2	`faultstring`	`soapenv12:Reason`	Results in the execution of the `getMessage` method for the wrapper exception class. For SOAP 1.2, the default locale of JavaVM is set in the `xml:lang` attribute.
3	`faultactor`	`soapenv12:Role`	Does not exist.
4	`detail`	`soapenv12:Detail`	Results in the marshalling of the fault bean.

The following is an example of the `javax.xml.ws.WebServiceException`:

```
//javax.xml.ws.WebServiceException is thrown
throw new javax.xml.ws.WebServiceException( "Web Service Exception." );
```

The following is an example of a SOAP fault message of the SOAP 1.1 specifications that will be sent (actually, there is no linefeed and indent):

```
<?xml version="1.0" ?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
 <S:Body>
 <ns2:Fault xmlns:ns2="http://schemas.xmlsoap.org/soap/envelope/"
            xmlns:ns3="http://www.w3.org/2003/05/soap-envelope">
 <faultcode>ns2:Server</faultcode>
 <faultstring>Web Service Exception.</faultstring>
 </ns2:Fault>
 </S:Body>
</S:Envelope>
```

#

The SOAP fault message always includes the namespace definition of the SOAP 1.1 and SOAP 1.2 specifications.

The following is an example of a SOAP fault message of the SOAP 1.2 specifications that will be sent (actually, there is no linefeed and indent):

```
<?xml version="1.0" ?>
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope">
  <S:Body>
    <ns3:Fault xmlns:ns2="http://schemas.xmlsoap.org/soap/envelope/
               xmlns:ns3="http://www.w3.org/2003/05/soap-envelope">
      <ns3:Code>
        <ns3:Value>ns3:Receiver</ns3:Value>
      </ns3:Code>
      <ns3:Reason>
        <ns3:Text xml:lang="ja">Something illegal.</ns3:Text>
      </ns3:Reason>
    </ns3:Fault>
  </S:Body>
</S:Envelope>
```

#

The SOAP fault message always includes the namespace definition of the SOAP 1.1 and SOAP 1.2 specifications.

(4) javax.xml.ws.soap.SOAPFaultException binding

If `javax.xml.ws.soap.SOAPFaultException` is thrown in the Web Service Implementation Class or the Provider Implementation Class, the `javax.xml.ws.soap.SOAPFaultException` is bound to the SOAP fault by the JAX-WS engine on the Web Service (binding based on the JAX-WS 2.2specifications).

The following table describes an example of `javax.xml.ws.soap.SOAPFaultException` binding:

Table 10-9: javax.xml.ws.soap.SOAPFaultException binding

No.	Child element of the SOAP fault		Contents
	SOAP 1.1 specifications	SOAP 1.2 specifications	
1	`faultcode`	`soapenv12:Code`	SOAP 1.1 specifications Results in the `getFault().getFaultCodeAsQName` method. However, for null, fixed to `QName soapenv:server`. SOAP 1.2 specifications Fixed to `QName soapenv12:Sender`. The child element `soapenv12:Subcode` of `soapenv12:Code` maintains the result.
2	`faultstring`	`soapenv12:Reason`	Results in the `getFaultReasonText` method. However, for null, results in the execution of the `getMessage` method.
3	`faultactor`	`soapenv12:Role`	Results in the `getFault().getFaultRole` method. However, for null, does not exist.
4	`detail`	`soapenv12:Detail`	Results in the marshalling of the results of the execution of the `getFault().getDetail` method. However, for null, does not exist.

The following is an example of `javax.xml.ws.soap.SOAPFaultException` for the SOAP 1.1 specifications:

```
SOAPFault soapFault = ...;
soapFault.setFaultCode( new QName( "http://sample.org", "UserDefined" ) );
soapFault.setFaultActor( "http://example.com/sample" );
soapFault.setFaultString( "SOAPFaultException happens." );
Detail detail = soapFault.addDetail();
SOAPElement soapElement = detail.addChildElement( new QName( "", "detailTest" ) );
soapElement.addTextNode( "TEST." );
```

```
//javax.xml.ws.soap.SOAPFaultException is thrown
throw new SOAPFaultException( soapFault );
```

The following is an example of `javax.xml.ws.soap.SOAPFaultException` for the SOAP 1.2 specifications:

```
SOAPFactory soapFactory = SOAPFactory.newInstance( SOAPConstants.SOAP_1_2_PROTOCOL );
SOAPFault soapFault = soapFactory.createFault();
soapFault.appendFaultSubcode( new QName( "http://sample.org", "UserDefined" ) );
soapFault.setFaultRole( "http://example.com/sample" );
soapFault.addFaultReasonText( "SOAPFaultException happens.", Locale.getDefault() );
Detail detail = soapFault.addDetail();
SOAPElement soapElement = detail.addChildElement( new QName( "", "detailTest" ) );
soapElement.addTextNode( "TEST." );

//javax.xml.ws.soap.SOAPFaultException is thrown
throw new SOAPFaultException( soapFault );
```

The following is an example of a SOAP fault message of the SOAP 1.1 specifications that will be sent (actually, there is no linefeed and indent):

```
<?xml version="1.0" ?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <ns2:Fault xmlns:ns2="http://schemas.xmlsoap.org/soap/envelope/"
               xmlns:ns3="http://www.w3.org/2003/05/soap-envelope">
     <faultcode xmlns:ns0="http://sample.org">ns0:UserDefined</faultcode>
     <faultstring>SOAPFaultException happens.</faultstring>
     <faultactor>http://example.com/sample</faultactor>
     <detail><detailTest>TEST.</detailTest></detail>
    </ns2:Fault>
  </S:Body>
</S:Envelope>
```

#

The SOAP fault message always includes the namespace definition of the SOAP 1.1 and SOAP 1.2 specifications.

The following is an example of a SOAP fault message of the SOAP 1.2 specifications that will be sent (actually, there is no linefeed and indent):

```
<?xml version="1.0" ?>
<S:Envelope xmlns:S= "http://www.w3.org/2003/05/soap-envelope">
  <S:Body>
    <ns3:Fault xmlns:ns2="http://schemas.xmlsoap.org/soap/envelope/"
               xmlns:ns3="http://www.w3.org/2003/05/soap-envelope">
      <ns3:Code>
        <ns3:Value>ns3:Sender</ns3:Value>
        <ns3:Subcode>
          <ns3:Value xmlns:ns0="http://sample.org">ns0:UserDefined</ns3:Value>
        </ns3:Subcode>
      </ns3:Code>
      <ns3:Reason>
        <ns3:Text xml:lang="ja">SOAPFaultException happens.</ns3:Text>
      </ns3:Reason>
      <ns3:Role>http://example.com/sample</ns3:Role>
      <ns3:Detail>
        <env:Detail xmlns:env="http://www.w3.org/2003/05/soap-envelope">
          <detailTest>TEST.</detailTest>
        </env:Detail>
      </ns3:Detail>
    </ns3:Fault>
  </S:Body>
</S:Envelope>
```

#

The SOAP fault message always includes the namespace definition of the SOAP 1.1 and SOAP 1.2 specifications.

10.4.2 Fault processing on the Web Service client

This subsection describes the SOAP faults corresponding to the `wsdl:fault` element and the SOAP faults not corresponding to the `wsdl:fault` element respectively.

- **SOAP faults corresponding to the wsdl:fault element**

  When the JAX-WS engine on the Web Service client receives a SOAP fault message marshalled according to the section *10.4.1(1) Processing of service-specific exceptions*, the SOAP fault message is un-marshalled in the original fault bean and wrapper exception and thrown to the Web Service client. In other words, the service-specific exception thrown by the Web Service is sent transparently to the Web Service client (this operation is according to the JAX-WS 2.2 specifications).

- **SOAP faults not corresponding to the wsdl:fault element**

  When the JAX-WS engine on the Web Service client receives a SOAP fault message not corresponding to the `wsdl:fault` element, the SOAP fault message is un-marshalled in `javax.xml.ws.soap.SOAPFaultException` and thrown to the Web Service client (this operation is according to the JAX-WS 2.2 specifications).

  For `SOAP 1.2`, when the SOAP fault has multiple `soapenv12:Text` elements, only one Reason Text is set in the `SOAPFault` object included in the `SOAPFaultException` object. The value (string) set in the following location and locale is set:

  - Value: value of the last `soapenv12:Text` element
  - Locale: default locale of JavaVM

10.4.3 Propagation of the Java exception

When both the Web Service and Web Service client are deployed on the Cosminexus JAX-WS engine, the Java exception occurring in the Web Service can be propagated to the Web Service client. This is applicable to both, POJO and EJB Web Services. This subsection describes the Java exception propagation method and the operations.

Note

The propagation of the Java exception is not a functionality given in the SOAP 1.1 specifications and JAX-WS 2.2 specifications; therefore, when you connect with the basic Web Service products other than the Cosminexus JAX-WS functionality, unintended operations might occur and communication might fail. The stack trace might also include internal information about Web Service implementation (if system settings information and individual information is handled, that information). Therefore, assuming that this functionality is used in real operations, we recommend that you do not implement the Web Service. Use this functionality as required during development.

(1) Java exception propagation method

To propagate the Java exception that occurred in the Web Service, specify true for the `com.cosminexus.jaxws.fault.SOAPFaultBuilder.captureStackTrace` property in the action definition file.

For the `com.cosminexus.jaxws.fault.SOAPFaultBuilder.captureStackTrace` property, see the section *10.1.2 Settings for the common definition file*.

(2) Operations for propagating the Java exception (in the Web Service)

When the Java exception is propagated in the Web Service, the `{http://jax-ws.dev.java.net/}exception` element specific to the Cosminexus JAX-WS functionality is added as the child element of the `detail` element or `soapenv12:Detail` element, and the information about the occurred Java exception is marshalled.

The following is an example of a runtime exception:

```
//runtime exception is thrown
catch( NullPointerException ){
 throw new IllegalArgumentException( "Something illegal.", e );
}
```

When the exception is thrown in this manner, an example of the SOAP fault message of SOAP 1.1 specifications that will be sent is as follows (actually, there is no linefeed and indent):

```
<?xml version="1.0" ?>
<S:Envelope xmlns:S= "http://schemas.xmlsoap.org/soap/envelope/">
 <S:Body>
 <ns2:Fault xmlns:ns2="http://schemas.xmlsoap.org/soap/envelope/"
            xmlns:ns3="http://www.w3.org/2003/05/soap-envelope">
 <faultcode>ns2:Server</faultcode>
 <faultstring>Something illegal.</faultstring>
 <detail>
 <ns2:exception xmlns:ns2="http://jax-ws.dev.java.net/"
 class="java.lang.IllegalArgumentException"
 note="To disable this feature, set
com.cosminexus.jaxws.fault.SOAPFaultBuilder.captureStackTrace property to false">
 <message>Something illegal</message>
 <ns2:stackTrace>
 <ns2:frame class="com.example.sample.TestJaxWsImpl" file="TestJaxWsImpl.java"
line="32" method="jaxWsTest1"/>
 <ns2:frame class="sun.reflect.NativeMethodAccessorImpl"
file="NativeMethodAccessorImpl.java" line="native" method="invoke0"/>
 <ns2:frame class="sun.reflect.NativeMethodAccessorImpl"
file="NativeMethodAccessorImpl.java" line="39" method="invoke"/>
 ...
 <ns2:frame class="java.lang.Thread" file="Thread.java" line="595" method="run"/>
 </ns2:stackTrace>
 <ns2:cause class="java.lang.NullPointerException"
 note="To disable this feature, set
com.cosminexus.jaxws.fault.SOAPFaultBuilder.captureStackTrace property to false">
 <message>Something null.</message>
 <ns2:stackTrace>
 <ns2:frame class="com.example.sample.TestJaxWsImpl" file="TestJaxWsImpl.java"
line="32" method="jaxWsTest1"/>
 <ns2:frame class="sun.reflect.NativeMethodAccessorImpl"
file="NativeMethodAccessorImpl.java" line="native" method="invoke0"/>
 ...
 <ns2:frame class="sun.reflect.DelegatingMethodAccessorImpl"
file="DelegatingMethodAccessorImpl.java" line="25" method="invoke"/><ns2:frame
class="java.lang.reflect.Method" file="Method.java" line="585" method="invoke"/>
 <ns2:frame class="org.apache.tomcat.util.threads.ThreadPool$ControlRunnable"
file="ThreadPool.java" line="1510" method="run"/>
 <ns2:frame class="java.lang.Thread" file="Thread.java" line="595" method="run"/>
 </ns2:stackTrace>
 </ns2:cause>
 </ns2:exception>
 </detail>
 </ns2:Fault>
 </S:Body>
</S:Envelope>
```

The following is an example of a SOAP fault message of the SOAP 1.2 specifications that will be sent (actually, there is no linefeed and indent):

```
<?xml version="1.0" ?>
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope">
  <S:Body>
    <ns3:Fault xmlns:ns2="http://schemas.xmlsoap.org/soap/envelope/"
               xmlns:ns3="http://www.w3.org/2003/05/soap-envelope">
      <ns3:Code>
        <ns3:Value>ns3:Receiver</ns3:Value>
      </ns3:Code>
      <ns3:Reason>
        <ns3:Text xml:lang="ja">Something illegal.</ns3:Text>
      </ns3:Reason>
      <ns3:Detail>
        <env:Detail xmlns:env="http://www.w3.org/2003/05/soap-envelope">
          <detailTest>TEST.</detailTest>
        </env:Detail>
        <ns2:exception xmlns:ns2="http://jax-ws.dev.java.net/"
            class="javax.xml.ws.soap.SOAPFaultException"
            note="To disable this feature, set
com.cosminexus.jaxws.fault.SOAPFaultBuilder.captureStackTrace property to false">
          ...
          (Same as the SOAP fault of SOAP 1.1 specifications)
          ...
        </ns2:exception>
      </ns3:Detail>
    </ns3:Fault>
  </S:Body>
</S:Envelope>
```

(3) Operations for propagating the Java exception (In the Web Service client)

When the Java exception is propagated in the Web Service client, the exception (exception that occurred in the Web Service) un-marshalled from the information about the `{http://jax-ws.dev.java.net/}exception` element is set up in the cause of the wrapper exception class or `javax.xml.ws.soap.SOAPFaultException` thrown in the Web Service client.

The Web Service client can acquire the exception occurring in the Web Service by executing the `getCause` method.

10.4.4 HTTP status code when binding an exception to a fault

When the JAX-WS engine binds an exception to a fault at the Web Service machine and returns to the Web Service client, "500 Internal Server Error" is set up in the status code of the HTTP response. This is applicable to both, POJO and EJB Web Services.

10.4.5 Notes on customizing an error page

For customizing an *error page* on the J2EE server in which the JAX-WS engine of the Web service machine is operating or in the WAR file that includes Web Services, do not customize the HTTP status code 500. When the JAX-WS engine at the Web service machine returns a fault to the Web Service client, the HTTP status code is set up to "500 Internal Server Error". Therefore, if you customize the HTTP status code 500 instead of the SOAP fault, an invalid SOAP Message will be sent to the Web Service client.

10.5 Interface transparency

The Web Service and Web Service client have a sparse relationship and their only mutual interface is the defined contents of WSDL. In the Web Service, the interface information (Meta data) of the Web Service is published through WSDL and that Meta data is used in the Web Service client to generate and send and receive SOAP Messages.

When both, the Web Service and Web Service client are running using the Cosminexus JAX-WS engine, only the interface information of the WSDL is exchanged.

Since the Java interface is not permeable, when you develop a Web Service starting from SEI, the method signature in the Web Service and the Web Service client might differ.

This section describes the differences in the method signature on the basis of examples of Java methods before generation and Java methods after generation.

(1) When the Java method has an array parameter

The example assumes that a Web Service is to be developed starting from SEI with the following Java method. This Java method has an array (int type) parameter.

```
@WebMethod
public void test1( int[] param1 );
```

The following is a part of the WSDL that is mapped in this case:

```
...
<types>
 <xsd:schema targetNamespace="http://cosminexus.com/jaxws">

 <xs:element name="test1" type="tns:test1"/>

 <xs:element name="test1Response" type="tns:test1Response"/>

 <xs:complexType name="test1">
 <xs:element name="arg0" type="xs:int" nillable="true"
 minOccurs="0" maxOccurs="unbounded" />
 </xs:complexType>

 <xs:complexType name="test1Response">
 <xs:sequence/>
 </xs:complexType>
 </xs:element>
</types>
<message name="test1">
 <part name="parameters" element="tns:test1"/>
</message>

<message name="test1Response">
 <part name="parameters" element="tns:test1Response"/>
</message>

<portType ...>
 <operation name="test1">
 <input message="tns:test1"/>
 <output message="tns:test1Response"/>
 </operation>
 ...
</portType>
...
```

The parameter is mapped to the wrapper child element of `maxOccurs="unbounded"`.

If you execute the `cjwsimport` command by specifying this WSDL, the Java method of the generated service class is as follows:

```
@WebMethod
public String test1( java.util.List<Integer> arg0 );
```

The wrapper child element of `maxOccurs="unbounded"` is mapped to the `java.util.List` class. Also, in this case, the `xsd:int` type is mapped to the `java.lang.Integer` class.

(2) When the Java method has only one OUT parameter

The example assumes that a Web Service is to be developed starting from SEI with the following Java method. This Java method has only one `OUT` parameter and does not have a return value.

```
@WebMethod
public void test1( @WebParam(mode=WebParam.Mode.OUT) Holder<String> param1 );
```

The following is a part of the WSDL that is mapped in this case:

```
...
<types>
 <xsd:schema targetNamespace="http://cosminexus.com/jaxws">

 <xs:element name="test1" type="tns:test1"/>

 <xs:element name="test1Response" type="tns:test1Response"/>

 <xs:complexType name="test1">
 <xs:sequence/>
 </xs:complexType>

 <xs:complexType name="test1Response">
 <xs:sequence>
 <xs:element name="arg0" type="xs:string" minOccurs="0"/>
 </xs:sequence>
 </xs:complexType>
 </xs:element>
</types>
<message name="test1">
 <part name="parameters" element="tns:test1"/>
</message>

<message name="test1Response">
 <part name="parameters" element="tns:test1Response"/>
</message>

<portType ...>
 <operation name="test1">
 <input message="tns:test1"/>
 <output message="tns:test1Response"/>
 </operation>
 ...
</portType>
...
```

The `OUT` parameter is mapped to the wrapper child element referenced from the `wsdl:output` element.

If you execute the `cjwsimport` command by specifying this WSDL, the Java method of the generated service class is as follows:

```
@WebMethod
public String test1();
```

Since only one wrapper child element is referenced from the `wsdl:output` element, that wrapper child element is mapped to the return value.

(3) non-wrapper style arrays

This example assumes the development of a Web Service starting from SEI using the following Java method. This Java method has the non-wrapper style and array parameters of the `java.lang.String` class.

```
@WebMethod
@javax.jws.soap.SOAPBinding(
  parameterStyle=javax.jws.soap.SOAPBinding.ParameterStyle.BARE)
public String test1( String[] param1 );
```

The following is a part of the WSDL that is mapped in this case:

```
...
<types>
```

```
  <xsd:schema targetNamespace="http://jaxb.dev.java.net/array">
    <xsd:complexType name="stringArray" final="#all">
      <xsd:sequence>
        <xsd:element name="item" type="xsd:string" minOccurs="0"
          maxOccurs="unbounded" nillable="true" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:schema>

  <xsd:schema targetNamespace="http://cosminexus.com/jaxws"
    xmlns:ns1="http://jaxb.dev.java.net/array">
    <xsd:element name="test1" nillable="true" type="ns1:stringArray"/>
    <xsd:element name="test1Response" nillable="true" type="xsd:string"/>
  </xsd:schema>
</types>

<message name="test1">
  <part name="test1" element="tns:test1" />
</message>

<message name="test1Response">
  <part name="test1Response" element="tns:test1Response" />
</message>

<portType ...>
  <operation name="test1">
    <input message="tns:test1" />
    <output message="tns:test1Response" />
  </operation>
  ...
</portType>
...
```

The parameter is mapped to the `wrapper` child element of the `{http://jaxb.dev.java.net/array}stringArray` type.

When you execute the `cjwsimport` command by specifying the WSDL, the Java method of the generated service class will change as follows:

```
@WebMethod
@javax.jws.soap.SOAPBinding(
  parameterStyle=javax.jws.soap.SOAPBinding.ParameterStyle.BARE)
public String test1(net.java.dev.jaxb.array.StringArray test1);
```

The `wrapper` child element of the `{http://jaxb.dev.java.net/array}stringArray` type is mapped to the `net.java.dev.jaxb.array.StringArray` class.

10.6 Issuing the Meta data

The JAX-WS engine on the Web Service can issue the WSDL file coding the Meta data of the Web Service (Web Service Implementation Class or Provider Implementation Class) according to the request. The issued WSDL file can be used to generate the Java code required for developing the Web Service and the Web Service client by using the `cjwsimport` command.

This section describes the points to be noted when Meta data issuing is used.

(1) Conditions for issuing the Meta data

This section describes the respective conditions for issuing POJO and EJB Web Services.

(a) In POJO Web services

The following table lists the conditions for issuing the Meta data for POJO Web services. When the JAX-WS engine on the Web Service receives an HTTP request that fulfills all the conditions described in the following table, the Meta data is issued:

Table 10-10: HTTP request required for issuing the Meta data for POJO Web services

No.	Items		Conditions
1	HTTP method		GET method
2	URL	Schema	http or https
3		Host name (`: port number`)	Host name (and port number) where the Web Service requesting the issuing of the Meta data exists
4		Context path	Context path of the Web application that contains the Web Service that requests the issuing of the Meta data
5		Web Service name	Web Service that requests the issuing of the Meta data (service name of the Web Service Implementation Class or the Provider Implementation Class)
6		Query string	`wsdl` or `WSDL` (The characters are case sensitive)

The WSDL file associated with the Web Service corresponding to the requested URL is issued in the request source as HTTP response contents. The following query strings must be added in the request URL:

```
GET http://sample.com:8085/fromjava/AddNumbersImplService?wsdl HTTP/1.1
GET http://sample.com:8085/fromjava/AddNumbersImplService?WSDL HTTP/1.1
```

(b) In EJB Web Services

The following table lists the conditions for issuing the Meta data for EJB Web Services. When the JAX-WS engine on the Web Service receives an HTTP request that satisfies all the conditions described in the following table, the Meta data is issued:

Table 10-11: HTTP request required for issuing the Meta data for EJB Web services

No.	Item		Condition
1	HTTP method		GET method
2	URL	Schema	http or https
3		Host name (: port number)	Host name (and port number) where the Web Service requesting the issuing of the Meta data exists
4		Context path	Context path of the Web application that includes the Web Service that requests the issuing of the Meta data

No.	Item		Condition
5	URL	Web Service name	Web Service that requests the issuing of the Meta data (service name of the Web Service Implementation Class)
6		EJB class name	EJB class name of the Web Service that requests the issuing of the Meta data
7		Query string	"wsdl" or "WSDL" (Case sensitive)

The WSDL file associated with the Web Service corresponding to the requested URL is issued in the request source as HTTP response contents. The following query strings must be added in the request URL:

GET `http://sample.com:8085/statelessjava/AddNumbersImplService/AddNumbersImpl?wsdl HTTP/1.1`

GET `http://sample.com:8085/statelessjava/AddNumbersImplService/AddNumbersImpl?WSDL HTTP/1.1`

(2) Issued Meta data

The following table describes the correspondence between the request conditions and the issued Meta data:

Table 10-12: Correspondence between the request conditions and the issued Meta data

No.	Request conditions	Issued Meta data	Whether Web Service is applied	
			POJO	EJB#
1	If the `wsdlLocation` attribute exists in the `javax.jws.WebService` annotation (for a Web Service Implementation Class) or in the `javax.xml.ws.WebServiceProvider` annotation (for a Provider Implementation Class)	The WSDL file present in the location specified in the `wsdlLocation` attribute is returned.	Y	Y
2	If the `wsdlLocation` attribute is not specified in the `javax.jws.WebService` annotation (for a Web Service Implementation Class) or in the `javax.xml.ws.WebServiceProvider` annotation (for a Provider Implementation Class), but if the WSDL file that has the `wsdl:service` element exists in the `WEB-INF/wsdl` directory of the deployed WAR file	The WSDL file exists in the `WEB-INF/wsdl` directory of the deployed Web application is returned.	Y	--
3	If WSDL does not exist in the Web application (For other than No.1 and No.2)	A new WSDL is generated and returned, if the target is a Web Service Implementation Class. The Meta data is not issued, if the target is a Provider Implementation Class.	Y	Y

The pre-condition forLLegend:

Y: Applied

--: Not applied

Note:

In EJB Web Service, the Meta data is applicable only for Web Service Implementation Class.

The pre-condition for the issue of Meta data is that the Web Service application does not have an error, is deployed normally, and the execution is started.

The precautions related to the issuing of the Meta data are as follows:

- In the `wsdlLocation` attribute, specify the relative path to the WSDL file beginning with 'WEB-INF/wsdl' or 'META-INF/wsdl'. If the WSDL file specified in the `wsdlLocation` attribute does not exist in the `WEB-INF/wsdl` or `META-INF/wsdl` directory nor has an invalid WSDL syntax, an error occurs during the deployment of the Web Service.
- If the WSDL file specified in the `wsdlLocation` attribute does not include the `wsdl:service` element, an error occurs during the deployment of the Web Service.
- Even for point 1 and point 2, if the WSDL file that exists under the `WEB-INF/wsdl` or `META-INF/wsdl` directory includes one of the following, an error occurs during the deployment of the Web Service:
  - There are multiple files with the `wsdl:service` element.
  - There are multiple files with the `wsdl:portType` element (in a Web Service Implementation Class).
- Even for point 1 and 2, the requested WSDL of the Web application is returned with the updated contents based on the WSDL files.

With the Provider Implementation Class, basically you do not need a WSDL. Also, the standard specifications do not define the mapping rules for the `javax.xml.ws.WebServiceProvider` annotation and the WSDL. Therefore, for issuing the Meta data, you must properly include the created WSDL in a WAR file (unlike the Web Service Implementation Class in which the JAX-WS engine automatically generates a WSDL file). Also, if you do not include `cosminexus-jaxws.xml` in the WAR file, the `portName` and `targetNamespace` attributes of the `javax.xml.ws.WebServiceProvider` annotation will be required. Specify appropriate values according to the definition contents of the WSDL. For `cosminexus-jaxws.xml`, see the section *10.3 Customization using cosminexus-jaxws.xml*.

(3) Updating the WSDL

When the Web Service is deployed, the JAX-WS engine at the Web service machine automatically generates the WSDL file for the Web Service Implementation Class, as and when required. Even when the JAX-WS engine does not automatically generate a WSDL file, the JAX-WS engine returns a WSDL containing the following information based on the WSDL that is included in the WAR file:

- Header information

  In the WSDL file, add the published version and the date and time as header information. The following is an example of the added header information:

  ```
  <!-- Published by Cosminexus JAX-WS 0900 (2012.01.01 00:00). -->
  ```

- Location information

  Update the `location` attribute of the `soap:address` element and the `schemaLocation` attribute of the `xsd:include` element.

  For example, even when the value of the `location` attribute of the `soap:address` element is a WSDL such as `REPLACE_WITH_ACTUAL_URL`, this value is included in the WAR file, and is updated to an appropriate URL.

  The following is an example of the location information after the update:

  - Host name: `sample.com`
  - Context root: `/fromjava`
  - Web Service invocation URL: `/AddNumbersImplService`

  In such cases, the URL changes to `http://sample.com/fromjava/AddNumbersImplService` after the update.

(4) Enabling and disabling Meta data issue

You can specify the enabling or disabling for issuing the Meta data in the value of the `com.cosminexus.jaxws.security.publish_wsdl` property. You can specify in both, POJO and EJB Web Services.

For the com.cosminexus.jaxws.security.publish_wsdl property, see *10.1.2 Settings for the common definition file*.

(5) Notes when the WAR file contains multiple Web Service Implementation Classes or Provider Implementation Classes

The Meta data is acquired for each Web Service Implementation Class or Provider Implementation Class. If the WAR file contains multiple Web Service Implementation Classes or Provider Implementation Classes, the wsdl:binding element and the wsdl:port element of the WSDL definition generated and returned anew using the JAX-WS engine on the Web Service are only the wsdl:binding element and the wsdl:port element corresponding to the Web Service Implementation Class or the Provider Implementation Class of the URL specified in the request. The wsdl:binding element and the wsdl:port element corresponding to the other Web Service Implementation Classes or the Provider Implementation Classes that the WAR file contains are not included.

If you want to publish Meta data containing the wsdl:binding element and the wsdl:port element corresponding to all the Web Service Implementation Classes or the Provider Implementation Classes in the WAR file, the Web Service (application) developer must first create appropriate Meta data and then include the data in the WEB-INF/wsdl directory.

(6) Notes for importing and including WSDL definition or XML Schema

When importing or including the WSDL definition or XML Schema, if WSDL definition or XML Schema used for developing a Web Service is included in the WAR file as is, the Meta data might not be issued normally.

To include a file you want to import or include in the WAR file, you must check the path information included in the WSDL definition or XML Schema at the import source or include source and use the following methods to modify the path information appropriately:

- To include a WSDL definition or XML Schema that contains relative path coding in the WAR file
  Include the file you want to import or include under the WEB-INF/wsdl directory of the WAR file and modify the path information specified in the location attribute, schemaLocation attribute of the WSDL definition or XML Schema at the import source and include source to an appropriate relative path suitable to the environment.
- To include a WSDL definition or XML Schema that contains remote URL coding in the WAR file
  No modification is required for a URL that can be accessed from the Web Service client that acquires the WSDL definition. For a URL that cannot be accessed, include the file you want to import or include under the WEB-INF/wsdl directory of the WAR file and modify the path information specified in the location attribute, schemaLocation attribute of the WSDL definition or XML Schema at the import source and include source to an appropriate relative path suitable to the environment.
- To include a WSDL definition or XML Schema that contains local URL coding in the WAR file
  Include the file you want to import or include under the WEB-INF/wsdl directory of the WAR file and modify the path information specified in the location attribute and in the schemaLocation attribute of the WSDL definition at the import source and then include source to an appropriate relative path suitable to the environment.

(7) Notes on the transport attribute of the SOAP 12:binding element of the WSDL

In the SOAP 1.2 compatible WSDL that is automatically generated and issued, the transport attribute of the soap12:binding element is the following URL by default:

```
http://www.w3.org/2003/05/soap/bindings/HTTP/
```

If you want to change the above URL to the following URL, add the definition to the operation definition file.

```
http://schemas.xmlsoap.org/soap/http
```

Add the following definition:

```
com.cosminexus.jaxws.publish_wsdl.soap12binding=WSI_BP20_TRANSPORT
```

10.7 Displaying Web Service information

Execute the URL that invokes the Web Service Implementation Class or the Provider Implementation Class on the browser to display the Web Service information.

(1) Web Service information displayed

To display the Web Service information, specify the URL using the `GET` method. The following table describes the Web Service information displayed by using the HTTP `GET` method:

Table 10-13: Web Service information displayed

EndPoint	Information
Service Name: `QName` for service Port Name: `QName` for port	Address: URL for invoking the Web Service WSDL: URL for acquiring the WSDL for the Web Service For issuing the Meta data, see the section *10.6 Issuing the Meta data*. Implementation Class: Name of the Web Service Implementation Class or the Provider Implementation Class

(2) Method of displaying the Web Service information

To display the Web Service information, send the HTTP request containing a URL, such as the one shown in the following example, to the JAX-WS engine on the Web Service using the `GET` method:

```
http://sample.com/fromjava/AddNumbersImplService
```

The following information is specified in this URL:

- `sample.com`: Host name
- `/fromjava`: Context root
- `/AddNumbersImplService`: Web Service invocation

You can also enable or disable this functionality using the `com.cosminexus.jaxws.security.display_webservice_info` property. For the `com.cosminexus.jaxws.security.display_webservice_info` property, see *10.1.2 Settings for the common definition file*.

10.8 HTTP methods that can be used

You can use `POST` as the HTTP method. You can also use `GET` for issuing the Meta data and displaying Web Service information. You can use HTTP method in both the POJO and EJB Web Services.

When the following HTTP request methods arrive at the JAX-WS engine on the Web Service, the HTTP status code `405 Method Not Allowed` is returned:

- `GET` (excluding the issuing of the Meta data and displaying of the Web Service information)
- `DELETE`
- `HEAD`
- `OPTION`
- `PUT`
- `TRACE`

10.9 Initializing and destroying the Web Service

This section describes the initialization and destruction of the Web Service. This is applicable to both, POJO and EJB Web Services.

(1) Initializing the Web Service

In the JAX-WS engine on the Web Service, when the J2EE server starts the J2EE application, the Web Service is initialized. Specifically, the following processing is executed:

- Generation of mapping the information between the URL and the Web Service Implementation Class or the Provider Implementation Class for performing Web Service discovery
- If a WAR file does not contain request beans, response beans, or the `JavaBeans` classes of fault beans required to start POJO Web Services or if an EAR file does not contain request beans, response beans, or the EAR file does not contain the `JavaBeans` classes of fault beans required to start an EJB Web Service, the JAX-WS engine dynamically generates the `JavaBeans` classes.
- Checking of WSDL for issuing the Meta data (for a Web Service Implementation Class)
- Reading of the handler chain setup file and initialization of the handler when the handler is associated with the Web Service Implementation Class or the Provider Implementation Class

  (Also, includes the invocation of the method annotated using the `javax.annotation.PostConstruct` annotation if such a method exists)

When the initialization of the Web Service starts, the KDJW40001-I message is output in the log and standard output. If Web Service initialization ends normally, the KDJW40003-I message is output in the log and standard output. If an error occurs during Web Service initialization and if the initialization fails, the KDJW40002-E message and the error message that caused the error is output in the log and standard error output.

If an error occurs during Web Service initialization, an error is output in the log, but the startup of the J2EE application continues and ends normally. However, since the initialization failed, the deployed Web Service does not work. In this case, check if the KDJW40002-E message is output. If an error is output, correct the problem and deploy the Web Service again.

When starting a J2EE server, it might be difficult to investigate in a J2EE application why Web Service could not be initialized, when the deployed J2EE application is started automatically. In such cases, you can identify the J2EE application name and the context root name by checking logs shown in the section *39.3.1 Types of log*, and operation logs of the J2EE server.

The following is an example of checking the output contents of the message KDJE39103-E including `com.cosminexus.xml.ws.transport.http.servlet.WSServletContextListener`:

```
0095 2009/12/18 13:48:36.471     HEJB             0125FEFA 004413EE KDJE39103-
E          An exception javax.xml.ws.WebServiceException was raised in notification
of the listener class
com.cosminexus.xml.ws.transport.http.servlet.WSServletContextListener. (J2EE
application = Sample_application_fromwsdl, context root = /fromjava)
```

When a log such as the one shown above is output, an attempt to initialize the Web Service within the WAR file that is associated with the context root name `fromjava` and included in the J2EE application `Sample_application_fromwsdl` will fail.

If there is a problem with the user program, basically an error corresponding to the message KDJE39103-E occurs. However, check the following error messages also, as and when required:

- `KDJE39100-E`
- `KDJE39101-E`
- `KDJE39102-E`

For the operation logs of the J2EE server, see *5.2 Log contents of Application Server* in the *uCosminexus Application Server Maintenance and Migration Guide*. Also for the above messages, see *7.2. Messages from KDJE30000 to KDJE39999* in the manual *uCosminexus Application Server Messages*.

(2) Destroying the Web Service

In the JAX-WS engine on the Web Service, when the J2EE server ends the J2EE application, the Web Service is destroyed. Specifically, the following processing is executed:

- Deletion of mapping information between the URL and the Web Service Implementation Class or the Provider Implementation Class for performing Web Service discovery
- Destruction of the handler when the handler is associated with the Web Service Implementation Class or the Provider Implementation Class
  (Also includes the invocation of the method annotated using the `javax.annotation.PreDestory` annotation if such a method exists)

When the destruction of the Web Service starts, the KDJW40004-I message is output in the log and standard output. If Web Service destruction ends successfully, the KDJW40006-I message is output in the log and standard output. If an error occurs during Web Service destruction and if the destruction fails, the KDJW40015-E message and the error message that caused the error is output in the log and standard error output.

If an error occurs during Web Service destruction, an error is output in the log, but the end processing of the J2EE application continues and ends normally. In this case, check if the KDJW40015-E message is output and if an error is output, correct the problem.

10.10 Connecting through a proxy server

You can use a Web Service on an external network from a Web Service client through a proxy server.

This section describes the property settings required for connecting externally through a proxy server.

(1) Specifying the property values

To access a Web Service through a proxy server, specify the *JavaVM properties* or the properties specific to the Cosminexus JAX-WS functionality, and set up the proxy server information. The following table describes the properties and their specified contents, when establishing a connection through a proxy server:

Table 10-14: Properties used for connecting through a proxy server

No.	Properties	Specified contents	For non-SSL	For SSL
1	`http.proxyHost`[#1]	Specify the host name or IP address of the proxy server. If a null character is specified, connection is not established with the proxy server.	Y	--
2	`http.proxyPort`[#1]	Specify the port number of the proxy server. If a null character is specified in `http.proxyPort` when `http.proxyHost` is set up correctly, the 80th port of the host specified in `http.proxyHost` is accessed. If `http.proxyHost` is not specified, connection is not established with the proxy server even if `http.proxyPort` is specified.	O	--
3	`com.cosminexus.jaxws.http.proxyUser`[#2]	Specify an authentication user ID of the proxy server. If the `http.proxyHost` property and the `http.proxyPort` property are specified properly, connect to the proxy server anonymously when a null character is specified in the `com.cosminexus.jaxws.http.proxyUser` property.	O	--
4	`com.cosminexus.jaxws.http.proxyPassword`[#2]	Specify a password corresponding to the authentication user ID of the proxy server. If the following properties are specified properly, connect to the proxy server without specifying a password, when a null character is specified in the `com.cosminexus.jaxws.http.proxyPassword` property: • `http.proxyHost` • `http.proxyPort` • `com.cosminexus.jaxws.http.proxyUser`	O	--
5	`https.proxyHost`[#1]	Set up the host name or IP address of the proxy server to be used for connection by SSL protocol[#3]. To use a proxy server for connection by SSL protocol, make sure that you set up this property. Note that if a null character is specified, connection is not established with the proxy server.	--	Y
6	`https.proxyPort`[#1]	Set up the port number of the proxy server to be used for connection by SSL protocol[#3]. Note that if a null character is specified in `https.proxyPort` when `https.proxyHost` is set up correctly, the 443rd port	--	O

No.	Properties	Specified contents	For non-SSL	For SSL
6	`https.proxyPort`#1	of the host specified in `https.proxyHost` is accessed. If `https.proxyHost` is not specified, connection is not established with the proxy server even if `https.proxyPort` is specified.	--	O
7	`com.cosminexus.jaxws.https.proxyUser`#2	Set an authentication user ID of the proxy server to be used for connecting through the SSL protocol#3. If the `https.proxyHost` property and the `https.proxyPort` property are set properly, connect to the proxy server anonymously, when a null character is specified in the `com.cosminexus.jaxws.https.proxyUser` property.	--	O
8	`com.cosminexus.jaxws.https.proxyPassword`#2	Specify a password corresponding to the authentication user ID of the proxy server to be used for connecting through the SSL protocol#3. If the following properties are specified properly, connect to the proxy server without specifying a password, when a null character is specified in the `com.cosminexus.jaxws.https.proxyPassword` property: • `https.proxyHost` • `https.proxyPort` • `com.cosminexus.jaxws.https.proxyUser`	--	O
9	`http.nonProxyHosts`#1	Specify the host names that do not use the proxy server as required. When connecting to the host specified in this property, the proxy server specified in `http.proxyHost` is not used. To specify multiple hosts, use '|' as a separator. You cannot specify a character other than '|' (such as a space) between two host names.	O	O

Legend:

Y: Indicates that the specification of the property is required.

O: Indicates that the property is specified as required.

--: Indicates that the property need not be specified.

#1

System properties supported by the JavaVM in the standard manner. For the system properties of JavaVM, see the *JavaVM documentation*.

#2

The properties are specific to the Cosminexus JAX-WS functionality, and are simple properties. To execute detailed control, we recommend that you perform the implementation using the `java.net.Authenticator` class of the J2SE 6.0 standard in the Web Service client. For details, see the subsection *10.10(3) When not using properties specific to the JAX-WS functionality*.

#3

For connecting through the SSL protocol, see the section *10.11 Connection by SSL protocol*.

(2) How to set properties

Set up the properties specific to the Cosminexus JAX-WS functionality in the action definition file. For setting up the action definition files, see the section *10.1 Action definition file*. When using the properties specific to the JAX-WS functionality, note the contents described in the subsection *10.10(4) Notes on using properties specific to the JAX-WS functionality*.

How to set up the system properties of JavaVM differs depending on the execution of the Web Service client.

- When the Web Service client is executed using commands
  When the Web Service client is executed using commands (`cjclstartap`), set up the JavaVM property in the user property file for Java application (`usrconf.properties`) for the `cjclstartap` command.
- When the Web Service client is executed on the J2EE server
  When the Web Service client is executed on the J2EE server, set up the JavaVM property in the user property file for J2EE server (`usrconf.properties`).

The following is an example of property settings:

```
http.proxyHost=10.209.15.79
http.proxyPort=3128
https.proxyHost=10.209.15.79
https.proxyPort=3128
http.nonProxyHosts=10.209.15.80|www.hitachi.co.jp
```

There are no rules for the positions to add property settings.

(3) When not using properties specific to the JAX-WS functionality

The properties specific to the JAX-WS functionality are the simple properties. Therefore, for executing the detailed control, we recommend that you perform the implementation using the `java.net.Authenticator` class of the J2SE 6.0 standard with the Web Service client. For details, see the *J2SE 6 documentation*. The following is an example of the implementation using the `java.net.Authenticator` class:

```
java.net.Authenticator.setDefault( new java.net.Authenticator(){

  // Override the getPasswordAuthentication method
  public java.net.PasswordAuthentication getPasswordAuthentication() {

    // Set the user name
    String userName = ...

    // Set the password
    char[] password = ...

    // Generate PasswordAuthentication.
    java.net.PasswordAuthentication auth =
        new java.net.PasswordAuthentication( userName, password );

    return auth;
  }
} );
```

(4) Notes for using properties specific to the JAX-WS functionality

With the Cosminexus server, specify the value of the properties specific to the JAX-WS functionality in JavaVM using the `setDefault()` method of the `java.net.Authenticator` class of the Java SE standard. Therefore, note the following:

- Valid range
  The properties specific to the JAX-WS functionality are enabled in the entire process in which the Web Service client is operating (or an entire J2EE server if the Web Service client is operating on the J2EE server), and also affects HTTP connections other than the Cosminexus server. If you do not want to apply proxy settings in HTTP connections other than Cosminexus through properties specific to the JAX-WS functionality, you must perform implementation using the `setDefault()` method of the `java.net.Authenticator` class in the user program (Web Service client) instead of the property specific to the JAX-WS functionality. For details, see the subsection *10.10(3) When not using properties specific to the JAX-WS functionality*.
- Conflict
  When using a property specific to the JAX-WS functionality, specify the settings in such a way so that no product other than Cosminexus invokes the `setDefault()` method of the `java.net.Authenticator` class, in the process in which the Web Service client is operating. Take special care when using any other product in the library. Depending on the timing of invoking the `setDefault()` method of the `java.net.Authenticator` class, the settings might be in conflict and the operation might become invalid.

When any other product such as a user program and the library invoke the `setDefault()` method of the `java.net.Authenticator` class, do not use properties specific to the JAX-WS functionality.

- Security exception
  When the `setDefault()` method of the `java.net.Authenticator` class throws `java.lang.SecurityException`, the message `KDJW10025-W` is output to a log, and the processing continues. Check the detailed message and remove the cause of the error, as and when required.

10.11 Connection by SSL protocol

From the Web Service client, you can connect with a Web Service that supports the SSL protocol.

This section describes the property settings required for connection by the SSL protocol.

(1) Specifying the property values

To access the Web Service with SSL protocol, specify the values for the properties supported in JDK and set up the information about the SSL protocol. The following table describes the properties for connection by SSL protocol and the specified contents.

Table 10-15: Properties for connection by SSL protocol

No.	Properties	Specified contents
1	`javax.net.ssl.trustStore`	Specify trust store.
2	`javax.net.ssl.trustStorePassword`	Specify the password for trust store.

Specify these properties as and when required. If trust store is not specified, the default value such as *JDK-installation-directory*/lib/security/jssecacerts is used.

For the JDK properties, see the *JDK documentation*.

(2) Method of specifying the properties

To enable the value specified for a property, specify the properties in the system property. How to set up a property differs depending on the execution of the Web Service client.

- When executing the Web Service client using commands

  When you execute the Web Service client using the command (`cjclstartap`), set up the properties of JavaVM in the user property file for Java applications (`usrconf.properties`).
- When executing the Web Service client on J2EE servers

  When you execute the Web Service client on J2EE servers, set up the properties of JavaVM in the user property file for J2EE server (`usrconf.properties`).

The following is an example of setting the properties:

```
javax.net.ssl.trustStore=trust-store
javax.net.ssl.trustStorePassword=trust-store-password
```

There is no fixed location for adding the settings of the properties.

(3) Notes on validating the host name

When connecting from a Web Service client to a Web Service supporting the SSL protocol, validate whether the host name to be included in the end point address matches with the host name in the certificate. The `HostnameVerifier` to be used is the default implementation of JDK. For the operation of the default `HostnameVerifier` of JDK, see the *JDK documentation*.

You can skip the host name validation by configuring the settings in the action definition file or message context. The following table describes the properties and the specified content for skipping the host name validation.

Table 10-16: Properties for skipping host name validation

No.	Property	Specified contents	Mandatory
1	`com.cosminexus.xml.ws.client.http.HostnameVerificationProperty`	For skipping the host name validation, specify `true`. When the validation is not to be skipped, specify `false`.	O

Legend:

O: Indicates that the property is to be specified as and when required.

Notes on skipping the host name validation

- The specification to a message context is enabled only when calling Web Service and the specification is not enabled when acquiring the meta data (WSDL) generated when creating the `javax.xml.ws.Service` class before calling Web Service.

  For setting whether to verify a host name when acquiring the meta data, either code in a common definition file or a process wise definition file, or download and use a separate WSDL on the local machine (If you use a WSDL existing on the local machine, connection to a remote machine is not established when you acquire the meta data). If a WSDL to be imported separately from a WSDL already exists, also download the WSDL to be imported on the local machine.

- If the application of the host name validation differs among the multiple Web Service clients operating in a process, do not include the properties in a process wise definition file or a common definition file. Include the properties only in the message context.

  Similarly, if the application of the host name validation differs among multiple processes operating on a system, do not include the properties in a common definition file. Include the properties only in a process wise definition file or in the message context.

For details on how to set up the properties in an action definition file, see *10.1.2 Settings for a common definition file*.
For details on how to set up the properties in the message context, see *19.2.5 Using the message context*.

10.12 Connection by basic authentication

From the Web Service client, you can connect with the Web Service that supports basic authentication.

This section describes the property settings required for connection by basic authentication.

(1) Specifying the property values

To access the Web Service with basic authentication, specify the value in the action definition file or message context. The following table describes the properties for connection by basic authentication and the specified contents:

Table 10-17: Properties for connection by basic authentication

No.	Properties	Specified contents	Required
1	`javax.xml.ws.security.auth.username`	Set up the user ID.	Y
2	`javax.xml.ws.security.auth.password`	Set up the password.	Y

Legend:

Y: Indicates that the specification of the property is required.

Precautions for connection by basic authentication

- The specification to the message context is enabled only when the Web Service is invoked and is not applied when the Meta data (WSDL) that occurs during the generation of the `javax.xml.ws.Service` class before Web Service invocation, is acquired.

  To set up basic authentication information during Meta data acquisition, specify the coding in the common definition file or process-wise definition file or separately download WSDL on the local machine and use it (when WSDL exists on the local machine is used, connection is not established with the remote machine when the Meta data is acquired). If WSDL separately imported from WSDL exists, download the imported WSDL on to the local machine.

- If performing basic authentication or not depends on multiple Web Service clients running with the same process, do not include these properties in the process-wise definition file or common definition file and include the properties only in the message context.

  Similarly, if performing basic authentication or not depends on multiple processes running on the same system, do not include these properties in the common definition file and include the properties only in the process-wise definition file or the message context.

(2) Method of specifying the properties

For specifying the properties in the action definition file, see *10.1.2 Settings for the common definition file*. For specifying the properties in the message context, see *19.2.5 Using the message context*.

10.13 Selecting the SOAP version

This section describes the selection of the SOAP version that is required for developing Web Services and Web Service applications.

10.13.1 Selecting the SOAP version (when developing Web Services)

This subsection describes how to select the version for developing Web Services starting from a WSDL, SEI, and provider.

(1) Starting from a WSDL

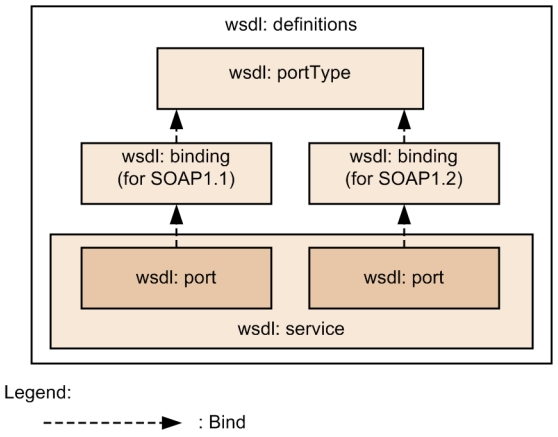
To send or receive messages of the SOAP 1.1 specifications or the SOAP 1.2 specifications is determined from the binding coded in the WSDL.

The skeletons of the Web Service Implementation Class generated using the `cjwsimport` command exists for each binding, so the skeletons are dedicated to either SOAP 1.1 specifications or the SOAP 1.2 specifications. You cannot dynamically change the skeletons during the execution.

On the other hand, as you can code multiple bindings in a WSDL, you can mix the ports of the SOAP 1.1 and SOAP 1.2 specifications in a single WSDL.

The following is an example of mixing the ports of the SOAP 1.1 and SOAP 1.2 specifications:

Figure 10−8: Example of mixing ports



With this example, bind a single port type separately to use with SOAP 1.1 and SOAP 1.2. This enables a single port type to receive messages of the format conforming to the SOAP 1.1 and SOAP 1.2 specifications.

(2) Starting from SEI

Use annotations to specify whether to bind to the SOAP 1.1 specifications or the SOAP 1.2 specifications. Specify an annotation for each Web Service Implementation Class. You cannot dynamically change the annotations during the execution.

You can omit annotations. If omitted, the SOAP 1.1 specifications will be applicable.

(3) Starting from a provider

Use annotations to specify whether to bind with the SOAP 1.1 specifications or the SOAP 1.2 specifications. Specify an annotation for each Provider Implementation Class. You cannot dynamically change the annotations during the execution.

10.13.2 Selecting the SOAP version (when developing a Web Service client)

You develop a Web Service client based on the Meta data of Web Services (such as the WSDL or the information dependent on the Web Service to which a connection is to be established). Based on the information defined in the Meta data, implement a Web Service client that communicates using an appropriate version of SOAP.

This subsection describes how to select a SOAP version when developing stub-based, dispatch-based, and API-based Web Service clients.

(1) Stub-based Web Service clients

When you execute the `cjwsimport` command, appropriate stubs are generated automatically based on the definition of the WSDL. Therefore, you need not specify whether to use the SOAP 1.1 specifications or the SOAP 1.2 specifications for communication.

Note that you always bind a port to a single binding. Therefore, the port acquisition method included in the service class is dedicated to either the SOAP 1.1 specifications or the SOAP 1.2 specifications.

(2) Dispatch-based Web Service clients

Select an appropriate SOAP version based on the Meta data of Web Services, and specify with APIs.

(3) API-based Web Service clients

Select an appropriate SOAP version based on the Meta data of Web Services, and specify with the APIs.

10.13.3 Selecting the SOAP version (during the execution)

This subsection describes how to select a SOAP version during the execution of Web Services and Web Service clients.

(1) For Web Services

The Web Service Implementation Classes and Provider Implementation Classes are dedicated to either the SOAP 1.1 specifications or the SOAP 1.2 specifications based on what you select during the development. Therefore, you can receive messages of either the SOAP 1.1 specifications or the SOAP 1.2 specifications. For receiving the messages corresponding to both the SOAP versions, you must define or implement multiple Web Service Implementation Classes or Provider Implementation Classes.

(2) For Web Service clients

A stub-based Web Service client sends messages of either the SOAP 1.1 specifications or the SOAP 1.2 specifications. If either of the following conditions becomes applicable due to an error during the development, SOAP Messages with mismatched SOAP versions will be sent, and an error will occur at the Web Service machine:

- If you generate the stubs for the SOAP 1.1 specifications, when the Web Service binding corresponds to the SOAP 1.2 specifications
- If you generate the stubs for the SOAP 1.2 specifications, when the Web Service binding corresponds to the SOAP 1.1 specifications

With a dispatch-based and an API-based Web Service client, specify the SOAP version with the APIs. You can dynamically make changes regarding the SOAP version of messages to be sent based on the implementation method, such as by enabling changes to the specified value in the configuration file.

10.14 Executing a client application using the command line

In a Web Service, a Java application running on the command line can be used as a client application.

This section describes the settings required for using a client application through a command line, examples of command line specification, and precautions.

10.14.1 Settings for command line usage

This subsection describes the settings required for using a client application through a command line.

(1) Settings for the option definition file for Java application

In the current directory where the client application is executed, create the option definition file for Java application and add the following keys and values:

- `add.class.path=`*JAR-or-directory-path-where-the-class-file-of-the-client-Web-Service-is-stored*
- `add.class.path=`*Cosminexus-installation-directory*`/jaxws/lib/cjjaxws.jar`
- `add.jvm.arg=-Dcosminexus.home=`*Cosminexus-installation-directory*
- `add.jvm.arg=-Dejbserver.server.prf.PRFID=`*PRF identifier* #
- `ejb.client.log.directory=` *directory-path-to-output-log-file*

#

Specify the identifier same as the PRF identifier specified using the `cprfstart` command. The default value when the identifier is not specified is PRF_ID.

(2) Settings for the user property file for Java application

In the current directory where the client application is executed, create the user property file for Java application and specify the settings as required.

(3) Adding a path

Add the following path in the environment variable PATH:

Cosminexus-installation-directory`/PRF/bin`

10.14.2 Executing a command line

This subsection describes the examples of specification when the Web Service client is executed from a command line.

(1) Starting the PRF daemon

The following is an example of specification when the default value is used as the PRF identifier:

Cosminexus-installation-directory`/PRF/bin/cprfstart`

The following is an example of specification when a value that is not a default value is used as the PRF identifier:

Cosminexus-installation-directory`/PRF/bin/cprfstart` *prfid*#

#

prfid indicates the PRF identifier. As the PRF identifier, specify the identifier same as the PRF identifier specified in the option definition file for Java application.

(2) Executing the application client

Executed using the Java application start command (`cjclstartap`). The following is an example of execution:

```
cjclstartap localhost.testMain
```

10.14.3 Precautions on using the command line

This subsection describes the precautions for using the command line.

- The log is output in the file path specified in the `ejb.client.log.directory` property of the option definition file for Java application. For log output, see *39.3.3(2) When using the Web Services client in the command line interface*.
- When using the command line and when collecting PRF trace, the PRF daemon must be started before executing the application. If the PRF daemon is not started, the processing continues without collecting the PRF trace.

10.15 HTTP status codes

The following table lists and describes the *HTTP status codes* returned by the JAX-WS engine at the service machine:

Table 10-18: HTTP Status Codes

Item no.	HTTP status code	Condition under which HTTP status code is returned
1	`200 OK`	When the invocation of the Request-Response operation type Web Service terminates successfully.
2	`202 Accepted`	When the invocation of Web Service corresponding to an asynchronous invocation terminates successfully, while you are using the addressing functionality or invoking Web Services for one-way operations.
3	`404 Not Found`	When the format of requesting is invalid, while the issue of the Meta data has been enabled. In such cases, the HTTP status code becomes `405 Method Not Allowed`. For issuing the Meta data, see the section *10.6 Issuing the Meta data*.
4	`405 Method Not Allowed`	In any of the following cases: • When an HTTP method that you cannot use arrives For the HTTP methods that you can use, see the section *10.8 HTTP methods that can be used*. • When you receive an HTTP request for issuing the Meta data, while issuing of Meta data has been disabled For issuing the Meta data, see the section *10.6 Issuing the Meta data*. • When you receive an HTTP request for displaying the information of Web Services, while the information display of Web Services has been disabled For displaying the information of Web Services, see the section *10.7 Displaying Web Service information*.
5	`415 Unsupported Media Type`	When the Content-Type HTTP header either does not exist or is invalid. Such cases result in the HTTP status code `500 Internal Server Error`.
6	`500 Internal Server Error`	When an error other than the errors described above occurs. This HTTP status code is also returned, when the execution results of a Web Service result in a SOAP fault.

10.16 HTTP header

This section describes the HTTP header of JAX-WS engine.

(1) action parameter of the Content-Type header

JAX-WS engine encloses the `action` parameter value of `Content-Type` header within (") double quotation marks and sets as the `action` parameter value of `Content-Type` header for HTTP request and HTTP response. Value already enclosed within (") double quotation marks is set as is as the `action` parameter value of `Content-Type` header.

Similar operations are executed in Web Service and Web Service client.

10.17 gzip compression of the HTTP request body

You can reduce the time required for the HTTP request communication between the Web Service client and the Web container by performing gzip compression of the HTTP request body. To compress the HTTP request body, you must attach the HTTP header indicating the sending of HTTP request body compressed in gzip format while sending the request message from the Web Service client. Implement the processing for attaching the HTTP header in the client application.

The following is an example of implementation in the client application:

```
Map<String, List<String>> httpHeaders =
  ( Map<String, List<String>> )context.get( MessageContext.HTTP_REQUEST_HEADERS );
if( null == httpHeaders ){
  httpHeaders = new HashMap<String, List<String>>();
}
List<String> contentEncondings = httpHeaders.get( "Content-Encoding" );
if( null == contentEncondings ){
  contentEncondings = new ArrayList<String>();
}
contentEncondings.add( "gzip" );
httpHeaders.put( "Content-Encoding", contentEncondings );
context.put( MessageContext.HTTP_REQUEST_HEADERS, httpHeaders );
```

10.18 Linking with the HTTP response compression functionality

The Cosminexus functionality for reducing the time required for the HTTP response communication between the Web container and the Web Service client by the gzip compression of the HTTP request body is called as *HTTP response compression functionality*.

You can link the JAX-WS engine with the HTTP response compression functionality. To link with the HTTP response compression functionality, you must attach the HTTP header indicating the receipt of compressed HTTP response in gzip format while sending the request message from the Web Service client. Implement the processing for attaching the HTTP header in the client application.

The following is an example of implementation in the client application.

```
Map<String, List<String>> httpHeaders =
  ( Map<String, List<String>> )context.get( MessageContext.HTTP_REQUEST_HEADERS );
if( null == httpHeaders ){
  httpHeaders = new HashMap<String, List<String>>();
}
List<String> acceptEncondings = httpHeaders.get( "Accept-Encoding" );
if( null == acceptEncondings ){
  acceptEncondings = new ArrayList<String>();
}
acceptEncondings.add( "gzip" );
httpHeaders.put( "Accept-Encoding", acceptEncondings );
context.put( MessageContext.HTTP_REQUEST_HEADERS, httpHeaders );
```

10.19 Invoking an EJB Web Service

This section describes the conditions for invoking EJB as a Web Service and available functions.

The EJB conditions are as follows:

- EJB version
  You can invoke EJB 3.0 or later versions as a Web Service.
- EJB type
  You can invoke EJB of stateless session Bean as a Web Service.
- Interface
  Business interface need not be provided in a EJB Web Service. You can share and use the home interface.
  When the EJB Web Service includes business interface, home interface, and component interface, you cannot invoke the method via Web Service through these interfaces.
  When EJB Web Service includes business interface, you can locally invoke EJB for the EJB Web Service. However, when the EJB Web Service does not include a business interface, you can invoke the EJB Web Service as a Web Service; however, you cannot invoke as an EJB.
  You can invoke the home interface as a Web Service only when it is specified by the `javax.ejb.RemoteHome` or the `javax.ejb.LocalHome` annotation. You cannot invoke the home interface as a Web Service if the home interface is specified by DD.

(1) EJB functions in EJB Web Service invocation

The following are the EJB functions that can be used concurrently when invoking the EJB as a Web Service. However, you can use these functions only in the EJB Web Service Implementation Class and not in a Handler Chain.

- Using interceptor
- CMT and BMT transaction management
- Access management by the `javax.annotation.security.PermitAll` and the `javax.annotation.security.DenyAll` annotation
- Resource connection
- Injecting a Web Services context by using the `javax.annotation.Resource` annotation
  For details on injecting a Web Services context, see *10.21.2 Injecting a Web Services context*.
- Timer Service

The following functions cannot be used concurrently when invoking the EJB as a Web service.

- Inheriting transaction context from client
- Inheriting security context from client

(2) Application Server functions available when invoking EJB Web Service

JAX-WS can concurrently use Application Server functions that can be set by using `cosminexus.xml` when invoking EJB as a Web Service. The following are the available Application Server functions:

- Setting the number of concurrently executed threads.
- Setting the pending queue size
- Setting the security role

Set the WAR file name for using these functions. For the WAR file name settings, see *3.5.4(3) Name of the WAR file for settings*. Note that you can use the functions for settings other than the `cosminexus.xml` settings, in Web applications other than the EJB Web Services.

10.20 Preventing the resending of a request by sun.net.www.http.HttpClient

The JAX-WS engine on the Web Service client machine executes communication by using the HTTP client implementation of JDK. Contrary to RFC 2616, the HTTP client implementation of JDK resends a request only once when an error occurs in the HTTP communication and an appropriate response is not received from the server. You can use the system properties of JDK to prevent the resending of requests.

(1) Specifying a property value

For preventing the resending of a request by the HTTP client implementation of JDK, set `sun.net.http.retryPost=false` in the system property. This is a standard system property supported by JDK 6 or later. For details on the system properties, see the *JDK documentation*.

(2) How to set a property

The method for setting a property is the same method as that for setting a system property for establishing a connection according to the SSL protocol. For details on how to set a property, see *10.11(2) How to set a property*.

10.21 Injection

This section describes the `javax.xml.ws.WebServiceRef` annotation and injection of service classes and ports, and the Web Services context injection.

10.21.1 Injecting service classes and ports

When you specify the `javax.xml.ws.WebServiceRef` annotation in the following fields or methods of the Web Services clients operating on the J2EE server, the J2EE server generates and injects a service class and port when generating a Web Services class instance. For the `javax.xml.ws.WebServiceRef` annotation, see *19.3 Support range of annotations*.

- A service class type field or port type field (excluding `static` and `final`)
- The `setter` method that uses a service class type field or port type field as an argument (excluding `static` and `final`)

Using the `javax.xml.ws.WebServiceRef` annotation for injection has the following advantages:

- Using the `javax.xml.ws.WebServiceRef` annotation for injection can help reduce the amount of coding required when developing an application, which in turn, facilitates the Web Services client creation process.
- You can improve the run-time performance of a Web Services client by generating Web Services instances when developing J2EE applications. You cannot achieve this improved performance when you do not use the port injection.

For generating the Web Services client instances, see *10.21.1(2) Generating a Web Services client instance*.

Notes

- You can specify the `javax.xml.ws.WebServiceRef` annotation only when implementing a Web Services client as a servlet or an EJB, and not when implementing a Web Services client as any other application. For example, you cannot inject a service class or a port in the command line application Web Services clients.
- Generate a service class or a port instance only once, that is, when starting a J2EE application. Generate only one instance for each service class or port in which the `javax.xml.ws.WebServiceRef` annotation is specified.
- Inject a service class or a port every time you generate an instance of a Web Services client.
- While implementing a Web Services client as an EJB by enabling the pooling in Stateless Session Bean, you can generate instances of multiple Web Services clients when starting a J2EE application. In such cases, generate one instance of a service class or a port in which the `javax.xml.ws.WebServiceRef` annotation is specified and inject this generated instance in every instance of the Web Services client. For generating Web Services client instances, see *10.21.1(2) Generating an instance of a Web Services client*.
- If the configuration is such that the Web Service A is invoked from the Web Service B, you cannot inject a service class or a port of the Web Service B to the Web Service A by specifying the `javax.xml.ws.WebServiceRef` annotation.
- You cannot specify the `javax.xml.ws.WebServiceRef` annotation in a class which references a Web Services client.
- You cannot use the reload functionality to replace a J2EE application that uses the `javax.xml.ws.WebServiceRef` annotation to inject a service class or port. To replace a J2EE application of this type, first stop and delete the application to be replaced and then import and start the new application. For details on the reload functionality, see *uCosminexus Application Server Common Container Functionality Guide*.

(1) Example of specifying the javax.xml.ws.WebServiceRef annotation

Examples of specifying the `javax.xml.ws.WebServiceRef` annotation are as follows:

- Specifying the annotation in a field

```
...
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.xml.ws.WebServiceRef;

import com.sample.AddNumbersImpl;
import com.sample.AddNumbersImplService;

public class TestClient extends HttpServlet {
    // An example of a service class
    // Inject a service class instance in a service class type field
    @WebServiceRef
    private AddNumbersImplService service;

    // An example of a port
    // Inject a port instance in a port type field
    @WebServiceRef(AddNumbersImplService.class)
    private AddNumbersImpl port;

    @Override
    public void init() {
        // You need not execute the following processes because the Application
server injects a service class and a port before
        // starting the Web Service client
        //service = new AddNumbersImplService();
        //port = service.getAddNumbersImplPort();
    }

    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException {
        ...
```

- Specifying the annotation in the `setter` method

```
...
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.xml.ws.WebServiceRef;

import com.sample.AddNumbersImpl;
import com.sample.AddNumbersImplService;

public class TestClient extends HttpServlet {
    private AddNumbersImplService service;
    private AddNumbersImpl port;

    @Override
    public void init() {
        // You need not execute the following process because the application
server injects a service class and a port by using the
        // setter method when starting the application
        //service = new AddNumbersImplService();
        //port = service.getAddNumbersImplPort();
    }

    // An example of a service class
    @WebServiceRef
    public void setAddNumbersImplService(AddNumbersImplService service) {
        // Inject the service class instances into the service argument
        this.service = service;
    }

    // An example of a port
    @WebServiceRef(AddNumbersImplService.class)
    public void setAddNumbersImpl(AddNumbersImpl port) {
        // Inject the port instances into the port argument
        this.port = port;
    }

    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException {
        ...
```

(2) Generating an instance of a Web Services client

When starting a J2EE application, configure the following settings to generate an instance of a Web Services client.

For a servlet

Specify the `load-on-startup` element in `web.xml` of the WAR file with a Web Services client. If you do not want to specify the `load-on-startup` element, generate the instances when running the first Web application. For details, see *uCosminexus Application Server Web Container Functionality Guide* .

For EJB

Set the pooling in Stateless Session Beans to enabled state. Generate the number of instances equivalent to the minimum pooling in Stateless Session Beans when starting a J2EE application. When not using the pooling in Stateless Session Beans, generate an instance when running the first J2EE application. For details, see *uCosminexus Application Server EJB Container Functionality Guide*.

Notes

In an environment where the Web Services client and Web Service to connect to, are both deployed to the same J2EE server, an error (KDJW40043-E) occurs while injecting ports or service classes when starting the J2EE server.

Troubleshooting if injection fails:

- In the `wsdlLocation` element of the `javax.xml.ws.WebServiceRef` annotation, specify the WSDL document stored locally by using the relative path or absolute path.
- Stop the J2EE application that has the Web Services client before you stop the J2EE server. Restart the J2EE server and then start the Web Services client.
- Deploy Web Service and the Web Services client on different J2EE servers. First, start the J2EE server on which Web Service is deployed and then start the J2EE server on which the Web Services client is deployed.
- Perform the settings such that you can reference the WSDL document that was locally stored when injecting a service class or a port, by using the catalog functionality.

(3) Using the handler framework

When using the handler framework in a service class or port injected by specifying the `javax.xml.ws.WebServiceRef` annotation, set the handler chain by using the APIs. For the handler chain settings, see *36.9.2 Setting the handler chain in the Web Services client*. We recommend that you configure the handler chain settings when initializing a Web Services client because you need to perform the settings for ports only once. You need not perform the handler chain settings for ports every time you invoke Web Service. You can use the following methods for initializing a Web Services client:

Implementing a Web Services client as a servlet

A method in which the `init` method or the `javax.annotation.PostConstruct` annotation is specified

Implementing a Web Services client as an EJB

A method in which the `javax.annotation.PostConstruct` annotation is specified

(4) Enabling features

By concurrently specifying the `javax.xml.ws.WebServiceRef` annotation and the annotation for features in the port type fields or in the `setter` method for port type fields, you can enable the features of the port to be injected. The features will be enabled only if you specify the `javax.xml.ws.WebServiceRef` annotation in the fields in which the annotation for features is specified or in the `setter` method for the fields. You, however, cannot specify the annotation for features in the service class type fields or in the setter method for the fields.

When enabling the features, you can specify the following annotations in the ports. For annotations, see *16.2 Customized mapping from Java to WSDL*.

- `javax.xml.ws.soap.Addressing`
- `javax.xml.ws.soap.MTOM`
- `com.sun.xml.ws.developer.StreamingAttachment`

You can use attachments in the MTOM/XOP specification format on the ports to be injected, if you enable the feature as follows:

- Example of specifying the annotation in a field

```
...
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.xml.ws.WebServiceRef;

import com.sample.AddNumbersImpl;
import com.sample.AddNumbersImplService;

public class TestClient extends HttpServlet {
    // Enable the MTOM/XOP specification format attachment in the port to be
injected
    @MTOM
    @WebServiceRef(AddNumberImplService.class)
    private AddNumbersImpl port;

    @Override
    public void init() {
        ...
    }

    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException {
        ...
```

- Example of specifying the annotation in the `setter` method

```
...
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.xml.ws.WebServiceRef;

import com.sample.AddNumbersImpl;
import com.sample.AddNumbersImplService;

public class TestClient extends HttpServlet {
    private AddNumbersImpl port;

    @Override
    public void init() {
        ...
    }

    // Enable the MTOM/XOP specification format attachment in the port to be
injected
    @MTOM
    @WebServiceRef(AddNumberImplService.class)
    public void setAddNumbersImpl(AddNumbersImpl port) {
        this.port = port;
    }

    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException {
        ...
```

(5) Changing the properties of a request context

We recommend that you change the request context properties of ports to be injected when initializing the Web Services client. You can use the following methods for initializing the Web Services client:

The method of implementing as a servlet

A method in which the `init` method or the `javax.annotation.PostConstruct` annotation is specified

The method of implementing as an EJB

A method in which the `javax.annotation.PostConstruct` annotation is specified

Notes

In the Web Services clients that share a single port across multiple threads, if you change the request context properties of the port while multiple threads are operating, communication errors might occur or invalid SOAP messages might be sent. Therefore, you must change the request context properties of the port that is shared across multiple threads before the shared threads start operating.

You can change the request context properties of a port in the Web Services client that is implemented as a servlet as follows:

```
...
import java.util.Map;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.xml.ws.BindingProvider;
import javax.xml.ws.WebServiceRef;

import com.sample.AddNumbersImpl;
import com.sample.AddNumbersImplService;

public class TestClient extends HttpServlet {

    @WebServiceRef(AddNumbersImplService.class)
    AddNumbersImpl port;

    @Override
    public void init() {
        // Set the request context during initialization
        Map<String, Object> context = ((BindingProvider)port).getRequestContext();
        context.put("com.cosminexus.jaxws.connect.timeout", 60000);
    }
    ...
}
```

10.21.2 Injecting a Web Services context

The `javax.xml.ws.WebServiceContext` interface is one of the service APIs explained in the *JAX-WS 2.2 specifications, section 5.3*. For using the Web services context, inject a resource by using the `javax.annotation.Resource` annotation.

Specify the `javax.annotation.Resource` annotation in the `javax.xml.ws.WebServiceContext` type field or in the `setter` method of Web Services Implementation Class or Provider Implementation Class. The `setter` method uses the `javax.xml.ws.WebServiceContext` type as an argument. By specifying the `javax.annotation.Resource` annotation in the above mentioned field or method, you can inject the information regarding the request being processed in the field that corresponds to the specified field or the specified `setter` method. You can access the message context information by acquiring the message context by using the `getMessageContext` method of the `javax.xml.ws.WebServiceContext` interface. However, you cannot acquire the message context by using the `getMessageContext` method when accessing EJB Web Services Implementation Class as an EJB You can optionally do the following to access the message context information:

- You can reference the user-defined message context properties of the APPLICATION scope added in the service-side handler when inbounding, from Web Services Implementation Class or Provider Implementation Class.
- You can reference the user-defined message context property added in Web Services Implementation Class or Provider Implementation Class, from the service-side handler.

For the `getMessageContext` method, see *19.2.3(2) javax.xml.ws.WebServiceContext interface*, and for the message contexts, see *19.2.5 Using a message context*.

(1) Specifying the javax.annotation.Resource annotation

When using the `javax.annotation.Resource` annotation to inject a Web Services context, specify the `javax.annotation.Resource` annotation in the following fields or methods of Web Services Implementation Class or Provider Implementation Class (including the parent Implementation Class). The operation is not guaranteed if you specify the `javax.annotation.Resource` annotation in the following fields or methods. Also, you can specify the annotation either in the following fields or in the `setter` method corresponding to these fields.

- `javax.xml.ws.WebServiceContext` type fields (excluding the `static` or `final` fields)
- The `setter` methods# that use the `javax.xml.ws.WebServiceContext` type fields as arguments, and that are not to be published (excluding the `static` or `final` fields)

  #

  A non-public `setter` method or the `javax.jws.WebMethod` annotation for which the `exclude` element is `true`.

Further, when using the `javax.annotation.Resource` annotation to inject a Web Services context, you cannot specify any elements in the `javax.annotation.Resource` annotation. If you specify any element, the operation is not guaranteed.

The specification example of the `javax.annotation.Resource` annotation is as follows:

- Specifying the annotation in a field

```
import javax.annotation.Resource;
import javax.jws.WebService;
import javax.servlet.ServletContext;
import javax.xml.ws.handler.MessageContext;
import javax.xml.ws.WebServiceContext;

@WebService
public class AddNumbersImpl {
    // Inject the information about the request that is currently being processed
in the wsContext field
    @Resource
    private WebServiceContext wsContext;

    public int add(int number1, int number2) throws AddNumbersFault {
        // Acquire the message context
        MessageContext mContext = wsContext.getMessageContext();
        // Acquire the properties
        ServletContext sContext =
(ServletContext)mContext.get(MessageContext.SERVLET_CONTEXT);
        ...
    }
    ...
}
```

- Specifying the annotation in the `setter` method

```
import javax.annotation.Resource;
import javax.jws.WebService;
import javax.xml.ws.handler.MessageContext;
import javax.xml.ws.WebServiceContext;

@WebService
public class AddNumbersImpl {
    // Inject the information about the request that is currently being processed
in the wsContext field corresponding to the setter method
    private WebServiceContext wsContext;

    @Resource
    private void setWebServiceContext(WebServiceContext wsContext) {
        this.wsContext = wsContext;
    }
    public int add(int number1, int number2) throws AddNumbersFault {
        // Acquire the message context
        MessageContext mContext = wsContext.getMessageContext();
        // Set the properties
        mContext.put("userPropKey", "userPropValue");
        ...
    }
    ...
}
```

(2) Notes when adding a user-defined message context property

This subsection describes the notes when adding a user-defined message context property:

(a) Adding a property in the service-side handler when inbounding

You can add the user-defined message context properties by using the service-side handler for inbounding. When referencing these properties from Web Services Implementation Class or Provider Implementation Class, you must first set the user- defined message context properties in the service-side handler and then use the `setScope(java.lang.String name, MessageContext.Scope scope)` method, which is an API of the `javax.xml.ws.handler.MessageContext` interface, for setting the user-defined properties as the APPLICATION scope.

For referencing the user-defined message context properties (added in the service-side handler) from the service-side handler when outbounding, set only the user-defined message context properties in the service-side handler. You need not set the scope by using the `setScope(java.lang.String name, MessageContext.Scope scope)` method. For the `setScope(java.lang.String name, MessageContext.Scope scope)` method, see *19.2.4(8) javax.xml.ws.handler.MessageContext interface*.

The following example shows how to add the user-defined properties in the service-side handler when inbounding:

```
import javax.xml.ws.handler.MessageContext;
import javax.xml.ws.handler.soap.SOAPHandler;
import javax.xml.ws.handler.soap.SOAPMessageContext;

public class ServiceSOAPHandlerImpl implements SOAPHandler<SOAPMessageContext> {

    public boolean handleMessage(SOAPMessageContext smContext) {
        // Acquire the message direction
        boolean outbound =
(boolean)smContext.get(MessageContext.MESSAGE_OUTBOUND_PROPERTY);
        if(outbound) {
           // Outbounding processes
           ...
        } else {
           // Inbounding processes

           // Specify the property key in ("userPropKey")
           // and a value in ("userPropValue")
           smContext.put("userPropKey", "userPropValue");
           // Set the property scope as the APPLICATION scope
           smContext.setScope("userPropKey", MessageContext.Scope.APPLICATION);
           ...
        }
        ...
    }
    ...
}
```

(b) Adding a property in Web Services Implementation Class or Provider Implementation Class

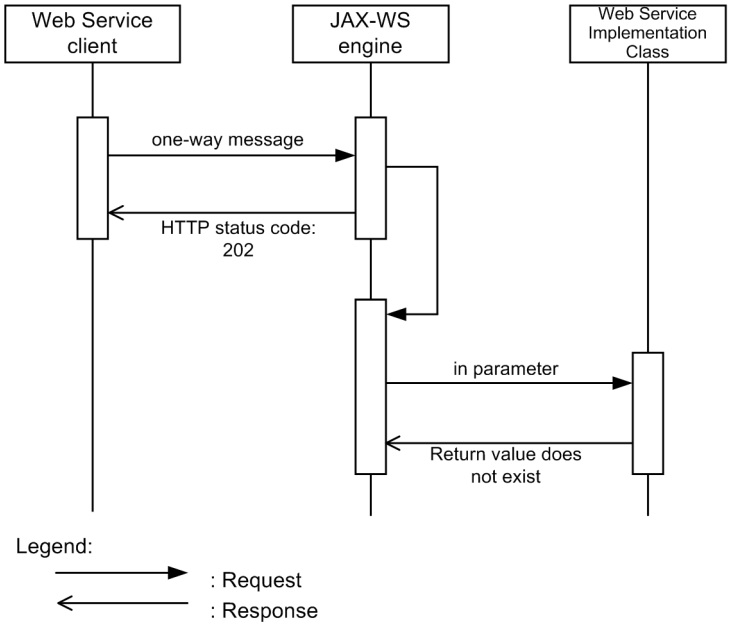
For referencing the user-defined message context properties that are added in Web Services Implementation Class or Provider Implementation Class from the service-side handler when outbounding, set only the user-defined message context properties in Web Services Implementation Class or Provider Implementation Class. Do not set the scope by using the `setScope(java.lang.String name, MessageContext.Scope scope)` method. The operation is not guaranteed if you use the `setScope(java.lang.String name, MessageContext.Scope scope)` method.

10.22 One-way operations

A one-way operation is a communication in one direction and thus has no corresponding SOAP message responses. But as the HTTP is used in the transfer protocol of the one-way messages, the Web Services client waits for an HTTP response, the HTTP body of which does not have a SOAP message. On the other hand, for the one-way operations for Web Services Implementation Class, the service-side JAX-WS engine returns the HTTP response as an HTTP status code `202 Accepted` before invoking a method belonging to Web Services Implementation Class.

The following figure shows the sequence of a one-way operation

Figure 10-9: Sequence of a one-way operation



10.22.1 Notes on one-way operations

- For Provider Implementation Class, if `null` is returned, the HTTP response as an HTTP status code `202 Accepted` is returned after the method invocation is complete.
- The one-way operations are not supported when you use the WSS (Web Services-Security), the WS-RM 1.2 functionality, or addressing functionality. The operation is guaranteed only when you individually use the WSS (Web Services-Security), the WS-RM 1.2 functionality, or addressing functionality.
- The one-way operations have no corresponding SOAP messages. Therefore, you must take care that the exceptions are not thrown explicitly when implementing a method of the one-way operation of Web Services Implementation Class.
- If the service-side JAX-WS engine receives an invalid SOAP message and the unmarshalling fails, the engine sends the SOAP fault, the HTTP status code of which is set as `500 Internal Server Error`, to the Web Services client.
- If the Web Services client that uses a one-way operation receives an HTTP status code other than `200 OK` or `202 Accepted`, the Web Services client throws the `javax.xml.ws.WebServiceException` exception.

10.23 A functionality to dynamically generate wrapper bean

With the functionality to dynamically generate the wrapper bean, the JAX-WS engine dynamically generates the wrapper bean (the request bean and response bean), and the `Javabeans` classes of the fault bean. You can use the functionality in Web Services that are developed starting from SEI. However, the Web Services clients or Web Services, for which the development started from the WSDL, do not support the functionality to dynamically generate the wrapper bean.

This section describes the performance and error checks of the functionality to dynamically generate the wrapper bean.

(1) Using the cjwsgen command for checking errors

When developing Web Services starting from SEI, compile Web Services Implementation Class by using the `javac` command. However, when compiling Web Services Implementation Class by using the `javac` command, the following mapping errors are not checked:

- Default mapping
  For details, see *1.4.2(1)(a) Default mapping*.
- Customization of mapping (annotation)
  For details, see *1.4.2(1)(b) Customization of mapping*.

For this reason, if definitions unsupported in the JAX-WS functionality are included in Web Services Implementation Class, the process of starting Web Service or acquiring the metadata after Web Service starts successfully, might lead to errors. To avoid errors, you must execute the `cjwsgen` command in Web Services Implementation Classes compiled by using the `javac` command and check for the errors in advance. If Web Service fails to start, the following error messages are output to the log or the standard error output.

- KDJW00003-E
- KDJW00005-E
- KDJW00006-E
- KDJW00007-E
- KDJW00008-E
- KDJW20026-E
- KDJW20027-E
- KDJW20041-E
- KDJW20042-E
- KDJW40011-E
- KDJW40013-E

(2) Performance

The JAX-WS engine dynamically generating a `JavaBeans` class is an on-memory processing. As compared to the process in which a `JavaBeans` class is statically generated, the starting performance of Web Service, therefore, declines when the JAX-WS engine puts a load on the Web Service when dynamically generating a `JavaBeans` class. However, the process of dynamically generating a `JavaBeans` class does not require the process of reading the WAR file, thereby reducing the load.

Because the `JavaBeans` class that is statically generated by JAX-WS is the same as the `JavaBeans` class that is statically generated by the `cjwsgen` command, the performance of the SOAP communication after you start Web Service is the same irrespective of whether the `JavaBeans` class is generated dynamically or statically.

11 Points on developing RESTful Web Services

This chapter describes the points and precautions that you must note in advance for each operation of developing RESTful Web Services (Web resources).

11.1 Creating a root resource class

You create a root resource class as a compiled Java class file (*.class). Similarly create a sub resource class or an exception mapping provider as and when required.

You include the compiled Java class file (*.class) in the directories that configure the WAR file. Add the file to either or both of the following locations:

- Under the classes directory
- In the JAR files included in the lib directory

A WAR file must contain at least one root resource class. For details on the storage destination, see *11.3.1 Configuring a WAR file*.

11.2 Creating web.xml

This section describes the `web.xml` file included in a WAR file that is used in a Web resource.

You name the file as `web.xml` and store it under the `WEB-INF` directory that configures the WAR files. Storing the `web.xml` file is mandatory.

The following sections describe both the definitions required for executing a Web Service and an example of a `web.xml` file.

(1) Definitions required for executing a Web Service

Create the `web.xml` file so that the file satisfies the following conditions:

- **Version**
  The version of `web.xml` must be 2.5 or later.
- **Servlet**
  You must include the following `servlet` elements in the `web-app` element:

```
<servlet>
 <servlet-name>CosminexusJaxrsServlet</servlet-name>
  <servlet-class>
   com.cosminexus.jersey.spi.container.servlet.ServletContainer
  </servlet-class>
</servlet>
```

- **Servlet Initialization Parameter**
  You include the following `init-param` element in the `servlet` element as and when required. Note that the `init-param` element is case-sensitive.

```
<init-param>
 <param-name>
  com.sun.jersey.config.property.packages
 </param-name>
 <param-value>
  A list of package names separated with semicolon, comma, or space
 </param-value>
</init-param>
```

  When you include the `init-param` element in the `servlet` element, you must publish the Web resources included in the package and sub-package specified in the `init-param` element. When you do not include the `init-param` element in the `servlet` element, publish all the Web resources included in the WAR file.

  ! Important note

  - You cannot use an * (asterisk) in the *param-value* of `com.sun.jersey.config.property.packages`.
  - The error (KDJJ10020-E) occurs if you use an asterisk or specify an incorrect package name. `500` is returned as the HTTP error code.

- **Servlet Mapping**
  Code the `servlet-mapping` element below the `web-app` element.
  The following is an example of how to code a `servlet-mapping` element.

```
<servlet-mapping>
 <servlet-name>CosminexusJaxrsServlet</servlet-name>
 <url-pattern>/*</url-pattern>
</servlet-mapping>
```

(2) Example of web.xml

An example of `web.xml` is as follows:

```
<web-app version="3.0"
  xmlns="http://java.sun.com/xml/ns/javaee"
```

```
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/
javaee/web-app_3_0.xsd">
  <servlet>
   <servlet-name>CosminexusJaxrsServlet</servlet-name>
   <servlet-class>
    com.cosminexus.jersey.spi.container.servlet.ServletContainer
   </servlet-class>
   <init-param>
    <param-name>com.sun.jersey.config.property.packages</param-name>
    <param-value>org.test.resources1;org.test.resources2</param-value>
   </init-param>
  </servlet>
  <servlet-mapping>
   <servlet-name>CosminexusJaxrsServlet</servlet-name>
   <url-pattern>/*</url-pattern>
  </servlet-mapping>
 </web-app>
```

When creating `web.xml` of version 2.5, specify `2.5` in the `version` attribute of the `web-app` element and specify `http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd` as the second location information in the `xsd:schemaLocation` attribute.

In this example, the following Web resources will be published assuming that org.test.resources1.ResourceA, org.test.resources1.ResourceB, org.test.resources2.ResourceC, and org.test.resources3.ResourceD are included in the WAR file:

- ResourceA
- ResourceB
- ResourceC

11.3 Creating an archive

This section describes how to configure a WAR file and create an EAR file.

11.3.1 Configuring a WAR file

You must configure the WAR file to be used in the Web resources as described in the following table.

Table 11-1: Configuring a WAR file

Directory	Remarks
/	--
META-INF/	--
MANIFEST.MF	--
WEB-INF/	--
web.xml	Indicates the created `web.xml` file.
classes/	Stores the compiled Java classes.
lib/	Stores the JAR files that include the compiled Java classes.

Legend:

--: Indicates that no explanation or supplementary information is available.

11.3.2 Creating an EAR file

For deploying Web resources to the J2EE server, create an EAR file that includes the already created WAR file. You need to use `application.xml` to create an EAR file.

For details on the configuration of an EAR file, see *1.4.2 J2EE applications in an archive format* in the *uCosminexus Application Server Application Development Guide*.

11.4 Implementing a client by using a client API for RESTful Web Services

This section describes how to implement a client by using a client API for RESTful Web Services (hereafter, referred to as a Web resource client). When implementing a client by using the standard Java APIs such as `java.net.URL` and `java.net.HttpURLConnection`, see the *JDK documentation*.

The Web resource clients have no specific restrictions. This allows you to develop, for example, the following Web services:

- Java applications (calling Web resources from Java applications)
- JSPs (calling Web resources from JSPs)
- Servlets (calling Web resources from servlets)
- EJBs (calling Web resources from EJBs)
- SOAP Web Services (calling Web resources from SOAP Web Services)
- Web resources (calling an additional Web resource from a root resource class and sub resource class)

A Web resource client differs from a client that calls SOAP Web Services in which you do not need to create Java sources such as a stub in advance by executing a command before implementing a Web resource client. Implement a Web resource client according to the specifications of a client API for RESTful Web Services. For details on the specifications of a client API for RESTful Web Services, see *25. Client API Support Environment for RESTful Web Services*.

11.4.1 Use case of a Web resource client

This subsection describes a basic use case of a client API for RESTful Web Services. You can use any of the following methods to call a Web resource:

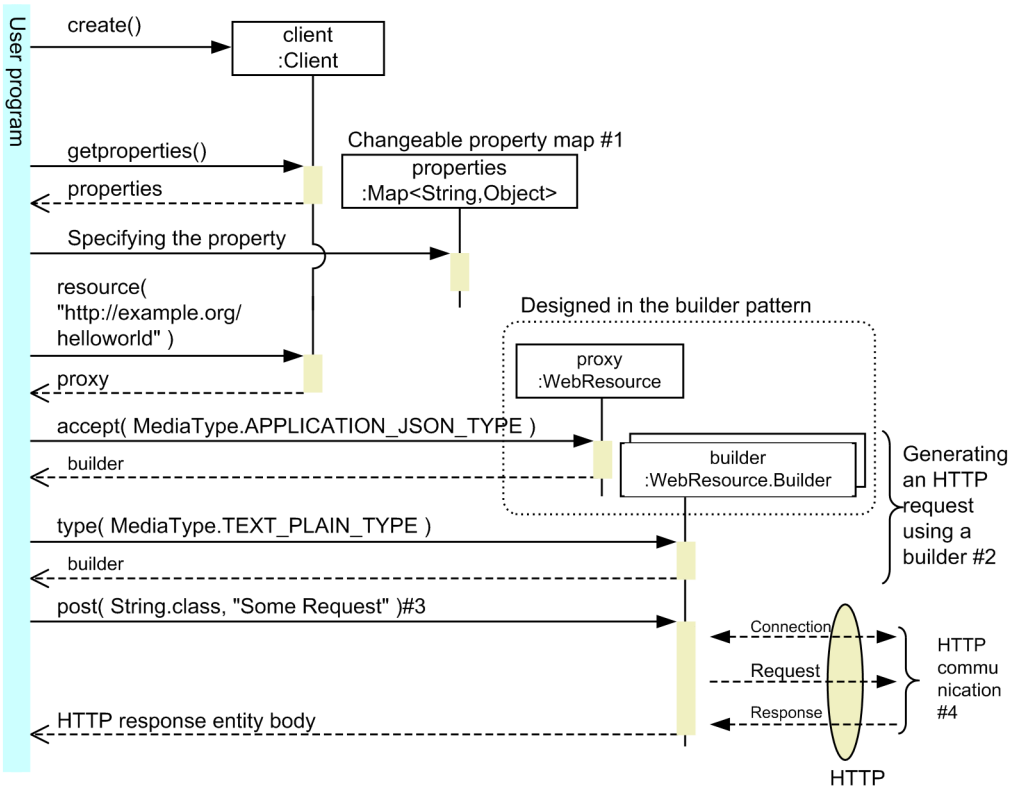
- Sending and receiving HTTP requests and HTTP responses by specifying a Java type
- Sending HTTP requests by specifying a Java type and receiving HTTP responses in a generic type (`ClientResponse`)
- Sending and receiving HTTP requests and HTTP responses in generic types (`ClientRequest` and `ClientResponse`).

The description of each method is as follows:

(1) Sending and receiving HTTP requests and HTTP responses by specifying a Java type

The following figure shows a use case describing the sending and receiving of HTTP requests and HTTP responses by specifying a Java type.

Figure 11-1: Use case describing the sending and receiving of HTTP requests and HTTP responses by specifying a Java type



#1

You can reference or change the properties included in a map by various methods. For details, see *11.4.3 Setting the properties and features*.

#2

You can generate a request by using various methods of a client API. For details, see *11.4.3 Setting the properties and features*.

#3

Sends an HTTP request entity body in the character string format by using the `HTTP POST` method and similarly, receives an HTTP response entity body in the character string format.

The client API also contains the methods corresponding to each HTTP method such as `DELETE`, `GET`, `HEAD`, `OPTIONS`, and `PUT`. For details, see *11.4.3 Setting the properties and features*.

#4

`HttpURLConnection` establishes the HTTP communication. The JAX-RS engine establishes the communication until the HTTP message is created and then delegates the communication to `HttpURLConnection` of the Java SE.

The coding example corresponding to figure 11-1 is as follows:

```
Client client = Client.create();
Map<String, Object> properties = client.getProperties();
properties.put(ClientConfig.PROPERTY_READ_TIMEOUT, 10000);
WebResource proxy = client.resource( "http://example.org/helloworld" );
WebResource.Builder builder = proxy.accept( MediaType.APPLICATION_JSON_TYPE );
builder = builder.type( MediaType.TEXT_PLAIN_TYPE );
String response = builder.post( String.class, "Some Request" );
```

The procedure in the example is as follows:

1. Create a `client` object by using the `create() static` method of the `Client` class.
2. Set a property in the changeable property map acquired with the `getProperties()` method of the `Client` class. You can also set the properties by using different methods. For details on properties and features, see *11.4.3 Setting the properties and features*.
3. Create a `WebResource` object by calling the `resource()` method of the `Client` class and generate an HTTP request by calling a method of the `WebResource` object. The `WebResource` class is designed by using the builder pattern and contains various methods to generate an HTTP request.
4. Establish the HTTP communication by calling the `post()` method of the `WebResource.Builder` class. The `WebResource` class also contains the methods corresponding to each HTTP method, such as `DELETE`, `GET`, `HEAD`, `OPTIONS`, and `PUT`.

(2) Sending HTTP requests by specifying a Java type and receiving HTTP responses in a generic type (ClientResponse)

You can also receive an HTTP response in a generic type such as the `ClientResponse` object. Acquiring an HTTP response with the `ClientResponse` object enables a user to obtain various types of information of the received HTTP response (HTTP header, entity body, and status code).

You can acquire an entity body by using the `getEntity()` method, thereby specifying the Java type. For details on the methods supported by the `ClientResponse` class, see *25.1 The support range of client API interfaces and classes*.

The method to acquire an HTTP response to be received with `ClientResponse` object is as follows:

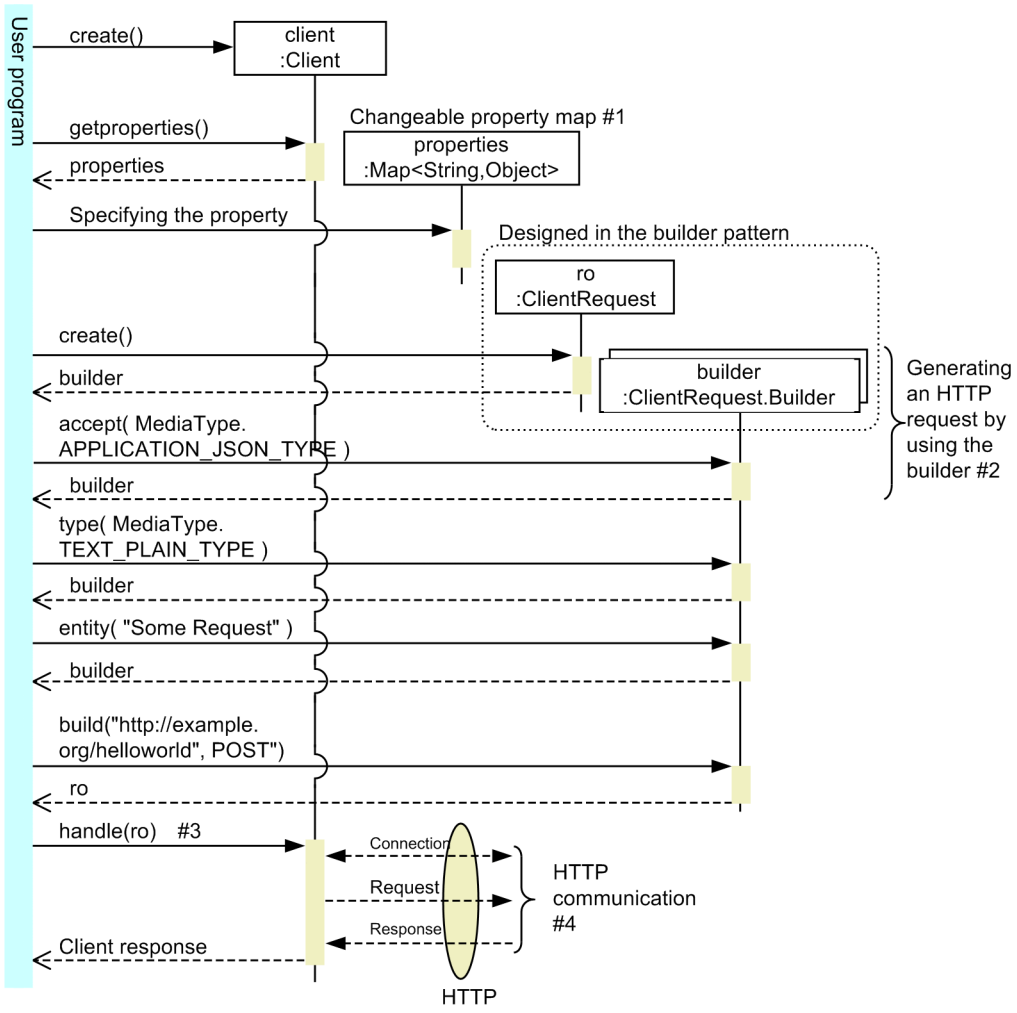
To create an HTTP request by using the `WebResource` object, specify `ClientResponse.class` in the parameter that specifies the Java type of the HTTP response. Specify `ClientResponse.class` instead of `String.class` in the coding example described in *11.4.1(1) Sending and receiving HTTP requests and HTTP responses by specifying a Java type.*.

The HTTP response is always `ClientResponse` if you directly send an HTTP request using the `Client` class. For details on the `Client` class, see *11.4.1(3) Sending and receiving HTTP requests and HTTP responses in a generic type*.

(3) Sending and receiving HTTP requests and HTTP responses in generic types

The following figure shows a use case describing sending and receiving HTTP requests and HTTP responses (`ClientRequest` and `ClientResponse`) in generic types.

Figure 11-2: Use case describing sending and receiving HTTP requests and HTTP responses in generic types.



#1

You can reference or change the properties included in a map with various methods. For details, see *11.4.3 Setting the properties and features*.

#2

You can create a request by using various methods of a client API. For details, see *11.4.3 Setting the properties and features*.

#3

You can send an HTTP request entity body in the character string format by using the `HTTP POST` method and similarly, receive an entity body of an HTTP response in the character string format.

A client API contains the methods corresponding to each HTTP method, such as `DELETE`, `GET`, `HEAD`, `OPTIONS`, and `PUT`. For details, see *11.4.3 Setting properties and features*.

#4

`HttpURLConnection` establishes a connection. The JAX-RS engine establishes the communication until the HTTP message is created and then delegates the communication to `HttpURLConnection` of the Java SE.

The coding example corresponding to figure11-2 is as follows.

```
Client client = Client.create();
Map<String, Object> properties = client.getProperties();
properties.put(ClientConfig.PROPERTY_READ_TIMEOUT, 10000);
ClientRequest ro;
```

```
ClientRequest.Builder builder = ClientRequest.create();
builder.accept( MediaType.APPLICATION_JSON_TYPE );
builder.type( MediaType.TEXT_PLAIN_TYPE );
builder.entity("Some Request");
ro = builder.build(new URI("http://example.org/helloworld"), "POST");
ClientResponse clientResponse = client.handle(ro);
//The actual response in the form of String
String response = clientResponse.getEntity(String.class);
```

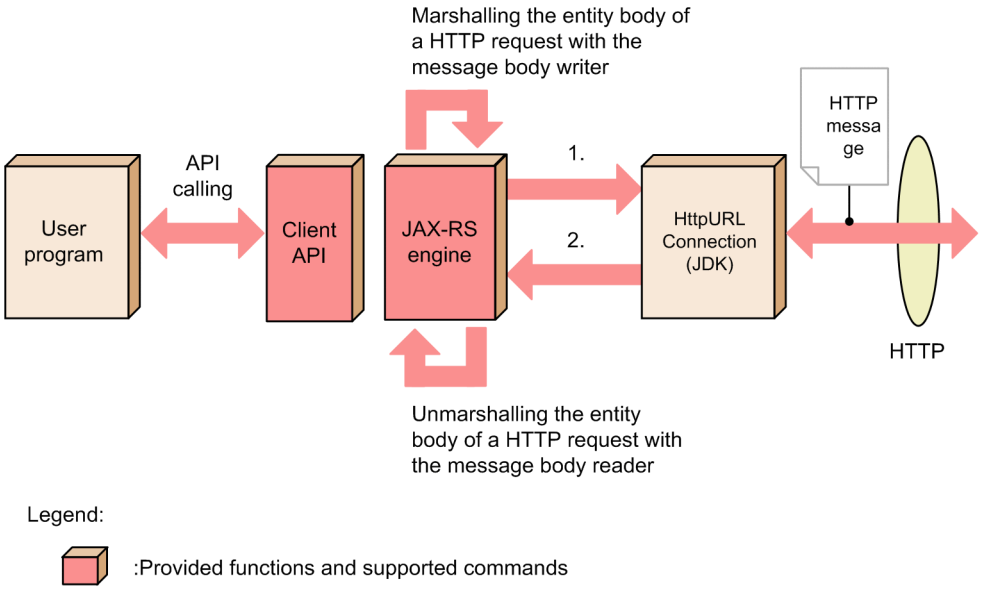
The procedure in the example is as follows:

1. Create a `client` object by using the `create() static` method of the `Client` class.
2. Set a property in the changeable property map acquired with the `getProperties()` method of the `Client` class. For details on the properties and features, see *11.4.3 Setting the properties and features*.
3. Create a `ClientRequest.Builder` object by calling the `create()` method of the `ClientRequest` class and generate an HTTP request by calling a method of the `ClientRequest.Builder` object. The `ClientRequest.Builder` class is designed in the builder pattern and contains various methods to generate HTTP requests.
4. Call a `build()` method of the `ClientRequest.Builder` class and create a `ClientRequest` object. Then, establish the HTTP communication by using the `handle()` method of the `Client` class.

11.4.2 Mechanism of a client API for RESTful Web Services

A client API for RESTful Web Services wraps the `HttpURLConnection/HttpsURLConnection` class and the JDK executes the actual HTTP communication. Hereafter, unless clearly specified, `HttpURLConnection/HttpsURLConnection` is collectively called `HttpURLConnection`.

The following figure shows the mechanism.



The following processes are executed in point 1. and point 2. in the figure:

1.

- The properties such as the connection timeout are set.
- An HTTP header is added.
- An entity body of the HTTP request is sent through the output stream acquired from `HttpURLConnection`.

2.

- An HTTP header is acquired.
- An entity body of the HTTP response is sent through the input stream acquired from `HttpURLConnection`.

(1) Relation between a user program and a client API for RESTful Web Services

A client API for RESTful Web Services mutually marshals and unmarshals a Java object and an HTTP request by using the entity body reader and entity body writer. A user program, therefore, need not directly use an input stream and output stream.

(2) Relation between JDK and client API for RESTful Web Services

The client APIs for RESTful Web Services delegate the processing of a transport layer to `HttpURLConnection`. The various built-in methods necessary for setting properties and HTTP headers are available; however, set the same values specified in the built-in methods to `HttpURLConnection`. For this reason, the JDK executes the processing that uses values specified in the actual HTTP communication and built-in methods.

11.4.3 Setting properties and features

The `Client` object contains a changeable property map that stores properties and features.

Hereafter, the term *property* is used collectively for both, property and features.

(1) Initializing the property map

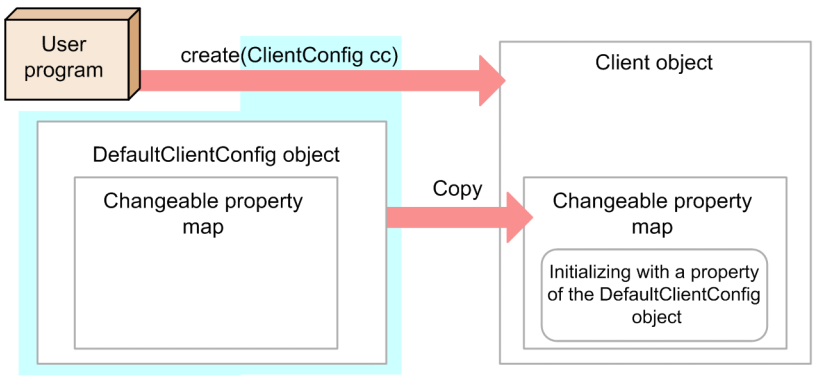
A changeable property map is initialized when creating a `Client` object. You can use any of the following methods to initialize a property:

Method 1

Create a `Client` object by specifying the `DefaultClientConfig` object in the parameter and then call the `create (ClientConfig cc) static` method of the `Client` class. Here, the changeable property map of the `Client` object is initialized# by the property contained in the `DefaultClientConfig` object.

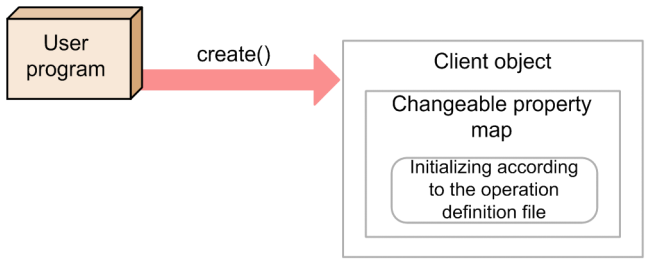
#

If the `DefaultClientConfig` object contains a property, the changeable property map is initialized with the value of that property. If the `DefaultClientConfig` object is without a property, the changeable property map is initialized with the settings mentioned in the action definition file.



Method 2

Create a `Client` object by using the `create() static` method of the `Client` class. Here, a changeable property map of the `client` object is initialized by the action definition file.



For details on an action definition file, see *13.1 Action definition file*.

(2) Structure of a changeable property map

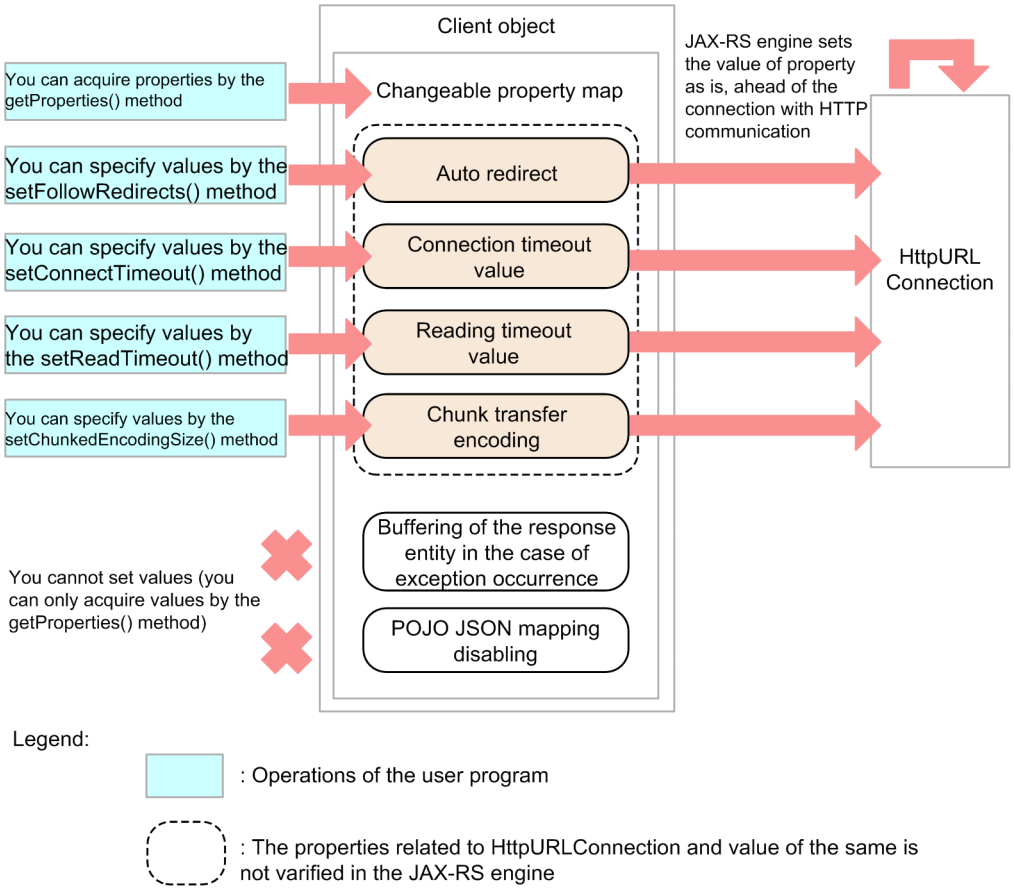
With the four properties related to the `HttpURLConnection` class, you can change the property map acquired with the `getProperties()` method of the `Client` class, and set the property map by using the `setter` method corresponding to each property.

Only the four properties related to HttpURLConnection have corresponding `setter` methods. When setting a property, change the property map acquired with the `getProperties()` method of the `Client` class.

The following figure shows the structure of a changeable property map.

For the list of supported properties, see *25.1.1 Supported properties and features*.

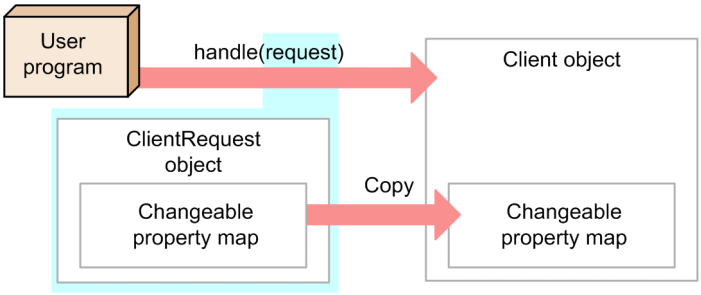
Figure 11–3: Structure of a changeable property map



Prior to the HTTP communication, the JAX-RS engine directly copies the values of the four properties related to the `HttpURLConnection` class, shown in the figure, to `HttpURLConnection` (however, does not verify the values). If the values are invalid, the `HttpURLConnection` class might throw an exception before or during the HTTP communication. When the values specified are `null`, those are not copied and are ignored.

The `ClientRequest` class also contains a changeable property map of the same format. For this reason, when establishing an HTTP communication by using the `handle()` method of the `Client` class, a user program can also set the property through the `ClientRequest` object. The following figure shows the mechanism.

Figure 11-4: Mechanism of copying the changeable property map



The JAX-RS engine copies the property included in the property map of the `ClientRequest` object to the property map of the `client` object only when the property map of the `Client` object does not contain the same property.

11.4.4 Setting an HTTP header

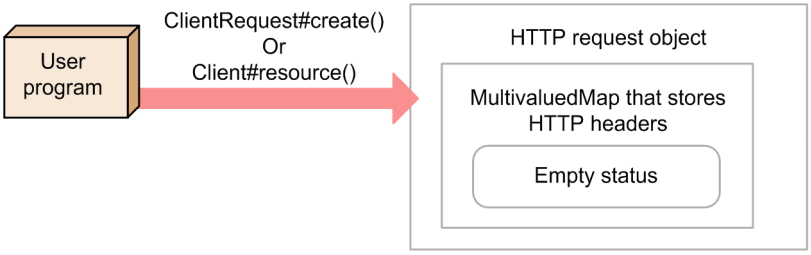
A client API provides various methods to set an HTTP header in an HTTP request object. For details on supported methods, see *25.1 The support range of client API interfaces and classes*. This subsection describes one of the following HTTP request objects:

- `WebResource` object or `ClientRequest` object
- `WebResource.Builder` object or `ClientRequest.Builder` object

An HTTP request object contains the `MultivaluedMap` object that stores a value corresponding to the HTTP header.

The following figure shows the initialization of the `MultivaluedMap` object that stores the HTTP header.

Figure 11-5: Mechanism of initializing the MultivaluedMap object

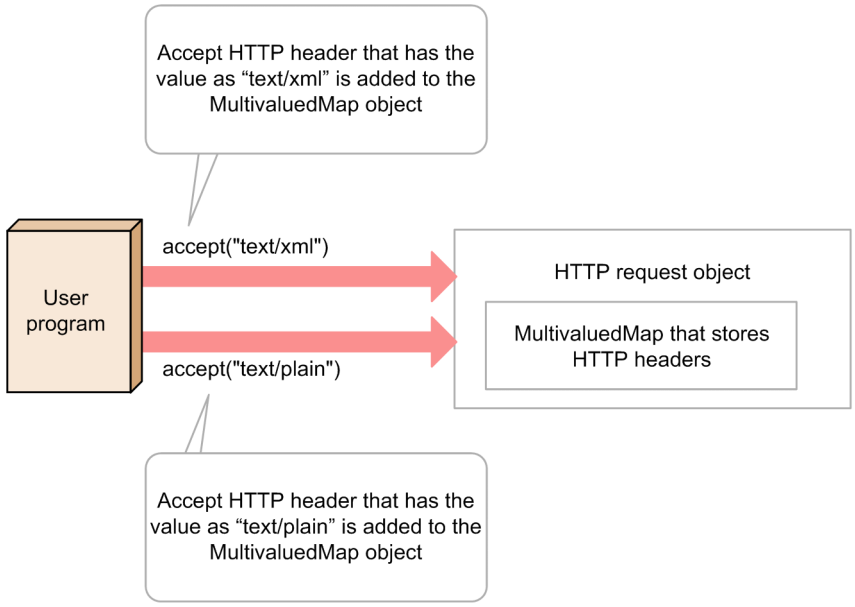


In the figure, a user program creates an HTTP request object by using the `resource()` method of the `Client` class or `create()` `static` method of the `ClientRequest` class.

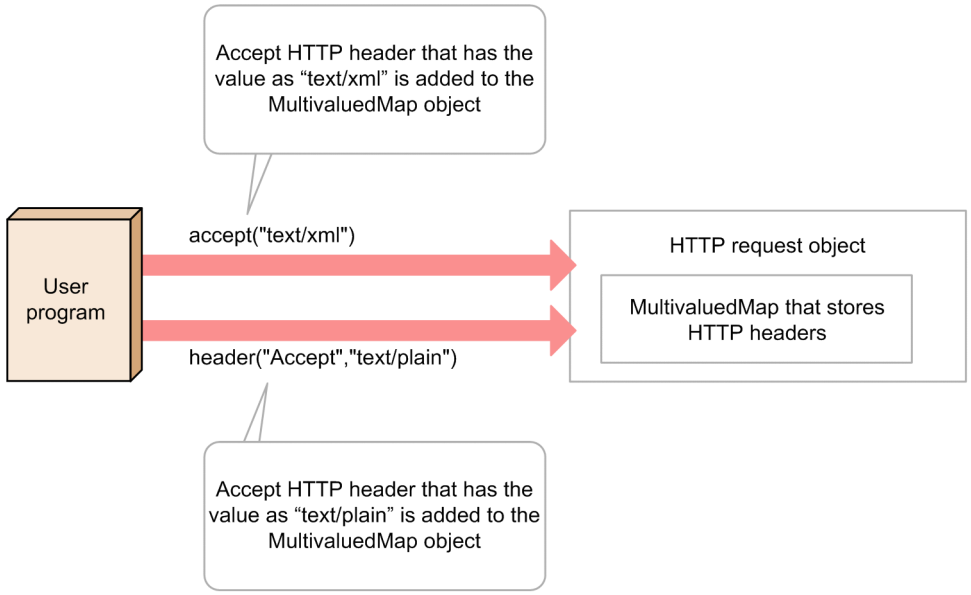
Here, an empty `MultivaluedMap` object is initialized.

The following figures show and describe the examples of setting an HTTP header in the `MultivaluedMap` object.

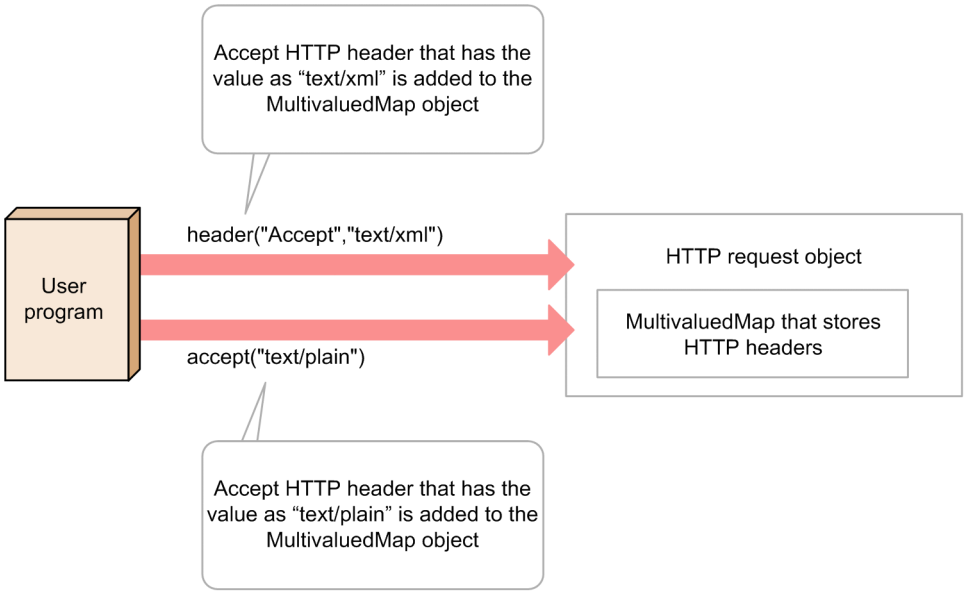
Example 1



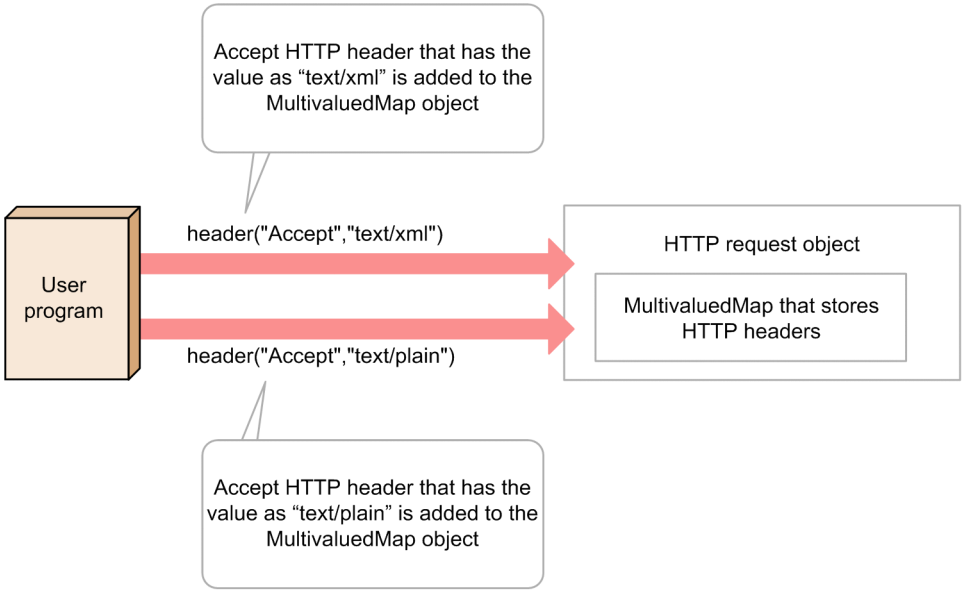
Example 2



Example 3



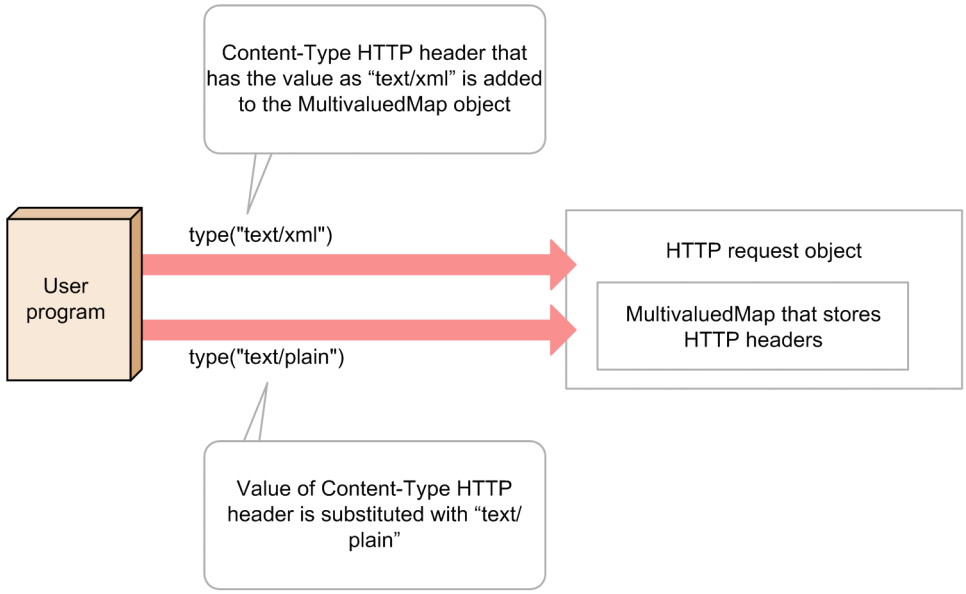
Example 4



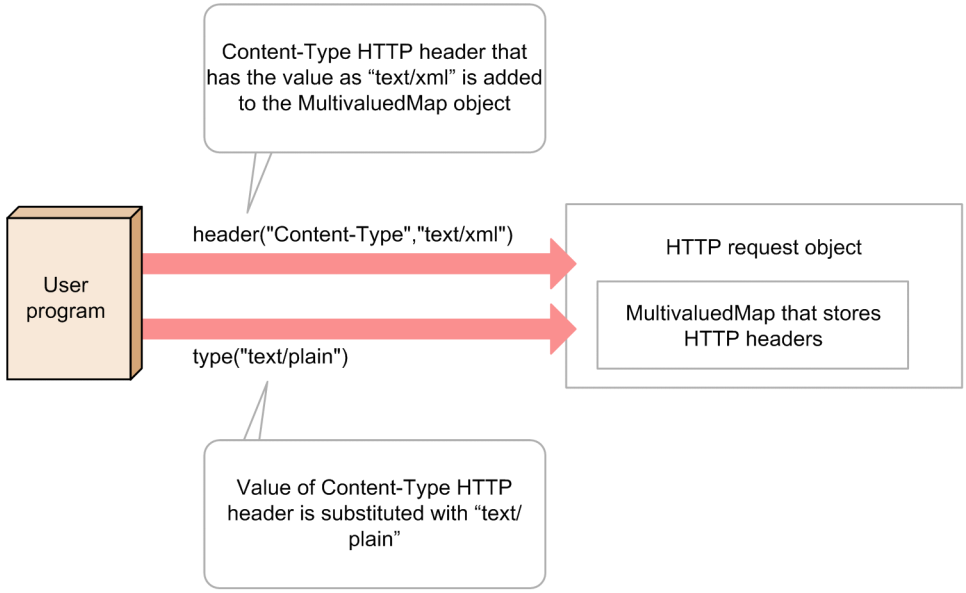
In all the methods shown in example 1 through example 4, a user program can create an HTTP request object that contains the HTTP Accept headers having the values `text/xml` and `text/plain`. This also applies to the HTTP Accept-Language header and HTTP Cookie header. For HTTP Accept-Language headers and HTTP Cookie headers, substitute the `accept` method with the `acceptLanguage()` method and `cookie()` method respectively.

The following figures show the operation in example 1 and example 3 for the HTTP Content-Type headers.

Example 1

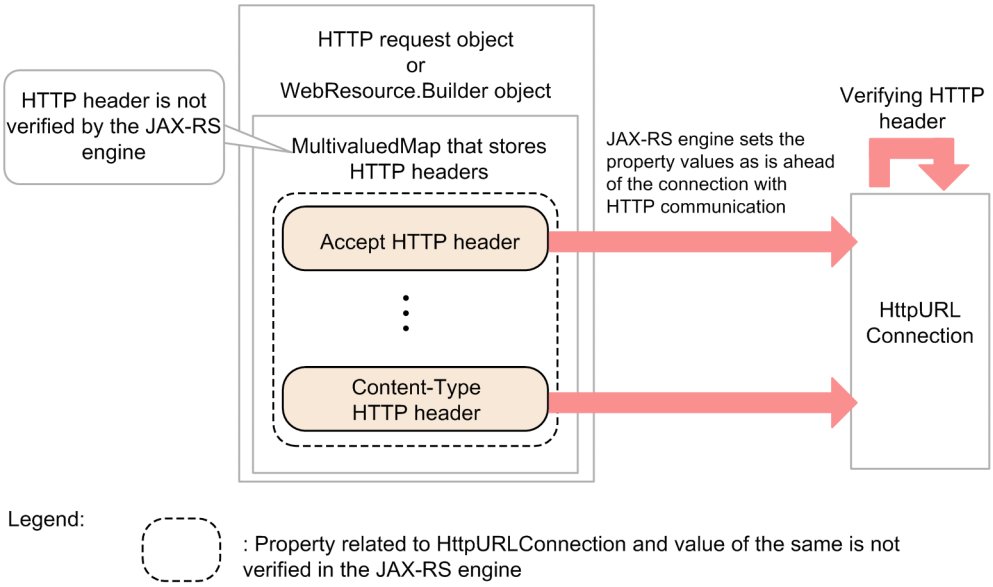


Example 3



The value of the Content-Type HTTP header set in the `HTTP request` object in both the examples above is `text/plain`. Also, because the HTTP Content-Type header can contain only one value, you cannot use the methods described in example 2 and example 4.

Prior to the HTTP communication, all the HTTP headers and respective values added in the HTTP request object are set in the `HttpURLConnection` object. The following figure shows the mechanism:



The corresponding HTTP header value included in the HTTP request object is set in the respective HTTP header. Accordingly, a non-null value of the corresponding HTTP header included in the HTTP request object is set to the value returned by the `toString()` method. The JAX-RS engine does not verify the value of an HTTP header. Set a value of an HTTP header in the HTTP request object according to the standard specifications.

Further, until establishing the HTTP communication, the operation when no specific HTTP header is set in the HTTP request object is same as the operation when no HTTP header is set in the `HttpURLConnection` object.

11.4.5 Notes

This subsection describes the *notes when implementing a Web resource client*

(1) Reusing the objects

We recommend that you reuse the already created `Client` object because creating an object incurs process cost. When creating a WebResource object more than once, or calling a Web resource more than once by using a method of the `Client` class, you need not create more than one `Client` objects.

Similarly, because creating WebResource also incurs cost, we recommend that you reuse the already created `WebResource` object. When creating more than one builders and HTTP requests for the same Web resource (URL), you need not create more than one `WebResource` object.

The methods used to set a `Client` class or methods of a `Client` class that destroy objects, however, are not thread safe. To know whether the methods are thread safe, check the following:

- When sharing a `Client` object across multiple threads, configure the settings for the `Client` object to be shared before multiple threads start operating.
- Destroy the `Client` object after the operation of multiple threads is complete.

If you execute these operations during the operation of multiple threads, the communication might fail and an invalid HTTP request might be sent.

When implementing a Web resource client with servlets or EJBs, create a `Client` object by using the initialization methods of servlets and EJBs and configure the required settings. Similarly, destroy the `Client` objects by using the destroy methods.

The following example shows a client API for RESTful Web Services used in servlets:

```
@WebServlet("/example")
public class ClientServlet extends HttpServlet {

  // A client object to be shared
```

```
   private Client client = null;

   // A WebResource object to be shared
   private WebResource proxy = null;

   @PostConstruct
   public void postConstruct() {
     // Create a client object to use
     // a client API for RESTful Web Services
     this.client = Client.create();
     // Setting the client: Obtain the property bag from the Client object
     Map<String, Object> properties = this.client.getProperties();
     // Setting the client: Set the read timeout
     properties.put(ClientConfig.PROPERTY_READ_TIMEOUT, 10000);
     //  Create a WebResource object from the Client object
     this.proxy = this.client.resource( "http://..." );
   }

  @PreDestroy
   public void preDestory() {
     //  destroy the Client object
     if( this.client != null ){
       this.client.destroy();
       this.client = null;
     }
   }

   @Override
   public void doGet( HttpServletRequest request, HttpServletResponse response )
     throws ServletException, IOException{
     ...
     ClientResponse clientResponse = null;
     // Invoke resource
     try {
       // Set a cookie and obtain an HTTP response from the Web resource
       Cookie cookie = new Cookie("cookie", "cookie%20value");
       clientResponse = this.proxy.cookie(cookie).get(ClientResponse.class);
     }catch (Exception e) {
       printStackTrace( e, out );
     }
     ...
   }
}
```

For details on the thread safety of a client API for RESTful Web Services, see *25.16 Thread safety of a client API for RESTful Web Services*.

(2) Settings for proxy SSL connection and basic verification

See the following respective sections for the proxy, SSL connection and basic verification settings.

- Settings for proxy
  10.10 Connecting through a proxy server
- Settings for SSL connection
  10.11 Connection by SSL protocol
- Settings for basic verification
  10.12 Connection by basic authentication

(3) Notes for using the Windows environment

The following exception might be logged if you use an environment in which you can send large number of requests from the Web resource client.

```
java.net.BindException: Address already in use: connect  [errno=10048, syscall=select]
```

This exception is thrown when a large number of requests are received for a Web resource client implemented as a servlet.

In such cases, take either or both of the following measures:

- Expand the range of port numbers available on the OS

Example: Changing the settings of MaxUserPort of the registry.

- Decrease the duration of `TIME_WAIT`

  Example: Changing the settings of `TcpTimedWaitDelay` of the registry.

However, see the documentation of your OS for details, as the specifications differ depending on the version, edition, and security update program installed on the OS. Note that the above settings are applied to the entire OS.

(4) Controlling the resending of a request

The JAX-RS engine on the Web resource client side communicates by using the HTTP client implemented by the JDK. The HTTP client implemented by the JDK resends the request only once if, in contradiction to RFC 2616, an error occurs in the HTTP communication and the client is unable to receive appropriate responses from the server. Using the system property of the JDK enables you to control the resending of a request. For details, see *10.20 Preventing the resending of a request by sun.net.www.http.HttpClient*. Also, when reading, substitute *Web Services client* with *RESTful Web Services client* and substitute *JAX-WS engine* with *JAX-RS engine*.

(5) Executing a client application using the command line

For details on the necessary settings when using a Java application running from the command line as a client application, examples of specifying the command line, and notes, see *10.14 Executing a client application using the command line*. Also, in the RESTful Web Services client, add the following key and value in the option definition file for Java applications.

```
add.class.path=installation-directory/jaxws/lib/cjjaxrs.jar
```

12 Examples of Developing RESTful Web Services

This chapter describes examples of developing RESTful Web Services (Web resources).

12.1 Configuration of development examples

The examples described in this subsection explain how to develop RESTful Web Services (Web resources). The development includes implementation of root resource classes, sub-resource classes, and an exception mapping provider.

The following table describes the configuration of the Web resources to be developed. An example of this development is available as a sample in the following directory:

Cosminexus-installation-directory\jaxrs\samples\tutorial\

Table 12-1: Configuration of Web resources

No.	Item	Value
1	Name of the J2EE server to be deployed	Jaxrsserver
2	Host name and port number of the Web server	webhost:8085
3	URL of the naming server	corbaname::testserver:900
4	Context root	Tutorial
5	Context path of the root resource class	Root
6	Root resource class	com.sample.resources.Resource
7	Sub-resource class	com.sample.resources.SubResource
8	Exception mapping provider	com.sample.providers.RuntimeExceptionMapper

The following table describes the configuration of a current directory for the Web resource development.

Table 12-2: Configuration of a current directory

Directory	Explanation
c:\temp\jaxrs\works\tutorial	A current directory.
server\	This is to be used in the Web resource development.
META-INF\	This corresponds to the META-INF directory of the EAR file.
application.xml	This is created in *12.3.4 Creating application.xml*.
src\	This stores source files (*.java) of a Web resource. This is used in *12.3.1 Creating root resource classes*.
WEB-INF\	This corresponds to the WEB-INF directory of a WAR file.
web.xml	This is created in *12.3.3 Creating web.xml*.
classes\	This stores the compiled class files (*.class). This is used in *12.3.2 Compiling Java sources*.
tutorial.ear	This is created in *12.3.5 Creating an EAR file*.
tutorial.war	
client\	This is used in the the Web resource client development.
src\	This stores source files (*.java) of a Web resource client. See *12.5.1 Creating Implementation Class of a Web resource client (by using client APIs)* or *12.5.2 Creating Implementation Class of a Web resource client (by using java.net.HttpURLConnection)* for details on how to create the Implementation Class of a Web resource.

Directory	Explanation
classes\	This stores compiled class files (*.class). This is created in *12.3.2 Compiling Java sources*.
usrconf.cfg	This is created in *12.6.1 Creating an options definition file for Java applications*.
usrconf.properties	This is created in *12.6.2 Creating a user property file for Java applications*.

Change the current directory path according to the environment to be developed.

Note that the directories and the file names listed in the above table are used in the descriptions hereafter. The parts in *bold* in the command execution examples and in Java sources indicate the specified and generated values used in the examples. Substitue those parts according to the environment to be built.

Furthermore, in the development examples described in this chapter, Web resources and Web resource clients are developed in the same environment. However, you can also develop them in different environments. For developing Web resources and HTTP clients in different environments, substitute the current directory path according to the respective environments.

12.2 Procedures in the development example

The following procedures are used for developing and executing RESTful Web Services in the development examples described in this chapter:

Developing a Web resource

1. Creating root resource classes (*12.3.1*)
2. Compiling Java sources (*12.3.2*)
3. Creating `web.xml` (*12.3.3*)
4. Creating `application.xml` (*12.3.4*)
5. Creating an EAR file (*12.3.5*)

Deploying and starting

1. Deploying the EAR file (*12.4.1*)
2. Starting Web Services (*12.4.2*)

Developing a Web resource client

1. Creating Implementation Class of a Web resource client (*12.5.1* or *12.5.2*)
2. Compiling Implementation Class of a Web resource client (*12.5.3*)

Calling a Web resource

1. Creating an option definition file for Java applications (*12.6.1*)
2. Creating a user property file for Java applications (*12.6.2*)
3. Starting the Web resource client (*12.6.3*)

12.3 Example of developing Web resources

This section describes an example of developing a Web resource.

12.3.1 Creating root resource classes

You create root resource classes. This subsection describes how to create a root resource class that contains the following resource methods, sub-resource methods, sub-resource locators and the bean properties, fields, and constructors that are to be injected.

Table 12-3: Root resource class to be created

Item		Request method identifier	Context path following / root	Remarks
Resource method	`resourceMethod1`	`@GET`	--	--
	`resourceMethod7`	`@POST`	--	• Includes parameters annotated using the `FormParam` annotation that injects form values. • Receives the HTTP entity of the MIME type "`*/*`". • Returns the HTTP entity of the MIME type "`application/xml`".
Sub-resource method	`subResourceMethod2`	`@GET`	`/getQueryParam`	--
	`subResourceMethod3`	`@POST`	`/getUriInfoAndEntity`	Includes the `Entity` parameter.
	`subResourceMethod4`	`@GET`	`/getHttpHeaders`	Includes the parameters annotated using the `Context` annotation that injects `HttpHeaders`.
	`subResourceMethod5`	`@GET`	`/getMatrixParam`	Includes the parameters annotated using the `MatrixParam` annotation that injects `Matrix` parameters.
	`subResourceMethod6`	`@GET`	`/getCookieParam`	Includes the parameters annotated using the `CookieParam` annotation that injects `Cookies`.
	`subResourceMethod8`	`@GET`	`/getPathParam`	Includes the parameters annotated using the `PathParam` annotation that injects the `path` parameter.
	`subResourceMethodThrowingException`	`@GET`	`/{path:[A-Z][a-z]+}/exception`	Throws exceptions. The exception mapping provider `RuntimeExceptionMapper` maps the thrown exceptions to HTTP entities.
	`pojoJsonMappingMethod`	`@POST`	`/pojoJsonMapping`	Processes data in the JSON format.
Sub-resource locator	`subResourceMethod9`	--	`/subresourceLocator`	Delegates operations to the sub-resource class `SubResource`.

Item	Request method identifier	Context path following / root	Remarks
Constructor	--	--	Includes the parameters annotated using the `Context` annotation that injects `UriInfo`.
Field	--	--	Annotated using the `Context` annotation that injects `Request`.
bean property	--	--	Annotated using the `QueryParam` annotation that injects `query` parameters.

Legend:
--: Not applicable.

An example of creating a root resource class is as follows:

```
package com.sample.resources;

import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.CookieParam;
import javax.ws.rs.FormParam;
import javax.ws.rs.MatrixParam;
import javax.ws.rs.PathParam;
import javax.ws.rs.QueryParam;
import javax.ws.rs.DefaultValue;

import javax.ws.rs.Encoded;

import javax.ws.rs.Consumes;
import javax.ws.rs.Produces;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.HttpHeaders;
import javax.ws.rs.core.Request;
import javax.ws.rs.core.UriInfo;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.Response.ResponseBuilder;

//sample:root resource class
//this class is accessed by URI "/root"
@Path("/root")
public class Resource {

    //injecting to the field of
    //the context "javax.ws.rs.core.Request object"
    private @Context
    Request request;

    // injection of a query parameter by using the setter() method
    @QueryParam("queryParam")

public void setQueryParam(String queryParam) {

        this.queryParam = queryParam;
    }
    //The private field that stores the query parameter injected by   //the
aforementioned setter()method
    private String queryParam;

    // injection to the constructor parameter of
    //the context "(javax.ws.rs.core.UriInfo object)
    public Resource(@Context UriInfo uriInfo) {
        this.uriInfo = uriInfo;
    }
    //Private field that stores the UriInfo object injected by

// the aforementioned constructor
```

```
    private UriInfo uriInfo;

    //Sample: Resource Method 1
    //  returns the context value
    //  injected to the field "javax.ws.rs.core.Request Object"
    // this method processes the HTTP GET request
    @GET
    public String resourceMethod1() {
        String returnString = "RequestMethod: " + request.getMethod();
        return returnString;

}

    //Sample: Sub Resource Method 2
    //returns the value of the query parameter injected to the setter()method

//this method processes the HTTP GET request

    @GET
    //this method is accessed by "/root/getQueryParam" of URI
    @Path("getQueryParam")
    public String subResourceMethod2() {
        String returnString = "QueryParameter: " + queryParam;
        return returnString;

}

    //Sample: Sub Resource Method 3
    //  // returns the context value "javax.ws.rs.core.UriInfo object"
    //  injected to the constructor parameter,

// and returns the received HTTP Request entity body

    // this method processes the HTTP POST request
    @POST
    //this method is accessed by "/root/getUriInfoAndEntity" of URI
    @Path("getUriInfoAndEntity")
    public String subResourceMethod3(
            // an entity parameter (a parameter that is not annotated) is
            // mapped from the HTTP request entity body
            String entity) {
        String returnString = "UriInfo: " + uriInfo.getPath() + "; Entity: " + entity;
        return returnString;
    }

    //Sample:Sub Resource Method 4
    //  returns the context value //(javax.ws.rs.core.HttpHeadersobject)
    //  injected to the sub resource method parameter
    // this method processes the HTTP GET request
    @GET
    // this method is accessed by "/root/getHttpHeaders" of URI
    @Path("getHttpHeaders")
    public String subResourceMethod4(
            //injecting to the subresource method parameter of
            //javax.ws.rs.core.HttpHeaders object
            @Context HttpHeaders httpHeaders) {
        String returnString = "HttpHeaders: " +

httpHeaders.getRequestHeader("header").get(0).toString();
        return returnString;
    }
    // sample: Sub Resource Method 5
    //  returns the value of the matrix parameter injected to
    //  the sub resource method parameter
    // this method processes the HTTP GET request
    @GET
    //this method is accessed by "/root/getMatrixPara" of URI
    @Path("getMatrixParam")
    public String subResourceMethod5(
            // the default value is assigned to the Matrix parameter
            @DefaultValue("defaultValue")
            //injecting the matrix parameter to
            // the subresource method
            @MatrixParam("matrix") String matrixParam) {
        String returnString = "MatrixParam: " + matrixParam;
        return returnString;
    }

    // sample: Sub Resource Method 6
    //  returns the value of Cookie injected to the sub resource method parameter
    //this method processes the HTTP GET request
    @GET
```

```
    //this method is accessed by "/root/getCookieParam"of URI
    @Path("getCookieParam")
    public String subResourceMethod6(
            // disables automatic URI decoding
            @Encoded

// injecting to the sub-resource method parameter of the Cookie

            @CookieParam("cookie") String cookieParam ) {
        String returnString = "CookieParam: " + cookieParam;
        return returnString;
    }
    // sample: Resource Method 7
    returns the value injected to the resource method parameter
    // this method uses the content defined in the MIME media type
    Consumes("*/*")
    // this method uses the content defined in the MIME media type
    //"application/xml"
    @Produces("application/xml")
    // this method processes the HTTP POST request
    @POST
    public Response resourceMethod7(
            // injecting the form parameter to the resource method parameter
            @FormParam("form") String formParam ) {
        ResponseBuilder rb = Response.status(200)
            .entity("<FormParam>" + formParam + "</FormParam>" )
            .type("application/xml");
        return rb.build();

}

    // sample: Sub Resource Method 8
    //  returns the value of the path parameter injected to the sub resource method
parameter
    // this method processes the HTTP GET request
    @GET
    //this method is accessed by
    //"/root/getPathParam/{path:[A-Z][a-z]+}" of URI
    //"getPathParam/{path:[A-Z][a-z]+}" is the URI template containing variables
    //"{path:[A-Z][a-z]+}" uses regular expressions
    @Path("getPathParam/{path:[A-Z][a-z]+}")
    public String subResourceMethod8(
            // injecting to the path parameter
            // of the sub resource method parameter
            @PathParam("path") String pathParam ) {
        String returnString = "PathParam: " + pathParam;
        return returnString;
    }

    // sample: Sub Resource Locator 9
    // The sub resource class is accessed by "/root/subresourceLocator" URI
    @Path("/subresourceLocator")
    public SubResource subResourceMethod9() {
        // the sub-resource locator returns a resource class instance
        // to process the HTTP request
        return new SubResource();
    }
    // sample:Sub Resource Method 10
    //  throws an exception
    // Exception mapping provider maps
    //the thrown exception to the HTTP response
    // this method processes the HTTP GET request
    @Path("/exception")
    //this method is accessed by "/root/exception" URI
    @GET
    public String subResourceMethodThrowingException() {
        throw new RuntimeException();

}

    //Sample: Sub Resource method 11
    //  processes data in the JSON format
    //This method processes the HTTP POST request
    @Path("pojoJsonMapping")
    //This method is accessed by "root/PojoJsonMapping" URI
    @POST
    @Consumes(MediaType.APPLICATION_JSON)
    @Produces(MediaType.APPLICATION_JSON)
    public CustomType pojoJsonMappingMethod(CustomType record) {
        //Confirm the contents of the record object received from //the client
        if(!record.getName().equals("Old Record Name")){
            throw new RuntimeException();
```

```
        }
        List<Integer> oldGrades = new ArrayList<Integer>();
        oldGrades.add(1);
        oldGrades.add(2);
        oldGrades.add(3);
        if(!record.getGrades().equals(oldGrades)){
            throw new RuntimeException();

}

        // Create a new CustomType object to return to the client
        CustomType newRecord = new CustomType();
        newRecord.setName("New Record Name");
        List<Integer> newGrades = new ArrayList<Integer>();
        newGrades.add(5);
        newGrades.add(6);
        newGrades.add(7);
        newRecord.setGrades(newGrades);

        return newRecord;

}
```

You save the created root resource class (`Resource.java`) in the `c:\temp\jaxrs\works\tutorial\server\src\com\sample\resources\` directory in the UTF-8 format.

An example of creating a sub-resource class is as follows. This sub-resource class has a resource method that receives the `HTTP GET` request. Note that creating a sub-resource class is optional.

```
package com.sample.resources;

import javax.ws.rs.GET;
import javax.ws.rs.HeaderParam;

//Sample: Sub Resource class
public class SubResource {

    //This method processes the HTTP GET request
    @GET
    public String getHandlerForSubResource(
            //Injecting header parameters to
            //the sub resource method parameters
            @HeaderParam("header") String headerParam) {
        return "Header: " + headerParam;
    }

}
```

You save the created sub-resource class (`SubResource.java`) in the `c:\temp\jaxrs\works\tutorial\server\src\com\sample\resources\` directory in the UTF-8 format.

An example of creating an exception mapping provider is as follows. The Exception Mapping Provider maps the runtime exception to the HTTP response that contains the value of "RuntimeException occurs" in the HTTP header and has the status code 501. Note that creating an Exception Mapping Provider is optional.

```
package com.sample.providers;

import javax.ws.rs.core.Response;
import javax.ws.rs.ext.Provider;
import javax.ws.rs.ext.ExceptionMapper;

//sample: exception mapping class
//this class maps RuntimeException to HTTP response
@Provider
public class RuntimeExceptionMapper implements
        ExceptionMapper<RuntimeException> {
    //this method is called when the application throws
    //a runtime exception
    public Response toResponse(RuntimeException re) {
        //sets "HTTP/1.1 501 Not Implemented" as the HTTP response //status
        //sets "header: RuntimeException occurs" in the header of //the HTTP response
        return Response.status(501)
            .header("header", "RuntimeException occurs").build();
    }
}
```

You save the created exception mapping provider (`RuntimeExceptionMapper.java`) in the `c:\temp\jaxrs\works\tutorial\server\src\com\sample\providers\` directory in the UTF-8 format.

An example of creating a POJO to be mapped to the data in the JSON format is as follows. Note that creating a POJO for mapping to the JSON fornat is an optional operation.

```
package com.sample.resources;

import java.util.List;

public class CustomType {

    private String name;
    private List<Integer> grade;

    public CustomType() {
    }
    public CustomType (String name, List<Integer> grades){
        this.name = name;
        this.grade = grades;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public List<Integer> getGrades() {
        return grade;
    }
    public void setGrades(List<Integer> grades) {
        this.grade = grades;
    }
    @Override
    public String toString() {
        return "Record [Name=" + name + ", Grades=" + grade.toString() + "]";
    }

}
```

Save the POJO created (`CustomType.java`) for mapping to JSON in the `c:\temp\jaxrs\works\tutorial\server\src\com\sample\resources\`directory in the UTF-8 format.

12.3.2 Compiling Java sources

You use the `javac` command to compile the created Java sources. An example of compiling is as follows:

```
> cd c:\temp\jaxrs\works\tutorial\server\
> javac -encoding UTF-8 -cp "%COSMINEXUS_HOME%\jaxws\lib\cjjaxws.jar;%COSMINEXUS_HOME%
\jaxrs\lib\cjjaxrs.jar;%COSMINEXUS_HOME%\jaxp\lib\csmjaxb.jar;%COSMINEXUS_HOME%\jaxp
\lib\csmjaxp.jar;%COSMINEXUS_HOME%\jaxp\lib\csmstax.jar;.\classes" -d WEB-INF\classes
\ src\com\sample\resources\Resource.java src\com\sample\resources\SubResource.java src
\com\sample\providers\RuntimeExceptionMapper.java src\com\sample\resources
\CustomType.java
```

When the `javac` command ends successfully, a class file (`*.class`) is generated in the subdirectory corresponding to the package name in the `c:\temp\jaxrs\works\tutorial\server\WEB-INF\classes\` directory.

For details on the `javac` command, see the *JDK documentation*.

12.3.3 Creating web.xml

You create `web.xml` that is required as a WAR file component.

The example of creating `web.xml` is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/
javaee/web-app_3_0.xsd">

    <description>Sample web service &quot;tutorial&quot;</description>
    <display-name>Sample_web_service_jaxrs_tutorial</display-name>
    <servlet>
        <servlet-name>CosminexusJaxrsServlet</servlet-name>
        <servlet-class>
            com.cosminexus.jersey.spi.container.servlet.ServletContainer
        </servlet-class>
        <init-param>
            <param-name>com.sun.jersey.config.property.packages</param-name>
            <param-value>com.sample.resources;com.sample.providers</param-value>
        </init-param>
        <!-- POJO JSON support web.xml configuration -->
        <init-param>
            <param-name>com.sun.jersey.api.json.POJOMappingFeature</param-name>
            <param-value>true</param-value>
        </init-param>
    </servlet>
    <servlet-mapping>
        <servlet-name>CosminexusJaxrsServlet</servlet-name>
        <url-pattern>/*</url-pattern>
    </servlet-mapping>
</web-app>
```

When creating `web.xml` of version 2.5, specify `2.5` in the `version` attribute of the `web-app` element and specify `http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd` as the second location information in the `xsd:schemaLocation` attribute.

Save the created `web.xml` in the `c:\temp\jaxrs\works\tutorial\server\WEB-INF\` directory in the UTF-8 format. For details on the `web.xml` settings, see *3.4 Creating web.xml*.

12.3.4 Creating application.xml

You create `application.xml` required as a component of the EAR file.

The following is an example of creating `application.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<application version="6"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/
javaee/application_6.xsd">

    <description>Sample application &quot;tutorial&quot;</description>
    <display-name>Sample_application_jaxrs_tutorial</display-name>
    <module>
        <web>
            <web-uri>tutorial.war</web-uri>
            <context-root>tutorial</context-root>
        </web>
    </module>
</application>
```

When creating `application.xml` of version 5, specify `5` in the `version` attribute of the `application` element and specify `http://java.sun.com/xml/ns/javaee/application_5.xsd` as the second location information in the `xsd:schemaLocation` attribute.

You save the created `application.xml` in the `c:\temp\jaxrs\works\tutorial\server\META-INF\` directory in the UTF-8 format. For notes on creating the `application.xml` file, see *5.2.2 Points to consider when editing application.xml* in the *uCosminexus Application Server Application Development Guide*.

12.3.5 Creating an EAR file

You use the `jar` command to create an EAR file that includes the files created so far.

The following is an example of creating an EAR file:

```
> cd c:\temp\jaxrs\works\tutorial\server\
> jar cvf tutorial.war .\WEB-INF
> jar cvf tutorial.ear .\jaxrs_sample.war .\META-INF\application.xml
```

When the `jar` command ends successfully, `tutorial.ear` is created in the `c:\temp\jaxrs\works\tutorial\server\` directory.

For details on the `jar` command, see the *JDK documentation*.

12.4 Examples of deploying and starting

This section describes the examples of deploying and starting Web resources.

12.4.1 Deploying EAR files

You use the `cjimportapp` command to deploy the created EAR file to the J2EE server.

The following example describes how to deploy EAR files:

```
> cd c:\temp\jaxrs\works\tutorial\server\
> "%COSMINEXUS_HOME%\CC\admin\bin\cjimportapp" jaxrsserver -nameserver
corbaname::testserver:900 -f tutorial.ear
```

For the `cjimportapp` command, see *cjimportapp (Importing J2EE applications)* in the *uCosminexus Application Server Command Reference Guide*.

For details on how to deploy (import) J2EE applications using the management portal, see *12.3.3 Importing J2EE applications* in the *uCosminexus Application Server Management Portal User Guide*.

12.4.2 Starting Web Services

You use the `cjstartapp` command to start Web Services.

The following example describes how to start Web Services:

```
> cd c:\temp\jaxrs\works\tutorial\server\
> "%COSMINEXUS_HOME%\CC\admin\bin\cjstartapp" jaxrsserver -nameserver
corbaname::testserver:900 -name Sample_application_jaxrs_tutorial
```

For details on the `cjstartapp` command, see *cjstartapp (Starting J2EE applications)* in the *uCosminexus Application Server Command Reference Guide*.

For details on how to start J2EE applications using the management portal, see *12.3.1 Starting J2EE applications* in the *uCosminexus Application Server Management Portal User Guide*.

12.5 Examples of developing a Web resource client

This section describes an example of developing a Web resource client. Develop a Web resource client by using the standard Java APIs such as client APIs for RESTful Web Services, `java.net.URL`, or `java.net.HttpURLConnection`.

12.5.1 Creating Implementation Class of a Web resource client (by using the client APIs)

Create Implementation Class of the client that uses client APIs for RESTful Web Services.

An example is as follows:

```
package com.sample.client;

import java.net.URI;
import java.net.URLEncoder;
import java.util.List;
import java.util.ArrayList;
import java.util.Map;

import com.cosminexus.jersey.api.client.Client;
import com.cosminexus.jersey.api.client.WebResource;
import com.cosminexus.jersey.api.client.ClientRequest;
import com.cosminexus.jersey.api.client.ClientResponse;
import com.cosminexus.jersey.api.client.config.ClientConfig;
import com.cosminexus.jersey.api.client.config.DefaultClientConfig;
import com.cosminexus.jersey.api.client.ClientHandlerException;
import com.cosminexus.jersey.api.json.JSONConfiguration;
import javax.ws.rs.core.Cookie;

//Sample: Executing the Web resource client
public class SampleClient {

    public static void main(String[] args) {

        final String HOST = args[0];
        final String PORT = args[1];

        SampleClient sampleClient = new SampleClient();

        try{
            sampleClient.demonstration13(HOST, PORT);
            sampleClient.demonstration14(HOST, PORT);
            sampleClient.demonstration15(HOST, PORT);

            System.out.println("\n----- Successfully Ended -----");

        }catch(Exception e){
            //Display a detailed exception message
            System.out.println(e.getMessage());
        }
    }
    private void demonstration13(String HOST, String PORT) {

        System.out.println("\n Demonstration 13 started.");
        System.out.println(" This demonstrates how to use Client API " +
                "to receive a response as a ClientResponse.");
        System.out.println(" This demonstrates usage of @Encoded at " +
                "@CookieParam. \n Automatic URI decoding should be disabled.");

        String url = null;
        Client client = null;
        ClientResponse response = null;

        String responseEntity = "";
        Map<String, List<String>> headers = null;
        int status;

        //Call the Web resource
        try {
            //Set a URI of the Web resource to be called
            url = new String("http://" + HOST + ":" + PORT+
                    "/tutorial/root/getCookieParam");
            Cookie cookie = new Cookie("cookie", "cookie%20value");
```

```
            //Create a client object to use client APIs
            client = Client.create();
            //Create and send an HTTP request and receive an HTTP response
            //- Create a WebResource object from the Client object
            //- Set "cookie=cookie%20value" to the Cookie header
            //- Send an HTTP GET request and
            //  receive an HTTP response as ClientResponse
            response = client.resource(url)
                             .cookie(cookie)
                             .get(ClientResponse.class);
            //Acquire headers of the HTTP response
            headers = response.getHeaders();
            //Acquire the status code of the HTTP response
            status = response.getStatus();
            //Acquire the entity of the HTTP response
            responseEntity = response.getEntity(String.class);
        }catch (Exception e) {
            System.out.println(" ERROR: " + e.getClass() + " was thrown. ");
            //Display stack trace
            e.printStackTrace();
            throw new RuntimeException(" Demonstration 13 failed.");
        }
        System.out.println(" The target URL is \"" + url + "\".");
        System.out.println(" The HTTP method  is " + "\"GET\"" + ".");
        System.out.println(" Connection and interaction ended successfully.");

        //Set the expected value of the status code and entity
        int statusExpect = 200;
        String responseEntityExpect = "CookieParam: cookie%20value";

        //Check if the status code and the entity have the expected values
        if (status == statusExpect
                & responseEntity.equals(responseEntityExpect)) {
            //Display the header of the HTTP response
            System.out.println(" Response headers are " + headers.toString() + ".");
            //Display the entity of the HTTP response
            System.out.println(" Response entity is " + responseEntity + ",");
            System.out.println(" which means target resource completed " +
                    "the process described above without any problem.");

            System.out.println(" Demonstration 13 ended successfully.");
        }else {
            System.out.println(" The response is not as expected.");
            throw new RuntimeException(" Demonstration 13 failed.");
        }

    }

    private void demonstration14(String HOST, String PORT) {

        System.out.println("\n Demonstration 14 started.");
        System.out.println(" This demonstrates how to send a ClientRequest " +
                "and receive a ClientResponse by using " +
                "Client#handle(ClientRequest request).");
        System.out.println(" This demonstrates usage of @Consumes and " +
                "@Produces.");
        URI url = null;
        Client client = null;
        ClientRequest.Builder requestBuilder = null;
        ClientRequest request = null;
        ClientResponse response = null;

        String responseEntity = "";
        Map<String, List<String>> headers = null;
        int status;

        //call the Web resource
        try {
            //Set the URI of the Web resource to be called
            url = new URI("http://" + HOST + ":" + PORT+ "/tutorial/root");
            //Create an entity of the HTTP request
            String data = URLEncoder.encode("form", "UTF-8") + "="
                + URLEncoder.encode("formValue", "UTF-8");
            //crate a Client object to use the client APIs
            client = Client.create();
            //Create a ClientRequest.Builder object
            requestBuilder = ClientRequest.create();
            //  Set "application/x-www-form-urlencoded"
            //to the "Content-Type" header of the HTTP request
            //- Set the entity of the HTTP request
               requestBuilder.type("application/x-www-form-urlencoded")
                             .entity(data);
            //Create ClientRequest from ClientRequest.Builder
```

```
            request = requestBuilder.build(url, "POST");
            //Call the Client#handle() method and send the HTTP POST //request
            //Receive an HTTP response as ClientResponse
            response = client.handle(request);
            //Acquire headers of the HTTP response
            headers = response.getHeaders();
            //Acquire the status code of the HTTP response
            status = response.getStatus();
            //Acquire the entity of the HTTP response
            responseEntity = response.getEntity(String.class);
        }catch (ClientHandlerException e) {
            System.out.println(" ERROR: " + e.getClass() + " was thrown. ");
            //Display stack trace
            e.printStackTrace();
            throw new RuntimeException(" Demonstration 14 failed.");
        }catch (Exception e) {
            System.out.println(" ERROR: " + e.getClass() + " was thrown. ");
            // Display stack trace
            e.printStackTrace();
            throw new RuntimeException(" Demonstration 14 failed.");
        }
        System.out.println(" The target URL is \"" + url + "\".");
        System.out.println(" The HTTP method  is " + "\"POST\"" + ".");
        System.out.println(" Connection and interaction ended successfully.");

        //Set the expected values of the status code and the entity
        int statusExpect = 200;
        String responseEntityExpect = "<FormParam>formValue</FormParam>";

        //Check if the status code and the entity have the expected values
        if (status == statusExpect
                & responseEntity.equals(responseEntityExpect)) {
            //Display headers of the HTTP response
            System.out.println(" Response headers are " + headers.toString() + ".");
            //Display the entity of the HTTP response
            System.out.println(" Response entity is " + responseEntity + ",");
            System.out.println(" which means target resource completed " +
                    "the process described above without any problem.");

            System.out.println(" Demonstration 14 ended successfully.");
        }else {
            System.out.println(" The response is not as expected.");
            throw new RuntimeException(" Demonstration 14 failed.");
        }

    }
    private void demonstration15(String HOST, String PORT) {

        System.out.println("\n Demonstration 15 started.");
        System.out.println(" This demonstrates JSON support of CJR.");
        System.out.println(" This demonstrates POJO and JSON mapping.");

        String url = null;
        Client client = null;
        CustomType response = null;

        //Call the Web resource
        try {
            //Set the URI of the Web resource to be called
            url = new String("http://" + HOST + ":" + PORT +
                    "/tutorial/root/PojoJsonMapping");
            //Specify settings to enable JSON POJO mapping
            ClientConfig cc = new DefaultClientConfig();
            cc.getFeatures()
              .put(JSONConfiguration.FEATURE_POJO_MAPPING, Boolean.TRUE);
            //Create a Client object to use client APIs
            //(Pass the ClientConfig object to enable the settings)
            client = Client.create(cc);
            //Create a CustomType object
            CustomType record = new CustomType();
            record.setName("Old Record Name");
            List<Integer> grades = new ArrayList<Integer>();
            grades.add(1);
            grades.add(2);
            grades.add(3);
            record.setGrades(grades);
            //Create an HTTP request
            //-Create a WebResource object from the Client object
            //-Set "application/json" to the Content-Type header
            //-Set CustomType object to the entity
            //-Send HTTP POST request and
            //receive HTTP response as CustomType
            response = client.resource(url)
```

```
                            .type("application/json")
                            .entity(record)
                            .post(CustomType.class);
        }catch (Exception e) {
            System.out.println(" ERROR: " + e.getClass() + " was thrown. ");
            //Display the stack trace
            e.printStackTrace();
            throw new RuntimeException(" Demonstration 15 failed.");
        }
        System.out.println(" The target URL is \"" + url + "\".");
        System.out.println(" The HTTP method  is " + "\"POST\"" + ".");
        System.out.println(" Connection and interaction ended successfully.");

        //Set the expected value of the entity
        String responseNameExpect = "New Record Name";
        List<Integer> responseGradesExpect = new ArrayList<Integer>();
        responseGradesExpect.add(5);
        responseGradesExpect.add(6);
        responseGradesExpect.add(7);

        //Check if the entity has the expected value
        if (response.getName().equals(responseNameExpect)
                & response.getGrades().equals(responseGradesExpect)) {
            //Display the entity of the HTTP response
            System.out.println(" Response is " + response.toString() + ",");
            System.out.println(" which means target resource completed " +
                    "the process described above without any problem.");

            System.out.println(" Demonstration 15 ended successfully.");
        }else {
            System.out.println(" The response is not as expected.");
            throw new RuntimeException(" Demonstration 15 failed.");
        }

    }
    private static class CustomType {

        private String name;
        private List<Integer> grade;

        public CustomType() {
        }

        public CustomType (String name, List<Integer> grades){
            this.name = name;
            this.grade = grades;
        }

        public String getName() {
            return name;
        }

        public void setName(String name) {
            this.name = name;
        }

        public List<Integer> getGrades() {
            return grade;
        }

        public void setGrades(List<Integer> grades) {
            this.grade = grades;
        }

        @Override
        public String toString() {
            return "Record [Name=" + name + ", Grades=" + grade.toString() + "]";
        }

    }
}
```

Save the created `SampleClient.java` in the `c:\temp\jaxrs\works\tutorial\client\src\com\sample\client\` directory in the UTF-8 format.

12.5.2 Creating Implementation Class of a Web resource client (by using java.net.HttpURLConnection)

Create Implementation Class of the client that uses the `HttpURLConnection` class.

The following example describes how to create an implementation class of a Web resource client:

```
package com.sample.client;

import java.net.URL;
import java.net.URLEncoder;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.BufferedReader;
import java.io.IOException;
import java.util.List;
import java.util.Map;

//sample:starting the Web resource client
public class SampleClient {

    public static void main(String[] args) {

        final String HOST = args[0];
        final String PORT = args[1];

        SampleClient sampleClient = new SampleClient();

        try{
            sampleClient.demonstration1(HOST, PORT);
            sampleClient.demonstration2(HOST, PORT);

            System.out.println("\n----- Successfully Ended -----");

        }catch(Exception e){
            //displaying a detailed exception message
            System.out.println(e.getMessage());
        }
    }

    private void demonstration1(String HOST, String PORT) {

        System.out.println("\n Demonstration 1 started.");
        System.out.println(" This demonstrates injection of " +
                "javax.ws.rs.core.Request instance " +
                "onto resource class field by using @Context.");

        URL url = null;
        HttpURLConnection httpConn = null;

        Map<String, List<String>> headers = null;
        List<String> status = null;
        String responseEntity = "";
```

```
        //calling the Web resource
        try {
            //setting the URI of the Web resource to be called
            url = new URL("http://" + HOST + ":" + PORT+
                    "/tutorial/root");
            //setting up the initial connection
            httpConn = (HttpURLConnection) url.openConnection();
            //setting the HTTP() method as "GET"
            httpConn.setRequestMethod("GET");
            //starting the connection
            httpConn.connect();
        }catch (MalformedURLException e) {
            System.out.println(" ERROR: " + e.getClass() + " was thrown. ");
            //displaying the stack trace
            e.printStackTrace();
            throw new RuntimeException(" Demonstration 1 failed.");
        }catch (IOException e) {
            System.out.println(" ERROR: " + e.getClass() + " was thrown. ");
            //displaying the stack trace
            e.printStackTrace();
            throw new RuntimeException(" Demonstration 1 failed.");
        }catch (Exception e) {
            System.out.println(" ERROR: " + e.getClass() + " was thrown. ");
            //displaying the stack trace
```

```
            e.printStackTrace();
            throw new RuntimeException(" Demonstration 1 failed.");
        }

        System.out.println(" The target URL is \"" + url + "\".");
        System.out.println(" The HTTP method  is " + "\"GET\"" + ".");

        try {
            //acquiring the HTTP response header
            headers = httpConn.getHeaderFields();
            //acquiring the HTTP status from the header
            status = headers.get(null);
            //acquiring the input stream reader and read HTTP response //entity body
            BufferedReader rd = new BufferedReader(new InputStreamReader(
                httpConn.getInputStream()));
            String line = "";
            while ((line = rd.readLine()) != null) {
                responseEntity += line;
            }
            //closing the input stream reader
            rd.close();
            //closing the connection
            httpConn.disconnect();
        }catch (Exception e) {
            System.out.println(" ERROR: " + e.getClass() + " was thrown. ");
            //displaying the stack trace
            e.printStackTrace();
```

```
            throw new RuntimeException(" Demonstration 1 failed.");
        }

        System.out.println(" Connection and interaction ended successfully.");

        //sets the expected value of the response status code and entity body
        String statusExpect = "[HTTP/1.1 200 OK]";
        String responseEntityExpect = "RequestMethod: GET";

        //checks whether the value of the status code and entity body is equal to the
expected value
        if (status.toString().equals(statusExpect)
                & responseEntity.equals(responseEntityExpect)) {
            //displaying the HTTP header
            System.out.println(" Response headers are " + headers.toString() + ".");
            //displaying the entity body
            System.out.println(" Response entity is " + responseEntity + ",");
            System.out.println(" which means target resource completed " +
                    "the process described above without any problem.");

            System.out.println(" Demonstration 1 ended successfully.");
        }else {
            System.out.println(" The response is not as expected.");
            throw new RuntimeException(" Demonstration 1 failed.");
        }

    private void demonstration2(String HOST, String PORT) {

        System.out.println("\n Demonstration 2 started.");
        System.out.println(" This demonstrates injection of QueryParam " +
                "onto resource bean setter method by using @QueryParam.");

        URL url = null;
        HttpURLConnection httpConn = null;
        String responseEntity = "";
        Map<String, List<String>> headers = null;
        List<String> status = null;

        //calling the Web resource
        try {
            //setting the URI of the Web resource to be called
            //assigning the query parameter to the URI

            url = new URL("http://" + HOST + ":" + PORT+
                    "/tutorial/root/getQueryParam?queryParam=queryValue");
            //setting up the initial connection
            httpConn = (HttpURLConnection) url.openConnection();
            //setting the HTTP() method as"GET"
            httpConn.setRequestMethod("GET");
            //starting the connection
            httpConn.connect();
```

```
        }catch (MalformedURLException e) {
            System.out.println(" ERROR: " + e.getClass() + " was thrown. ");
```

```
            //displaying the stack trace
            e.printStackTrace();
            throw new RuntimeException(" Demonstration 2 failed.");
        }catch (IOException e) {
            System.out.println(" ERROR: " + e.getClass() + " was thrown. ");
            //displaying the stack trace
            e.printStackTrace();
            throw new RuntimeException(" Demonstration 2 failed.");
        }catch (Exception e) {
            System.out.println(" ERROR: " + e.getClass() + " was thrown. ");
            //displaying the stack trace
            e.printStackTrace();
            throw new RuntimeException(" Demonstration 2 failed.");
        }

        System.out.println(" The target URL is \"" + url + "\".");
        System.out.println(" The HTTP method  is " + "\"GET\"" + ".");

        try {
            //acquiring the HTTP header
            headers = httpConn.getHeaderFields();
            //acquiring the HTTP status code from the HTTP header
            status = headers.get(null);
            //acquiring the input stream reader and reading the response //entity
body of the HTTP response
            BufferedReader rd = new BufferedReader(new InputStreamReader(
                httpConn.getInputStream()));
            String line = "";
            while ((line = rd.readLine()) != null) {
                responseEntity += line;
            }
            //closing the input stream reader
            rd.close();
            //closing the connection
            httpConn.disconnect();
        }catch (Exception e) {
            System.out.println(" ERROR: " + e.getClass() + " was thrown. ");
            //displaying the stack trace
            e.printStackTrace();
            throw new RuntimeException(" Demonstration 2 failed.");
        }

        System.out.println(" Connection and interaction ended successfully.");

        //setting the expected value of the response status code and entity body
        String statusExpect = "[HTTP/1.1 200 OK]";
        String responseEntityExpect = "QueryParameter: queryValue";

        //checking whether the status code and entity body are same as the expected
values
        if (status.toString().equals(statusExpect)
                & responseEntity.equals(responseEntityExpect)) {
            //displaying the HTTP header
            System.out.println(" Response headers are " + headers.toString() + ".");
            //displaying the entity body
            System.out.println(" Response entity is " + responseEntity + ",");
            System.out.println(" which means target resource completed " +
                    "the process described above without any problem.");

            System.out.println(" Demonstration 2 ended successfully.");
        }else {
            System.out.println(" The response is not as expected.");
            throw new RuntimeException(" Demonstration 2 failed.");
        }

    }
    }

}
```

You save the created `SampleClient.java` in the `c:\temp\jaxrs\works\tutorial\client\src\com\sample\client\` directory in the UTF-8 format.

12.5.3 Compiling Implementation Classes of a Web resource client

You use the `javac` command to compile the created Web resource clients.

The following example describes how to compile HTTP clients:

```
> cd c:\temp\jaxrs\works\tutorial\client\
> javac -encoding UTF-8 -cp "%COSMINEXUS_HOME%\jaxws\lib\cjjaxws.jar;%COSMINEXUS_HOME%
\jaxrs\lib\cjjaxrs.jar;%COSMINEXUS_HOME%\jaxp\lib\csmjaxb.jar;.\classes" -d .\classes
src\com\sample\client\SampleClient.java
```

When the `javac` command ends successfully, a class file (`*.class`) is generated in the subdirectory corresponding to the package name under the `c:\temp\jaxrs\works\tutorial\client\classes\` directory.

For details on the `javac` command, see the *JDK documentation*.

12.6 Examples of invoking Web resources

This section describes an example of invoking Web resources.

12.6.1 Creating an option definition file for Java applications

You create an option definition file (`usrconf.cfg`) for Java applications required for executing Web Services.

The following example describes how to create an option definition file for Java applications:

```
add.class.path= Cosminexus-installation-directory\jaxrs\lib\cjjaxws.jar
add.class.path=.\classes
ejb.client.log.directory=logs
add.jvm.arg=-Dcosminexus.home=Cosminexus-installation-directory
add.jvm.arg=-Dejbserver.server.prf.PRFID=PRF ID
```

In the *Cosminexus-installation-directory* part, you specify the absolute path of the location where you have installed Cosminexus. In the `PRF ID` part, specify the identifier of the PRF daemon.

You save the created option definition file for Java applications in the `c:\temp\jaxrs\works\tutorial\client\` directory. For details on the option definition file for Java applications, see *14.2 usrconf.cfg (Option definition file for Java applications)* in the *uCosminexus Application Server Definition Reference Guide*.

12.6.2 Creating a user property file for Java applications

You create a user property file for Java applications required for executing Web Services.

Because the settings are not specifically changed, you create an empty file called *usrconf.properties* in the `c:\temp\jaxrs\works\tutorial\client\` directory. For details on the user property file for Java applications, see *14.3 usrconf.properties (User property file for Java applications)* in the *uCosminexus Application Server Definition Reference Guide*.

12.6.3 Starting a Web resource client

You use the `cjclstartap` command to start a Web resource client.

An example of starting the Web resource client is as follows:

```
> cd c:\temp\jaxrs\works\tutorial\client\
> "%COSMINEXUS_HOME%\CC\client\bin\cjclstartap"  com.sample.client. SampleClient
webhost 8085
```

When the `cjclstartap` command ends successfully, the execution result of the Web resource client is displayed. The following example describes the execution results of each method for developing a Web resource client:

- When using client APIs

```
KDJE40053-I The cjclstartap command will now start. (directory for the user
definition file = c:\temp\jaxrs\works\tutorial\client, PID = 2636)

Demonstration 13 started.
 This demonstrates how to use a Client API to receive a response as a
ClientResponse.
 This demonstrates usage of @Encoded at @CookieParam.
 Automatic URI decoding should be disabled.
 The target URL is "http://webhost:8085/tutorial/root/getCookieParam".
 The HTTP method is "GET".
 Connection and interaction ended successfully.
 Response headers are {Transfer-Encoding=[chunked], Date=[Tue, 27 Dec 2011 0
7:59:41 GMT], Content-Type=[text/html], Server=[CosminexusComponentContainer]}.
 Response entity is CookieParam: cookie%20value,
 which means target resource completed the process described above without any
problem.
 Demonstration 13 ended successfully.
```

```
 Demonstration 14 started.
 This demonstrates how to send a ClientRequest and receive a ClientResponse by
using Client#handle(ClientRequest request).
 This demonstrates usage of @Consumes and @Produces.
 The target URL is "http://webhost:8085/tutorial/root".
 The HTTP method is "POST".
 Connection and interaction ended successfully.
 Response headers are {Transfer-Encoding=[chunked], Date=[Tue, 27 Dec 2011 0
7:59:41 GMT], Content-Type=[application/xml],
Server=[CosminexusComponentContainer]}.
 Response entity is <FormParam>formValue</FormParam>,
 which means target resource completed the process described above without any
problem.
 Demonstration 14 ended successfully.

 Demonstration 15 started.
 This demonstrates JSON support of CJR.
 This demonstrates POJO and JSON mapping.
 The target URL is "http://webhost:8085/tutorial/root/PojoJsonMapping".
 The HTTP method is "POST".
 Connection and interaction ended successfully.
 Response is Record [Name=New Record Name, Grades=[5, 6, 7]],
 which means target resource completed the process described above without any
problem.
 Demonstration 15 ended successfully.

----- Successfully Ended -----
KDJE40054-I The cjclstartap command was stopped. (PID = 2636, exit status = 0)
```

- When using `HttpURLConnection`

```
KDJE40053-I The cjclstartap command will now start. (directory for the user
definition file = c:\temp\jaxrs\works\tutorial\client, PID = 2636)

Demonstration 1 started.
 This demonstrates injection of the javax.ws.rs.core.Request instance into the
resource class field by using @Context.
 The target URL is "http://webhost:8085/tutorial/root".
 The HTTP method is "GET".
 Connection and interaction ended successfully.
 Response headers are {null=[HTTP/1.1 200 OK], Transfer-Encoding=[chunked],
Date=[Tue, 27 Dec 2011 0
7:59:41 GMT], Content-Type=[text/html], Server=[CosminexusComponentContainer]}.
 Response entity is RequestMethod: GET,
 which means the target resource completed the process described above without any
problem.
 Demonstration 1 ended successfully.

 Demonstration 2 started.
 This demonstrates injection of QueryParam onto the resource bean setter method by
using @QueryParam.
 The target URL is "http://webhost:8085/tutorial/root/getQueryParam?
queryParam=queryValue".
 The HTTP method is "GET".
 Connection and interaction ended successfully.
 Response headers are {null=[HTTP/1.1 200 OK], Transfer-Encoding=[chunked],
Date=[Tue, 27 Dec 2011 0
7:59:41 GMT], Content-Type=[text/html], Server=[CosminexusComponentContainer]}.
 Response entity is QueryParameter: queryValue,
 which means the target resource completed the process described above without any
problem.
 Demonstration 2 ended successfully.

----- Successfully Ended -----
KDJE40054-I The cjclstartap command was stopped. (PID = 2636, exit status = 0)
```

The part in *Italics* changes according to the execution time and environment.

For details on the `cjclstartap` command, see *cjclstartap (Starting Java applications)* in the *uCosminexus Application Server Command Reference Guide*.

13 Settings and Operations of the JAX-RS Functionality

This chapter describes the various settings of the JAX-RS functionality used when developing or operating RESTful Web Services (Web Service), gives an overview of connection from the Web resource clients, and also describes the operations of the JAX-RS engine that you must understand when operating RESTful Web Services.

13.1 Action definition file

Code the items such as log settings in an Action definition file. The Action definition file comprises the following two types:

- *Common definition file*

  A common definition file is used for setting up the common system operations. Only one common definition file is available.

- *Process-wise definition file*

  Create a process-wise definition file when you want to set up the process-specific operations. Create one process-wise definition file for each process that requires specific settings. For example, create process-wise definition file(s) when you want to change the settings of each J2EE server or Web Service client.

You can overwrite some of the definitions of a Web resource by using the definitions of the `web.xml` filters or initialization parameters. For details on `web.xml`, see *11.2 Creating web.xml*. You can also overwrite some of the definitions of the client through client APIs. For definitions that you can overwrite through the client APIs, see *25.1.1 Supported properties and features*.

This sub-section describes the coding standards for an action definition file, and the settings of each definition file.

13.1.1 Coding rules for the action definition file

The action definition file has the same coding format, coding rules and priority definitions as that of the JAX-WS functionality. For details, see *10.1.1 Coding rules for the action definition file.*

13.1.2 Settings of a common definition file

Use a common definition file to define the common system operations. The following points describe various details such as the file name, storage directory name, and settings of a common definition file:

(1) File name

The file name of a common definition file is as follows.

`cjrconf.properties`

(2) Storage directory

The storage directory of a common definition file is as follows. The storage destination is fixed.

Cosminexus-installation-directory`\jaxrs\conf`

(3) Settings

The following table lists and describes the key names to be set up and the corresponding values to be specified.

Table 13–1: Settings of a common definition file

No.	Settings	Key name	Values to be specified	Default value
1	Output level of the operation log	`com.cosminexus.jaxrs.logger.runtime.message.level`	Specify the output level of the operation log. Specify `ERROR`, `WARN`, `INFO`, `DEBUG`, or `NONE`. For details on the output contents corresponding to each of the specified values, see *39.3.4 Log importance and the output conditions*.	`INFO`
2	Number of operation log files	`com.cosminexus.jaxrs.logger.ru`	Specify the number of the operation log files. Specify a value in the range of 1 through 16.	`2`

No.	Settings	Key name	Values to be specified	Default value
2	Number of operation log files	ntime.message.file_num	Specify the number of the operation log files. Specify a value in the range of 1 through 16.	2
3	Size of the operation log	com.cosminexus.jaxrs.logger.runtime.message.file_size	Specify the size of the operation log. Specify a value in the range of 4096 through 16777216 (unit: bytes).	2097152
4	Output of the maintenance log	com.cosminexus.jaxrs.logger.runtime.maintenance.level	Specify whether to output the maintenance log. When you specify ALL, the maintenance log is output. When you specify NONE, the maintenance log is not output.	ALL
5	Number of maintenance log files	com.cosminexus.jaxrs.logger.runtime.maintenance.file_num	Specify the number of maintenance log files. Specify a value in the range of 1 through 16.	2
6	Size of the maintenance log	com.cosminexus.jaxrs.logger.runtime.maintenance.file_size	Specify the size of the maintenance log. Specify a value in the range of 4096 through 16777216 (unit: bytes).	16777216
7	Output level of the exception log	com.cosminexus.jaxrs.logger.runtime.exception.level	Specify the output level of the exception log. Specify ERROR, WARN, INFO, DEBUG, or NONE. For details on the output contents corresponding to each specified value, see *39.3.4 Log importance and the output conditions*.	INFO
8	Number of exception log files	com.cosminexus.jaxrs.logger.runtime.exception.file_num	Specify the number of exception log files. Specify a value in the range of 1 through 16.	2
9	Size of the exception log	com.cosminexus.jaxrs.logger.runtime.exception.file_size	Specify the size of the exception log. Specify a value in the range of 4096 through 16777216 (unit: Bytes).	16777216
10	Output level of the communication log (for Web resource) #1	com.cosminexus.jaxrs.logger.runtime.transport.server.level	Specify the output level of the communication log for the Web resource.#2 When you specify NONE, the communication log is not output. When you specify ALL, the sent and received HTTP header and the entity body are always output to the communication log. When you specify HEADER, the HTTP header of the sent and received messages is always output to the communication log.	NONE
11	Output level of the communication log (for the client in the Web resource)	com.cosminexus.jaxrs.logger.runtime.transport.client.level	Specify the output level of the communication log for the client in the Web resource.#2 If you specify NONE, the communication log is not output. If you specify ALL, the sent and received HTTP header and entity body are always output to the communication log. When you specify HEADER, the HTTP header of the sent and received messages is always output to the communication log.	NONE

No.	Settings	Key name	Values to be specified	Default value
12	Number of the communication log	com.cosminexus.jaxrs.logger.runtime.transport.file_num	Specify the number of the communication log. Specify a value in the range of 1 through 16.	2
13	Size of the communication log	com.cosminexus.jaxrs.logger.runtime.transport.size	Specify the size of the communication log. Specify a value in the range of 4096 through 1677721 (unit: Bytes).	16777216
14	Character encoding of the communication log	com.cosminexus.jaxrs.logger.runtime.transport.encoding	Specify the character encoding of the communication log. For the details on the encodings supported by J2SE 6.0, see *the J2SE 6.0 documentation*. When you specify DEFAULT, the default platform encoding is used.	DEFAULT
15	Deterring the WADL publication #1	com.sun.jersey.config.feature.DisableWADL	Specify whether to prevent the WADL publication. #2 When you specify true, the WADL publication is deterred. When you specify false, the WADL publication is not deterred.	false
16	Enabling the JSON to POJO mapping#1, #3	com.sun.jersey.api.json.POJOMappingFeature	Specify whether to enable the JSON POJO mapping.#2 If you specify true, the JSON POJO mapping is enabled. If you specify false, the JSON POJO mapping is disabled.	false
17	Automatic redirect#3	com.sun.jersey.client.property.followRedirects	Set whether the common definition file must automatically follow the HTTP redirect (a request of 300 level HTTP status codes). If you specify true, the file automatically follows the HTTP redirect. If you specify false, the file does not automatically follow the HTTP redirect. The operation when you specify the automatic redirect feature is same as the operation when you specify the value for automatic redirect in the argument and call the feature by using the setInstanceFollowRedirects method of the HttpURL Connection class of the Java SE.#2	true
18	The client socket connection timeout#3	com.sun.jersey.client.property.connectTimeout	Specify the client socket connection timeout. The timeout specified by using this property is effective when calling a Web resource. Specify a value in the range of 0 through 2147483647 (in milliseconds). If you specify 0, the connection timeout is not performed. If the settings of the OS to TCP connection are changed, the value set in the OS might be given priority.	0
19	The client socket read timeout #3	com.sun.jersey.client.property.readTimeout	Specify the client socket read timeout. The timeout specified by using this property is effective when calling a Web resource.	0

No.	Settings	Key name	Values to be specified	Default value
19	The client socket read timeout #3	com.sun.jersey.client.property.readTimeout	Specify a value within the range of 0 through 2147483647 (in milliseconds). If you specify 0, the connection does not time out. If the settings of the OS to TCP connection are changed, the value set in the OS might be given priority.	0
20	Chunked transfer encoding#3	com.sun.jersey.client.property.chunkedEncodingSize	Specify whether to use the chunked transfer encoding by specifying a value within the range of 0 through 2147483647 (unit: milliseconds). If you specify 0, the default value is applied. The operation when you specify the chunked transfer encoding feature is same as that when you specify the value for chunked transfer encoding in an argument and call the feature by using the setChunkedStreamingMode method of the HttpURL Connection class of the Java SE.	4096
21	Buffering of the response entity when an exception is thrown#3	com.sun.jersey.client.property.bufferResponseEntityOnException	Specify whether to automatically buffer a response entity and close the streaming if the UniformInterfaceException exception is thrown and the response contains the entity.#2 If you specify true, the entity of the HTTP response is automatically buffered and the streaming closes. If you specify false, the streaming of the entity of the HTTP response does not close automatically.	true

#1

On the Web resource side, the value specified in the servlet initialization parameter is given a priority over the value specified in the property.

#2

The property value is not case sensitive. If you specify an invalid value, the property value is used as the default value.

#3

The client side gives priority to the value specified in the property over the value specified in the client APIs.

(4) When changing the settings

Stop all the J2EE servers that are not using the process-wise definition file, and then change the settings of the common definition file. For details on the process-wise definition file, see *10.1.3 Setting up a process-wise definition file*.

To change the log-related settings, save the log as and when required and then make the necessary changes.

13.1.3 Setting up a process-wise definition file (JAX-RS)

Create a process-wise definition file when you want to make a process-specific definition.

You can use any name for the name of the process-wise definition file and the name of the storage destination directory. Enable the process-wise definitions by specifying the storage destination path in the system property. An example of specifying a process-wise definition file is as follows:

```
com.cosminexus.jaxrs.confpath=d:/tmp/example.properties
```

You must first stop the targeted processes (J2EE applications or Java applications) and then change the definition of the process-wise definition file.

To change the log related definitions, save the log as and when required and then make the necessary changes.

13.2 Operations of the JAX-RS engine

This section describes the operations and the support range of the Cosminexus JAX-RS engine.

The JAX-RS engine serves as the communication infrastructure of RESTful Web Services (Web resources). The operations of the engine on aWeb resource client and Web resource are as follows:

- JAX-RS engine on the Web resource client

  The JAX-RS engine receives Java objects from the Web resource client through client APIs for RESTful Web Services and generates an HTTP request. The engine sends the generated HTTP request to the called Web resource, receives an HTTP response from the Web resource, generates a Java object, and returns the generated Java object to the Web resource client.
- JAX-RS engine on the Web resource

The JAX-RS engine receives HTTP requests, locates the target resource class (discovery), and calls the method corresponding to the request (dispatch). When implementing discovery and dispatch, the engine executes the required injection, based on the annotation of the resource class. Also, the JAX-RS engine generates and returns HTTP responses from the target resource classes.

13.2.1 Discovery and dispatch

This subsection describes the discovery and dispatch operations of a Web resource, and also the mapping of the faults and the exception class.

(1) Discovery

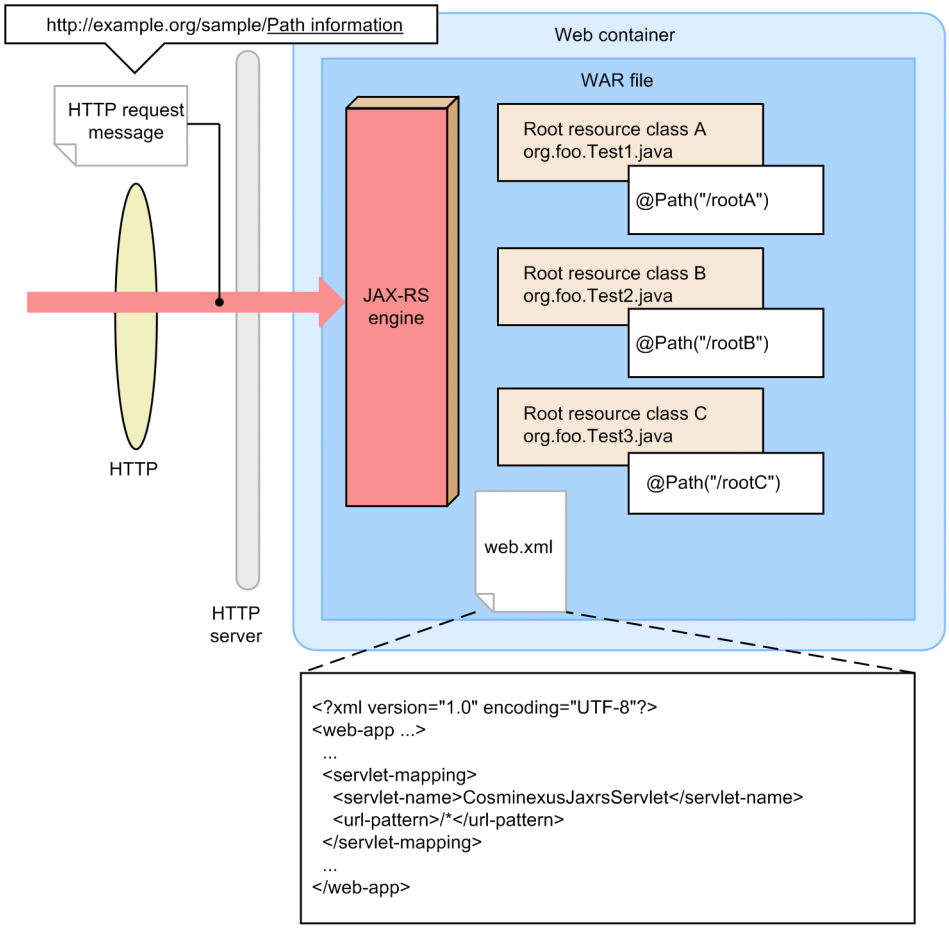
In the discovery operation, a request class is mapped based on the URL requested in an HTTP request. The example here explains the details on the mapping done when the following URL is requested:

`http://example.org/sample/rootA`

If you consider *sample* as the context root, "/rootA" (the underlined part) after the context root indicates the path information. The resource class is mapped based on this path information.

The following is an example of mapping with the path information. Note that the following example uses only the root resource class and not the sub-resource class as the resource class. Also, each root resource class consists of only a resource method.

Figure 13-1: Discovery



Among the deployed root resource classes, the JAX-RS engine calls a root resource class in which the value of the `Path` annotation is equal to the path information.

The correspondence between the path information and the called root resource class is as follows:

- When rootA is the path information:
  The root resource class A (org.foo.Test1.java) is called.
- When rootB is the path information:
  The root resource class B (org.foo.Test2.jav) is called.
- When rootC is the path information:
  The root resource class C (org.foo.Test3.java) is called.

Note that if a sub-resource class has been deployed and the `Path` annotation value of the corresponding sub-resource locator is equal to the path information, that sub-resource class is called.

(2) Dispatching HTTP messages

The JAX-RS engine calls and executes a resource method depending on the content of the received HTTP message (`HTTP Content-Type` header) and the acceptable media type (`HTTP Accept` header). The dispatch operation is executed based on the media types specified in the `Consumes` and `Produces` annotations.

Note that if the resource class contains the sub-resource method, the discovery and dispatch operations are concurrently determined.

13.3 Publishing the meta data

The JAX-RS engine automatically publishes WADL (meta data) of RESTful Web Services (Web resource).

This section describes the points to be noted when using the WADL publications.

(1) Conditions for publishing the meta data

The following two methods are used for publishing WADL of a Web resource:

- `HTTP GET` method
- `HTTP OPTIONS` method

The following table describes the conditions for publishing WADL of a Web resource by using the `HTTP GET` method. WADL is published when the JAX-RS engine receives an HTTP request that fulfills all the conditions described in the following table.

Table 13-2: HTTP request (GET) required for publishing the meta data of a Web resource

No.	Item		Condition
1	HTTP method		`GET` method
2	URL	Schema	http or https
3		Host name (:Port number)	Host name (and port number) having a Web resource that requests the publication of the meta data
4		Context path	Context path of a Web application, which includes a Web resource that requests the publication of the meta data
5		The path following the context path	`application.wadl`

For example, consider the context root of a Web application (WAR file) containing a Web resource to be a *sample*, and the name of the host on which a Web application is published is *example.org*. In this case the URL will be . Here, all the Web resources specified in the `com.sun.jersey.config.property.packages` initialization parameter (`init-param` element) of `web.xml` are included in the WADL to be published. If the `com.sun.jersey.config.property.packages` initialization parameter (`init-param` element) is not coded in `web.xml`, all the Web resources in the WAR file are included. For details on the `com.sun.jersey.config.property.packages` initialization parameter (`init-param` element) of `web.xml`, see *11.2 Creating web.xml*.

An example of WADL of a Web resource is as follows:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<application xmlns="http://research.sun.com/wadl/2006/10">
 <doc xmlns:jersey="http://jersey.dev.java.net/" jersey:generatedBy="Cosminexus JAX-
RS 09-00"/>
 <resources base="http://example.org/sample/">
  <resource path="root">
   <method name="GET" id="resourceMethod">
    <response>
     <representation mediaType="*/*"/>
    </response>
   </method>
   <method name="POST" id="postHandler">
    <request>
     <representation mediaType="*/*">
      <param xmlns:xs="http://www.w3.org/2001/XMLSchema" type="xs:string"
       style="query" name="form"/>
     </representation>
    </request>
    <response>
     <representation mediaType="text/html"/>
    </response>
   </method>
   <resource path="subresourceMethod">
    <method name="GET" id="subResourceMethod">
     <request>
```

```
      <param xmlns:xs="http://www.w3.org/2001/XMLSchema" default="1"
       type="xs:string" style="matrix" name="matrix"/>
      <param xmlns:xs="http://www.w3.org/2001/XMLSchema" type="xs:string"
       name="cookie"/>
     </request>
     <response>
      <representation mediaType="*/*"/>
     </response>
    </method>
   </resource>
   <resource path="exception">
    <method name="GET" id="subResourceMethodThrowingException">
     <response>
      <representation mediaType="*/*"/>
     </response>
    </method>
   </resource>
   <resource path="subresourceLocator/{id}">
    <param xmlns:xs="http://www.w3.org/2001/XMLSchema" type="xs:string"
     style="template" name="id"/>
    <method name="GET" id="getHandlerForSubResource">
     <request>
      <param xmlns:xs="http://www.w3.org/2001/XMLSchema" type="xs:string"
       style="header" name="HeaderKey"/>
     </request>
     <response>
      <representation mediaType="*/*"/>
     </response>
    </method>
   </resource>
  </resource>
 </resources>
</application>
```

The following table describes the conditions for publishing the meta data of a Web resource by using the `HTTP OPTIONS` method. The meta data is published when the JAX-RS engine on the Web Service machine receives an HTTP request that fulfills all the conditions described in the following table.

Table 13-3: HTTP request (OPTIONS) required for publishing the meta data of a Web resource

No.	Item		Condition
1	HTTP method		`OPTIONS` method
2	URL	Schema	http
3		Host name (: Port number)	Host name (and port number) having a Web resource that requests the publication of the meta data
4		Context path	Context path of a Web application which includes a Web resource that requests the publication of the meta data
5		Web resource path	Value of the `Path` annotation used in a Web resource

For example, consider the value of the `Path` annotation of the Web resource A to be "*/rootA*", the host name to be "*example.org*", and the context path to be "*sample*". In this case, the URL to publish the meta data will be: `http://example.org/sample/rootA`.

In such a case, only the requested Web resources are included in WADL to be published.

Tip

If a method of the target Web resource contains the `OPTIONS` annotation and can process the `HTTP OPTIONS` request, then the JAX-RS engine does not publish WADL and calls a method that processes the `HTTP OPTIONS` request of a Web resource. If the Web resource fails to process the `HTTP OPTIONS` request, the JAX-RS engine automatically generates WADL.

An example of WADL in which the `HTTP OPTIONS` method is used is as follows.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<application xmlns="http://research.sun.com/wadl/2006/10">
 <doc xmlns:jersey="http://jersey.dev.java.net/" jersey:generatedBy="Cosminexus JAX-
RS 09-00"/>
```

```
  <resources base="http://example.org/sample/">
   <resource path="/root">
    <method name="GET" id="resourceMethod">
     <response>
      <representation mediaType="*/*"/>
     </response>
    </method>
    <method name="POST" id="postHandler">
     <request>
      <representation mediaType="*/*">
       <param xmlns:xs="http://www.w3.org/2001/XMLSchema" type="xs:string"
        style="query" name="form"/>
      </representation>
     </request>
     <response>
      <representation mediaType="text/html"/>
     </response>
    </method>
    <resource path="subresourceMethod">
     <method name="GET" id="subResourceMethod">
      <request>
       <param xmlns:xs="http://www.w3.org/2001/XMLSchema" default="1"
        type="xs:string" style="matrix" name="matrix"/>
       <param xmlns:xs="http://www.w3.org/2001/XMLSchema" type="xs:string"
        name="cookie"/>
      </request>
      <response>
       <representation mediaType="*/*"/>
      </response>
     </method>
    </resource>
    <resource path="exception">
     <method name="GET" id="subResourceMethodThrowingException">
      <response>
       <representation mediaType="*/*"/>
      </response>
     </method>
    </resource>
    <resource path="subresourceLocator/{id}">
     <param xmlns:xs="http://www.w3.org/2001/XMLSchema" type="xs:string"
      style="template" name="id"/>
     <method name="GET" id="getHandlerForSubResource">
      <request>
       <param xmlns:xs="http://www.w3.org/2001/XMLSchema" type="xs:string"
        style="header" name="HeaderKey"/>
      </request>
      <response>
       <representation mediaType="*/*"/>
      </response>
     </method>
    </resource>
   </resource>
  </resources>
 </application>
```

You can choose whether to publish WADL by specifying `true` or `false` in the `com.sun.jersey.config.feature.DisableWADL` property. Specify this property in:

- `cjrconf.properties` file
- Servlet initialization parameter of `web.xml`

If the property is specified in both of the above mentioned locations, a priority is given to the servlet initialization parameter.

The following table describes the existence of the WADL publication.

Table 13-4: Existence of WADL publication

No.	Servlet initialization parameter	Value	Operations of the JAX-RS engine
1	com.sun.jersey.config.feature.DisableWADL	`true`	Does not publish WADL. The value is not case sensitive.
		`false`	Publishes WADL. The value is not case sensitive.

No.	Servlet initialization parameter	Value	Operations of the JAX-RS engine
1	com.sun.jersey.config.feature.DisableWADL	A value other than `true` or `false`	Publishes WADL by using the settings mentioned in the common definition file (`cjrconf.properties`) and ignores the value specified in `web.xml`. Checks if the default value `false` has been specified and then publishes WADL.

13.4 Connecting through a proxy server

You can connect a Web resource client to a Web resource present in an external environment through a proxy server.

This section describes the settings of the properties required for connecting to an external environment through a proxy server.

(1) Specifying values in a property

To access a Web resource through a proxy server, specify the properties of JavaVM and information of the proxy server. The following table describes the JavaVM properties and contents to be specified for connecting through a proxy server. For details on the system properties of JavaVM, see the *JavaVM documentation*.

Table 13-5: The JavaVM properties for connecting through a proxy server

No.	JavaVM property	Specified content	Without SSL	With SSL
1	`http.proxyHost`	Specify the host name or the IP address of a proxy server. If you do not specify any value, connection with the proxy server is not established.	Y	N
2	`http.proxyPort`	Specify the port number of a proxy server. When `http.proxyHost` is appropriately specified, port number `80` of the host specified in `http.proxyHost` is accessed if you do not specify any value in `http.proxyPort`. When you do not specify any value in `http.proxyHost`, connection with the proxy server is not established even if `http.proxyPort` is specified.	S	N
3	`https.proxyHost`	Specify the host name or the IP address of a proxy server to be used in connections by the SSL protocol#. You must specify the host name or the IP address to use a proxy server in connections by the SSL protocol. Note that if you do not specify the host name or the IP address, connection with the proxy server is not established.	N	Y
4	`https.proxyPort`	Specify the port number of a proxy server to be used in connections by the SSL protocol#. Note that when `https.proxyHost` is appropriately specified, the port number `443` of the host specified in `https.proxyHost` is accessed, if you do not specify any value in `https.proxyPort`. If you do not specify any value in `https.proxyHost`, connection with the proxy server is not established even if `https.proxyPort` is specified.	N	S
5	`http.nonProxyHosts`	Specify, as and when required, the host names that are not required for using a proxy server. When connecting with the host specified by using this property, connection is not established through the proxy server specified in `http.proxyHost` or `https.proxyHost`. When specifying multiple hosts, use the separator (`|`). You cannot use any character (such as space) other than (`|`) for separating the two host names.	S	S

Legend:

Y: Specifying the property is mandatory.

S: Specify the property as and when required.

N: Need not specify the property.

#

For connections by the SSL protocol, see *10.11 SSL protocol Connections*.

(2) How to set the properties

For details on setting the system properties of JavaVM, see *10.10(2) How to set the properties*.

(3) Connecting through a non-anonymous proxy server

When accessing a Web resource through a non-anonymous proxy server, we recommend that you establish a connection by using the `java.net.Authenticator` class of the J2SE 6.0 standard in the Web resource client. For details, see the *J2SE 6 documentation*.

For examples of establishing a connection by using the `java.net.Authenticator` class, see *10.10(3) When not using properties specific to the JAX-WS functionality*.

13.5 Connecting with an SSL protocol

You can connect a Web resource client to an SSL protocol compliant Web resource.

To access a Web resource by using the SSL protocol, specify values in the properties supported by the JDK and the SSL protocol information. The following table lists and describes the properties used in the connection by the SSL protocol and the contents to be specified.

Table 13-6: Properties used in connections by the SSL protocol

No.	Property	Specified content
1	`javax.net.ssl.trustStore`	Specifies a truststore.
2	`javax.net.ssl.trustStorePassword`	Specifies the password of the truststore.

Specify these properties as and when required. Note that if you do not specify the truststore, the default value of *JDK-installation-directory*`/lib/security/jssecacerts` is used.

For details on the JDK properties, see the *JDK documentation*. For details on setting the properties and notes, see *10.11(2) How to specify the properties* and *10.11(3) Notes on validating a host name*.

13.6 Connecting by basic authentication

You can connect a Web resource client to a Web resource that is supported in the basic authentication.

This section describes the implementation required to connect by basic authentication.

(1) Implementation required to connect by basic authentication

To access a Web resource by basic authentication, implement a process that adds the required HTTP headers. An implementation example when using client APIs for RESTful Web Services is as follows.

```
// The user ID and password for basic authentication
String username = ...
String password = ...
// Generates Client objects
Client client = Client.create();
// Generate an HTTP request having the Authorization HTTP header
// and post to the Web resource
client.resource( "http://example.org/helloworld" )
.header( HttpHeaders.AUTHORIZATION,
"Basic " + encode(username + ":" + password))
.post( String.class, "Some Request" );
...
String encode( String value ){
String encoded;
// Encode the value parameter by using the Base64 algorithm value
// and set the result in the encoded parameter
...
return encoded;
}
```

13.7 Troubleshooting

This section describes the specific points to be noted when troubleshooting the Web resources and Web resource clients. For details on general precautions, types of errors, troubleshooting, and logs, see *39. Troubleshooting*.

For details, see *17. Web Resources and Providers* and *25. Support range of client APIs for RESTful Web Services.*.

13.7.1 Checking the syntax when initializing a Web resource (KDJJ20003-W and KDJJ10006-E)

The JAX-RS engine initializes the root resource class and the exception mapping provider when a Web resource included in a Web application (WAR file) is called for the first time. The corresponding sub-resource locator initializes the sub-resource class.

This sub-section describes the cases of error detection during the syntax check executed at the time of initialization.

(1) If a fatal error occurs

If a fatal syntax error which leads to the non-completion of the initialization process is detected, such as in the case when no `public` constructor is declared in a resource class, an error message (`KDJJ10006-E`#) is output to the log. An HTTP response with the HTTP status code 500 is returned to the root resource class and the exception mapping provider, and `java.lang.RuntimeException` is thrown to the J2EE server. For details on the exceptions thrown to the J2EE server, see *13.7.4 Throwing exceptions to the J2EE server*. In the sub-resource class, `java.lang.RuntimeException` is thrown, which can be handled with the exception mapping provider.

#

Other error messages are also output if there is any additional information to report.

The error message `KDJJ10006-E` includes a list of detailed information (sub messages). Sub messages contain detailed error information for the respective syntax errors. Eliminate the syntax errors by referencing the sub-messages. The following are the notes on the error message `KDJJ10006-E`:

- If multiple syntax errors are detected, you must eliminate all the errors.
- If a Web application (WAR file) contains multiple root resource classes, sub-resource classes, and exception mapping providers and if `KDJJ10006-E` is output for any one of these, you cannot use any other root resource class, sub-resource class, and exception mapping provider. Check all the root resource classes, sub-resource classes, and exception mapping providers included in a Web application (WAR file) until the error message `KDJJ10006-E` ceases to be output.

(2) If a minor error occurs

If a minor syntax error is detected and no critical error is detected, such as in the case when `void` is the return value of the resource method that receives the HTTP GET request, the warning message (`KDJJ20003-W`) is output to the log. If also a critical error is detected, the error message (`KDJJ10006-E`) is output to the log.

For details on the cases where the error message KDJJ10006-E is output to the log, see *13.7.1(1). If a fatal error occurs*

If the warning message `KDJJ20003-W` is output to the log, the process is continued, and initialization is successfully performed except for the methods for which the minor syntax errors were detected. Revise the syntax of the methods for which the minor syntax errors were detected until the warning message `KDJJ20003-W` ceases to be output.

13.7.2 Errors detected in the received HTTP request processing

If an HTTP request cannot be processed, such as in the case when a resource method to dispatch is unavailable, `javax.ws.rs.WebApplicationException` or another exception that can be handled with an exception mapping provider is thrown. An error message is also output if there is additional information to be reported.

13.7.3 Exceptions that can be handled with an exception mapping provider

If an exception that can be handled with an exception mapping provider is thrown, you can customize the mapping of the exception with an HTTP response to be sent by creating an exception mapping provider and appropriately including the provider in a Web application (WAR file). For details on the operations of an exception mapping provider, see *17.1.8 Exception handling*.

13.7.4 Throwing exceptions to the J2EE server

The JAX-RS engine of Web resources supports a servlet-based mechanism laid down in the JAX-RS specifications. Accordingly, as and when required, the exceptions that are thrown to the JAX-RS engine are also thrown to the J2EE server to which a Web application (WAR files) that includes a Web resource is deployed.

If an error occurs during the operation of a Web resource, check the log files of the JAX-RS functionality and the J2EE server.

13.7.5 Exception (KDJJ18888) that occurs when using client APIs

If the `ClientHandlerException` exception or `UniformInterfaceException` exception is thrown while using client APIs, the error message KDJJ18888-E is output to the log.

14 Commands

You can use the `cjwsimport` command and the `apt` command to generate SEI and JavaBean classes required for SOAP Web Services. Also, if you are using the `cjwsgen` command, you can generate WSDL from the already compiled Java source.

This chapter describes how to use the `cjwsimport` command, `apt` command, and `cjwsgen` command.

14.1 cjwsimport command

The `cjwsimport` command follows the mapping rules defined in the JAX-WS 2.2 specifications and performs the mapping of a WSDL file to Java. If you execute the `cjwsimport` command, the Java source required for implementing Web Services and Web Service clients will be generated and compiled from WSDL files.

This subsection describes the format and the specified contents when executing the `cjwsimport` command.

(1) Format

The *specification format of the cjwsimport command* is as follows:

```
cjwsimport [Options] URL-or-file-path-of-the-WSDL-file
```

Example of specification

- Specifying the local WSDL file using the relative path (`wsdl/input.wsdl`)
  ```
  cjwsimport -d generated wsdl/input.wsdl
  ```
- Specifying the local WSDL file using the absolute path (`/tmp/wsdl/input.wsdl`)
  ```
  cjwsimport -d generated /tmp/wsdl/input.wsdl
  ```
- Specifying the local WSDL file using a URL (`file:///tmp/wsdl/input.wsdl`)
  ```
  cjwsimport -d generated file:///tmp/wsdl/input.wsdl
  ```
- Specifying the remote WSDL file using a URL (`http://example.com:8080/fromjava/test?wsdl`)
  ```
  cjwsimport -d generated http://example.com:8080/fromjava/test?wsdl
  ```

Notes for executing the cjwsimport command

You cannot assume a current directory with the characters such as `%`, `&`, and `^` to execute the `cjwsimport` command. If you assume such a current directory, the operations will not be guaranteed.

Notes for specifying a WSDL file

- Specify one file path (relative path or absolute path) for a WSDL file or the URL to WSDL in the argument. If a file other than the WSDL file is specified, an error message will be output in the standard error output as well as log and the processing will end (KDJW51200-E).
- When you specify WSDL using the file path, do not use the characters `&` and `^` in the file path. If such characters are used, the operations might not function properly. Also, if you are specifying a file path containing a blank space, enclose the entire file path within double quotation marks (""). If you do not enclose the entire file path within double quotation marks (""), the operations are not guaranteed.
- When you specify a WSDL using a URL, use the characters defined in the RFC 2396 specifications and the strings complying with the rules in the RFC 2396 specifications. Additionally, in accordance with the rules of the RFC 2396 specifications, you must perform the percent encoding for the character strings with the UTF-8 format. If you do not follow the rules of the RFC 2396 specifications or if you specify the characters and character strings that are not encoded, the operations might not function properly.
- You cannot specify a WSDL by using the `jar` protocol. If you specify a WSDL by using the `jar` protocol, the operation is not guaranteed.
- When you specify the WSDL file using the file path or a URL to WSDL, specify the correct file path or URL. If you specify a wrong WSDL file path or WSDL URL and if the WSDL file cannot be found, an error message will be output in the standard error output and log and the processing will end (KDJW51180-E or KDJW51189-E).
- The extension of the WSDL file specified in the argument is optional (you might specify an extension other than `.wsdl`).
- The WSDL file is not case sensitive.
- The length of the string to be specified is not limited. However, an error occurs when you exceed the limit set for the OS.
- If you specify a WSDL file for which you do not have access permission, a JDK error occurs and the processing ends.

(2) List of options

You can specify the options listed in the following table as the options for the `cjwsimport` command:

Table 14-1: List of options for the cjwsimport command

Options	Set items	Specified contents
`-d` *directory*	Output destination directory of the class file	Specifies the output destination directory of the compiled class file (`*.class`). For details about the values that you can specify, see *Notes for specifying the -d option and the -s option* described outside the table. If this option is not specified, the class file will be output in the current directory.
`-keep`	None	Specified for generating the source file (`*.java`).
`-s` *directory*	Output destination directory of the source file	Specifies the output destination directory to output the source file (`*.java`). For details about the values that you can specify, see *Notes for specifying the -d option and the -s option* described outside the table. The output destination directory changes depending on the specification of the `-d` option and the `-s` option. For details about the specification of options and the output destination, see *Table 14-3*.
`-verbose`	None	Specified to output the detailed processing passage when the command is executed.
`-b` *path*	Path of the external binding file	Specifies the path of the external binding file when the external binding file is used. For details about the values that you can specify, see *Notes for specifying the -b option* described outside the table.
`-p` *package*	Package name of the Java code	Specifies the package name of the Java source. When you specify this option, the customization of the package name specified in the binding declaration and the generation algorithm for the default package name defined in standard specifications is overwritten.
`-generateService`	Generating the Web Service Implementation Class	Specified when the Web Service Implementation Class (skeleton class) is generated. For details about the generated files, see *14.1(4) Generated files*.
`-help`	None	Specified for displaying help. When you specify this option, the specification of all options except `-version` is ignored; help is displayed and terminated.
`-version`	None	Specified for displaying the version information. When you specify this option, the specification of other options is ignored; the version information is displayed and terminated.
`-wsdllocation`	Values to be specified in the `wsdlLocation` element of the `javax.xml.ws.WebServiceClient` annotation	Specifies the values to be specified in the `wsdlLocation` element of the `javax.xml.ws.WebServiceClient` annotation.
`-catalog` *file*	Catalog file path	Specifies the values for using the catalog functionality. For details on the values you can specify, see *Notes for specifying the -catalog option*given below this table.

Creating a directory for generating a file

When you execute the `cjwsimport` command, a directory corresponding to the package name of the generated file will be created in the specified output destination directory and the file will be output in that directory.

The following is an example of command specification and output destination (for a request bean) when `http://example.com/sample` is coded in the namespace URI of `Type referenced by the wrapper element of the request message` for the WSDL file (`test.wsdl`):

- Example of command specification
  `cjwsimport -d ./output -keep input/test.wsdl`
- Output destination (Request bean)
  The compiled class file and source file of the request bean will be output in the following directory:
  `./output/com/example/sample/`

Take the following precautions for the specified values of the options such as the values that can be specified for the option and the operations when the specification is omitted:

Common precautions for options

The following precautions are common to all the options:

- The specification order for the options and arguments is optional. The specification order for each option is also optional.
- Make sure that you specify the argument for the options with arguments. If the argument is not specified, an error message will output in the standard error output and log and the processing will end (KDJW51001-E).
- When you specify the same option more than once, except for the `-catalog` option, the option specified at the end will be enabled.
- The specification of the option is case sensitive.
- The length of the string to be specified is not limited. However, an error occurs when you exceed the limit set for the OS.
- Do not use the character strings containing `&` and `^` in the option specifying the path. If you use such character strings, the operations are not guaranteed. Also, if you specify a file path containing a blank space, enclose the entire file path within double quotation marks (""). If you do not enclose the entire file path within double quotation marks (""), the operations are not guaranteed.
- If you specify an option that cannot be specified, an error message will output in the standard error output and log and the processing will end (KDJW51001-E).

Notes for specifying the -d option and the -s option

Take the following precautions for the values specified for the `-d` option and the `-s` option:

- The specified value is not case sensitive.
- If the specified output destination directory does not exist, an error message will output in the standard error output and log and the processing will end (KDJW51181-E).
- If the file is specified in the wrong output destination directory, an error message will output in the standard error output and log and the processing will end (KDJW51182-E).
- If you specify a WSDL file without access permission, a JDK error occurs and the processing ends.

Specification of -d, -s, and -keep options and file output destination

The output destination directory of the compiled class file and source file differs depending on the specification of the `-d` option, `-s` option, and the `-keep` option.

The following table describes whether the option is specified and the output destination directory of the compiled class file:

Table 14-2: Presence or absence of option specification and the output destination directory of the compiled class file

Presence or absence of option specification			Presence or absence of source file output and output destination directory
`-d`	`-s`	`-keep`	
Y	--	--	Output in the directory specified in the `-d` option.
N	--	--	Output in the current directory.

Legend:

Y: Indicates that the option is specified.

N: Indicates that the option is not specified.

--: Indicates that the presence or absence of option specification does not affect the output destination directory.

The following table describes the presence or absence of option specification and the output destination directory of the source file:

Table 14-3: Presence or absence of option specification and the output destination directory of the source file

Presence or absence of option specification			Presence or absence of source file output and output destination directory
-d	-s	-keep	
Y	Y	--	Output in the directory specified in the -s option.
Y	N	Y	Output in the directory specified in the -d option.
Y	N	N	Not output.
N	Y	--	Output in the directory specified in the -s option.
N	N	Y	Output in the current directory.
N	N	N	Not output.

Legend:

Y: Indicates that the option is specified.

N: Indicates that the option is not specified.

--: Indicates that the presence or absence of the option specification does not affect the output destination directory.

Notes for specifying the -b option

Take the following precautions for the values specified for the -b option:

- The specified value is not case sensitive.
- Specify the external binding file using the file path. The operations might not function properly if the file is specified using the URL format.
- If the specified external binding file does not exist, an error message is output in the standard error output and log and the processing ends (KDJW51184-E).
- If a file other than the external binding file is specified, the operations might not function properly.
- If a directory is specified by mistake, an error message is output in the standard error output and log and the processing ends (KDJW51185-E).
- If you specify a WSDL file without access permission, a JDK error occurs and the processing ends.
- Specify the file same as the WSDL file to be customized by the specified external binding file and the WSDL file to be customized specified in the argument of the cjwsimport command. If the files are not the same, an error message is output in the standard error output and log and the processing ends (KDJW51190-E).
- The specification of the extension of the external binding file (.wsdl) is optional.

Notes for specifying the -p option

Take the following precautions for the values specified in the -p option:

- Specify the package name using one-byte alphanumeric characters (0 to 9, A to Z, a to z), underscore (_), and period (.). If other characters are used, the operations might not function properly.
- For each label (xxx, yyy, zzz) that is separated using periods (.) such as xxx.yyy.zzz, specify strings that can be used with Java identifiers. If you specify characters that cannot be used, an error message will output in the standard error output and log and the processing will end.

Notes for specifying the -wsdllocation option

- Specify the -wsdllocation option with the URI format. If you specify this option in any other format, the operations are not guaranteed.

- You cannot specify the `jar` protocol for the values to be specified in the `-wsdllocation` option. If you specify the `jar` protocol, the operation is not guaranteed. When you specify the `jar` protocol as a URL for pointing to the WSDL, use the constructor that acquires the URL in a parameter while generating a service class.

Notes for specifying the -catalog option

Take the following precautions for specifying the values in the `-catalog` option:

- If you specify the `-catalog` option more than once, the operation is not guaranteed.
- The value to be specified is not case-sensitive.
- Specify the file path of the catalog file. Follow the `java.io.File` class specifications for the specification format. If you specify the URL of the catalog file, the operation is not guaranteed.
- Specify the file path for the catalog file by using single-byte alphanumeric characters (0 to 9, A to Z and, a to z), spaces, periods (.), underscores (_), colons (:), slashes (/), and \. If you use any other characters to specify the file path, the operation is not guaranteed.
- You can specify any name for the catalog file.
- If the specified catalog file does not exist, a warning message (KDJW51219-W) is output to the standard error output and log. The catalog functionality is disabled and the processing continues.
- If you specify a file other than the catalog file, the operation is not guaranteed.
- If a directory is specified by mistake, a warning message (KDJW51220-W) is output to the standard error output and log. The catalog functionality is disabled and the processing continues.
- If the specified catalog file is coded in an unsupported syntax, a warning message (KDJW51221-W) is output to the standard error output and log. The catalog functionality is disabled and the processing continues.
- If a catalog file that does not require access permission is specified, a JDK error occurs, the catalog functionality is disabled and the processing continues.

Operations, when a value is specified for an ignored value

Among all the values specified for the command, if a value is specified for values that are ignored even if specified, an error might occur in the processing performed later.

(3) Specifying the WSIMPORT_OPTS environment variable

If you specify an option character string in the `WSIMPORT_OPTS` environment variable, you can add the option with the java command that starts the `cjwsimport` command. By default, nothing is specified in the `WSIMPORT_OPTS` environment variable, so specify any character string, as and when required.

For example, by using the `WSIMPORT_OPTS` environment variable, you can specify the JDK system properties required for the SSL communication to enable execution of the `cjwsimport` command for WSDLs that you could not access before unless you used HTTPS. The following is an example.

```
> set WSIMPORT_OPTS=-Djavax.net.ssl.trustStore=trust-store -
Djavax.net.ssl.trustStorePassword=trust-store-password
> "%COSMINEXUS_HOME%\jaxws\bin\cjwsimport.bat" https://securehost:443/fromwsdl/
TestJaxWsService?wsdl
```

(4) Generated files

The following table describes the files generated when the `cjwsimport` command is executed:

Table 14-4: List of files generated for the cjwsimport command

No.	Java code	Contents	Output by the -generateService option	
			Specified	Not specified
1	Request bean class	This is the JavaBean class corresponding to the type referenced by the wrapper element of the request message.	Y	Y

No.	Java code	Contents	Output by the -generateService option	
			Specified	Not specified
1	Request bean class	If the specified WSDL file does not have a wrapper style, the request bean class is not output.	Y	Y
2	Response bean class	This is the JavaBean class corresponding to the type referenced by the wrapper element of the response message. If the specified WSDL file does not have a wrapper style, the response bean class is not output.	Y	Y
3	Fault bean class	This is the JavaBean class corresponding to the type referenced by the fault message. If a fault is not defined in the specified WSDL file, the fault bean class is not output.	Y	Y
4	Wrapper exception class	This is the wrapper exception class of the fault bean class.	Y	Y
5	ObjectFactory class	This is the factory class of the JAXB 2.2 specifications.	Y	Y
6	Other classes in the JAXB 2.2 specifications	These are other classes in the JAXB 2.2specifications. These are the Java interfaces and Java classes corresponding to the various elements, types coded in the syntax of the XML Schema specification.	Y	Y
7	package-info class	This is the package information.	Y	Y
8	SEI	This is the service end point interface.	Y	Y
9	Skeleton class	This is the skeleton class implementing SEI. Implementation is added in this class.	Y	--
10	Service class	This class is used for accessing the Web Service.	--	Y

Legend:

Y: Indicates that the file is output.

--: Indicates that the file is not output.

Precautions for generating a file

If a file with the same name as the skeleton class exists in the output destination directory of the generated file, a warning message is output in the standard error output and log (KDJW51203-W). At this time, the processing continues without the output of the skeleton class.

Also, apart from the Web Service Implementation Class, if a file with the same name exists in the output destination directory of the generated file, the file will be overwritten.

Output of the Javadoc header information

In the generated file, the Cosminexus-related information is output in the header information as Javadoc.

(5) Operations during processing

When you execute the `cjwsimport` command, the message (KDJW50001-I) indicating command execution will output in the standard output and log and the Java source generation and compilation processing will be performed. The following is a description of the processing:

Java source generation processing

When the Java source generation starts, a message indicating the start of generation is output in the standard output and log (KDJW50004-I). If an attempt to generate the Java source fails, the error message that causes the error is output in the standard error output and log (KDJW50005-E).

Java source compilation processing

When the Java source compilation starts, a message indicating the start of compilation is output in the standard output and log (KDJW50006-I). If an attempt to compile the Java source fails, the error message that causes the error is output in the standard error output and log (KDJW50007-E).

(6) End code

The following is the *end code of the cjwsimport command*:

`0`: Normal termination

If an error due to which processing cannot continue is not detected in the middle of the processing, a message indicating that the processing has finished is displayed in the standard output and log and the processing ends (KDJW50002-I).

`1`: Abnormal termination

If even a single error due to which processing cannot continue is detected in the middle of processing, an error message is displayed in the standard output and log and the processing ends (KDJW50003-E). For details about the action for abnormal termination, see *14.1(7) Action for abnormal termination*.

If a negligible error is detected due to which processing cannot continue is detected in the middle of the processing, a warning message is output and the processing continues.

Note that sometimes a log is not output depending on the specified output level (importance). For details on the settings for the output level of a log, see *10.1.2 Settings for the common definition file*.

(7) Action for abnormal termination

If an abnormal termination occurs during the execution of the `cjwsimport` command, an error message will output and the processing ends. In this case, remove the cause of the error that is output and re-execute the `cjwsimport` command.

Even for multiple errors, the error detected first is displayed. In this case, execute the `cjwsimport` command repeatedly and remove each of the causes of the displayed errors.

14.2 apt command

The apt command is a JDK command that interprets annotations, generates additional Java code, and compiles by including the basic Java code. You use the apt command for the development of a Web Service starting from SEI. The apt command interprets the annotation (coded in compliance with the JAX-WS 2.2 specifications) coded in the Web Service Implementation Class, adds the required JavaBean class and generates the Java code.

For details about the format, argument, and options of the apt command, see *JDK documentation*. This subsection describes the contents that are not defined in the *JDK documentation* and the precautions for executing the command.

(1) Web Service Implementation Class and SEI specified in the argument

In the *argument of the apt command*, specify a Web Service Implementation Class and SEI (if SEI is referenced) respectively. If you specify two or more Web Service Implementation Classes, an error message will output in the log and an error message is returned to the apt command (KDJW61002-E). However, when only SEI is specified and the Web Service Implementation Class does not exist, a warning message is output in the standard error output and log and the processing continues (KDJW61001-W). In this case, the JavaBean class is not generated.

For details about the notes related to SEIs and Web Service Implementation Classes other than the above, see the *16.1 Default mapping of Java to WSDL* and *16.2 Customized mapping of Java to WSDL* sections.

The following warning is displayed in the apt command if EJB Web Service Implementation Class with the javax.ejb.Stateless annotation is specified in the argument.

```
Warning: annotation type without processor: javax.ejb.Stateless
```

(2) Required options

To execute the apt command, specifying the -classpath option, -J-Dcosminexus.home option and -factory option is mandatory. The values of each option are as follows:

The -classpath option

- *Cosminexus-installation-directory*/jaxws/lib/cjjaxws.jar
- *Cosminexus-installation-directory*/jaxp/lib/csmjaxb.jar
- *Cosminexus-installation-directory*/jaxp/lib/csmjaxp.jar
- *Cosminexus-installation-directory*/jaxp/lib/csmstax.jar
- *Cosminexus-installation-directory*/CC/client/lib/j2ee-javax.jar
- *Cosminexus-installation-directory*/CC/client/lib/HiEJBClientStatic.jar

For Windows x86

- *HNTRLib2-installation-directory*#1/classes/hntrlib2j.jar
- *HNTRLib2-installation-directory*#1/classes/hntrlibMj.jar

For Windows x64

- *HNTRLib2-installation-directory*#1/classes/hntrlib2j64.jar
- *HNTRLib2-installation-directory*#1/classes/hntrlibMj64.jar

The -J-Dcosminexus.home option

Cosminexus-installation-directory

The -factory option#2

com.cosminexus.istack.ws.AnnotationProcessorFactoryImpl

#1

HNTRLib2-installation-directory specifies the execution results of the following command:

- For Windows x86

```
> "%COSMINEXUS_HOME%\common\bin\gethntr2conf.exe" HNTR2INSTDIR
```

- For Windows x64

```
> "%COSMINEXUS_HOME%\common\bin\gethntr2conf64.exe" HNTR2INSTDIR
```

#2

If you execute the `apt` command without specifying the `-factory` option, the `JavaBean` class is generated by the JDK and not by the JAX-WS functionality of Cosminexus.

(3) Generated files

The following table describes the files generated when the `apt` command is executed:

Table 14-5: List of files generated for the apt command

No.	Java code	Contents
1	Request bean class	This is the JavaBean class for the request message. Output for the wrapper style.
2	Response bean class	This is the JavaBean class for the response message. Output for the wrapper style.
3	Fault bean class	This is the JavaBean class corresponding to the fault. Output when the wrapper exception class is defined in the specified Java code and when the fault bean does not exist.

Creating a directory for generating a file

When you execute the `apt` command, a directory corresponding to the package name of the generated file is created in the specified output destination directory and the file is output in that directory.

The following is an example of specification and output destination:

- Example of command specification

  ```
  apt -d ./output -s ./output/ -sourcepath . com/example/test.java
  ```

- Output destination

  The source files other than the JavaBean class and the compiled class file are output in the following directory:

  ```
  ./output/com/example/
  ```

  However, if the Java code specified in the command has the JavaBean class, the source file of the JavaBean class and its compiled class file are output in the `jaxws` sub-package of SEI package (excluding the case when the package name is customized using annotations).

  ```
  ./output/com/example/jaxws/
  ```

In the annotation processor provided by the Cosminexus JAX-WS functionality, if the output destination directory of the Java code does not exist or is not a directory and is invalid, an error message is output in the log and an error is notified to the `apt` command (KDJW61003-E).

The output destination directory of the compiled class file is the value of the `-d` option specified in the argument of the `apt` command. When you do not specify the `-d` option, the output destination directory is the value specified in the `-s` option. When you do not specify both the `-d` option and `-s` option, the current directory serves as the output destination directory.

The output directory of the source file is the value of the `-s` option specified in the `apt` command argument. When the `-s` option is not specified, the current directory is the output directory.

The `apt` command processes the generated source file by recursive interpretation, and hence the output destination directory of a source file might be treated as the input directory of the source file. Therefore, note that when a source file is scheduled to the output destination directory, the file might be treated as an input of the `apt` command. We recommend that you do not schedule a source file to the output destination directory.

Output of the Javadoc header information

In the generated file, the Cosminexus-related information is output in the header information as `Javadoc`.

(4) Operations when a negligible error that allows processing to continue is detected

If a negligible error that allows processing to continue is detected in the middle of the processing, a warning message is output and the processing continues.

Note that sometimes log is not output depending on the specified output level (importance). For details on the settings for the output level of a log, see *10.1.2 Settings for the common definition file*.

(5) Action for abnormal termination

If an error occurs during the execution of the `apt` command, an error message is output and the processing ends.

If an error message is output, remove the cause of the error that is output and re-execute the `apt` command. Remove the causes of each of the displayed errors and repeatedly execute the `apt` command until the command terminates normally. If the file is already generated, delete the files generated before the `apt` command is executed.

Note that sometimes log is not output depending on the specified output level (importance). For details on the settings for the output level of a log, see *10.1.2 Settings for the common definition file*.

14.3 cjwsgen command

The `cjwsgen` command generates the Java code (JavaBean classes) and resource files (WSDL and XSD) that are required to deploy Web Services, based on the class files of the Service Implementation Class. Note that a Service Implementation Class also includes the classes and SEIs that are referenced by the Service Implementation Class.

This section describes the format and specification contents for executing the `cjwsgen` command.

(1) Format

The following is the specification format of the `cjwsgen` command:

```
cjwsgen [Options] Fully qualified name of the Service Implementation Class
```

Specification examples

- Checking the WSDL before deploying:
  `cjwsgen -wsdl -cp . com.example.UserInfoImpl`
- Generating the Java code and resource files of a Web Service (SOAP 1.2) from an existing Web Service (SOAP 1.1):
  `cjwsgen -soap 1.2 -cp . com.example.UserInfoImpl`
- Generating the Java code and resource files of a Web Service (with service name `MyService`) from an existing Web Service:
  `cjwsgen -servicename {http://example.com/}MyService -cp . com.example.UserInfoImpl`
- Generating the Java code and resource files of a Web Service (with port name `MyServicePort`) from an existing Web Service:
  `cjwsgen -portname {http://example.com/}MyServicePort -cp . com.example.UserInfoImpl`

Notes for executing the cjwsgen command

You cannot assume a current directory with the characters such as `&`, and `^` to execute the `cjwsgen` command. If you assume such a current directory and execute the command, the operations will not be guaranteed.

Check that source files other than the source files generated by the `cjwsgen` command do not exist under the following directories:

- When you output a source file: the output destination directory of the source file
- When you do not output a source file : the work directory used by the `cjwsgen` command

For details on the output destination and the work directory of a source file, see *Combination of the -d, -s, and -keep options and the output destination directory and the work directory*.

Notes for specifying the Service Implementation Class

- Deploy the source files of the Service Implementation Class in a directory different from the class files. If you deploy the source files in the same directory as the class files, an error might occur.
- Specify the class files of the Service Implementation Class with their fully qualified names in the arguments.
- If you do not specify the arguments, an error message will output to the standard error output and logs, and the processing will end (KDJW71023-E).
- If you specify other than a class file in the arguments, an error message will output to the standard error output and logs, and the processing will end (KDJW71026-E).
- If you specify other than the Service Implementation Class in the arguments, an error message will output to the standard error output and logs, and the processing will end (KDJW71025-E).
- If the specified Service Implementation Class is not found, an error message will output to the standard error output and logs, and the processing will end (KDJW71026-E).
- If you do not have the access privileges for the specified Service Implementation Class, an error message will output to the standard error output and logs, and the processing will end (KDJW71026-E).

- If you specify a Service Implementation Class in which the `javax.jws.WebService` annotation is not specified, an error message will output to the standard error output and logs, and the processing will end (KDJW71029-E).
- If you specify multiple classes containing the Service Implementation Class in the arguments, a warning message will output, and the processing will continue (KDJW71027-W). To confirm which Service Implementation Class was handled effectively, see the warning message.
- Do not specify the Service Implementation Class acting as an inner class in an argument. If you specify, the operation is not guaranteed.
- For details about the notes related to SEIs and Web Service Implementation Classes other than the above, see the *16.1 Default mapping of Java to WSDL* and *16.2 Customized mapping of Java to WSDL* sections.

(2) List of options

You can specify the options listed in the following table for the `cjwsgen` command:

Table 14-6: List of options of the cjwsgen command

Option	Set item	Specification contents
`-d` *directory*	Path of the output destination directory of already compiled class file	Specifies the output destination directory of the already compiled class file (`*.class`). For details about the values that you can specify, see *Notes for specifying the -d, -s, and -r options* described outside the table. If you specify any other option, a file other than the already compiled class file might be output to the directory specified in this option. For details, see *Combination of the -d, -s, and -keep options and the output destination directory* and the work directory and *Combinations and output destination directory of the -d, -r, -soap, -servicename, -portname, -soap12binding, and -wsdl options* described outside the table.
`-s` *directory*	Path of the output destination directory of the source file	Specifies the output destination directory, when you output the source file (`*.java`).
`-r` *directory*	Path of the output destination directory of the resource files	Specifies the output destination directory when you output the resource files (`*.wsdl` and `*.xsd`).
`-keep`	None	Specifies whether or not to maintain the source file (`*.java`).
`-wsdl`	None	Specifies whether or not to output the resource files (`*.wsdl` and `*.xsd`).
`-soap` *version*	Version of SOAP binding	Specifies the version of SOAP binding used by the Web Service during communication.
`-servicename` *service-name*	Service name	Specifies the service name after making changes.
`-portname` *port-name*	Port name	Specifies the port name after making changes.
`-soap12binding` *soap-spec*	Value indicating the URL set in the `transport` attribute of the `soap12:binding` element	Specifies the URL to be set in the `transport` attribute of the `soap12:binding` element (child element of the `wsdl:binding` element) for SOAP1.2.
`-classpath` *class-path*	Class path that includes the Service Implementation Class	Specifies the class path that includes the Service Implementation Class specified in the argument.
`-cp` *class-path*		
`-verbose`	None	Specify this option to output the detailed processing progress, when executing a command.
`-help`	None	Specify this option to display **Help**.

Option	Set item	Specification contents
-help	None	If you specify this option, all options except -version are ignored, **Help** is displayed, and the processing terminates.
-version	None	Specify this option to display the version information. If you specify this option, the other options are ignored, the version information is displayed, and the processing terminates.

The following are the notes on the specification values of options, such as the values that you can specify in an option and the operation executed when you omit an option:

Notes common to options

The notes common to all options are as follows:

- The specification order of options and arguments is optional. The specification order of each option is also optional.
- Always specify the arguments for options with arguments. If you do not specify the arguments, an error message will output to the standard error output and logs, and the processing will end (KDJW71001-E).
- If you specify the same option more than once, the option specified last will be valid. However, if you specify an invalid value for an option, an error will occur.
- The specification of an option is case sensitive.
- The length of the character string to be specified is not limited. However, an error occurs when you exceed the limit specified for the OS.
- If you specify a file path containing a blank space in the option, enclose the entire path within double quotation marks (""). If you do not enclose the entire path within double quotation marks (""), the operations are not guaranteed.
- If you specify an option that you cannot specify, an error message will output to the standard error output and logs, and the processing will end (KDJW71001-E).

Notes for specifying the -d, -s, and -r options

The following are the notes related to the specification values of the -d option, -s option, and -r option:

- The specification value is not case sensitive.
- If the specified output destination directory does not exist, an error message will output to the standard error output and logs, and the processing will end (KDJW71002-E).
- If you specify the file in the wrong output destination directory, an error message will output to the standard error output and logs, while **Help** will output to the standard output, and the processing will end (KDJW71003-E).
- Do not use the following characters in the path of the output destination directory to be specified in the -d option. If you use the following characters, the operations are not guaranteed:
  `# % & ^ ; ` { } [ ]`
- Do not use the following characters in the path of the output destination directory to be specified in the -r option. If you use the following characters, the operations are not guaranteed:
  `% & ^ ` { } [ ]`
- Do not use the following characters in the path of the output destination directory to be specified in the -s option. If you use the following characters, the operations are not guaranteed:
  `& ^`
- If you specify a directory for which you do not have the access privileges, a JDK error will occur and the processing will end.

Notes for specifying the -soap option

The following are the notes related to the specified values of the -soap option:

- You can specify only 1.1 or 1.2 as the version of the SOAP binding. If you specify any other value, an error message will output to the standard error output and logs, while **Help** is output to the standard output, and the processing will end (KDJW71004-E).

- If you have specified a value in both the `-soap` option and the `javax.xml.ws.BindingType` annotation, priority is given to the value of the `-soap` option.
- If you omit the `-soap` option, the specification values of the `javax.xml.ws.BindingType` annotation become enabled.
- If you specify 1.1 in the `-soap` option when the specification value of `javax.xml.ws.BindingType` is the `SOAP 1.2 over HTTP` binding, an error message will output to the standard error output and logs, and the processing will end (KDJW71007-E).

Notes for specifying the -servicename option

Code the `-servicename` option in the `QName` format. The following is a coding example of the `-servicename` option:

```
{Namespace URI}Service name
```

Namespace

- Enclose the namespace URI within a parentheses ({ }). If you omit the namespace or do not enclose within the parentheses ({ }), an error message will output to the standard error output and logs, and the processing will end (KDJW71009-E).
- If you do not close the parentheses, an error message will output to the standard error output and logs, and the processing will end (KDJW71008-E).
- If you do not enter a value within the parentheses ({ }), an error message will output to the standard error output and logs, and the processing will end (KDJW71008-E).
- Specify single-byte alphanumeric characters for the namespace URI. If you specify other than the single-byte alphanumeric characters, the operations are not guaranteed.

Protocol

- Only a domain name beginning with `http://` or `urn:` is valid as the namespace URI to be coded in the `-servicename` option. The namespace URIs beginning with `https://` and `file://` are handled as invalid. If you specify a namespace URI beginning with other than `http://` or `urn:`, an error message will output to the standard error output and logs, and the processing will end (KDJW71011-E).
- You cannot specify a relative path for the namespace URI to be coded in the `-servicename` option. If you specify a relative path for the namespace URI, an error message will output to the standard error output and logs, and the processing will end (KDJW71012-E).

Information that you cannot specify

You cannot code query strings, anchors, port numbers, user names, and passwords in the namespace URI to be coded in the `-servicename` option. If you specify this information in the namespace URI, an error message will output to the standard error output and logs, and the processing will end (KDJW71013-E).

Character strings that you can code

In a segment demarcated with the delimiters such as a forward slash (/) or a period (.), you can code a character string that satisfies all the conditions described in the following table:

Table 14-7: Conditions for character strings that you can code in a namespace (When you specify the **-servicename option)**

No.	Condition	Examples of invalid character strings	Operation when an invalid character string is specified

1	Character strings using only single-byte alphanumeric characters (0 to 9, A to Z, and a to z)	`http://hitachi.com/`
		`http://133.145.224.19/`
		`http://[1080:2C14;D30:BA04:275:806:270C:418A]/`
			The operation is not guaranteed (no error message is displayed).

2	Character strings containing other than reserved terminology of Java	`http://hitachi.com/abstract`	The operation is not guaranteed.
3	Character strings that do not begin with a numeric character	`http://1hitachi.com`	

Service name

- If you omit the service name, an error message will output to the standard error output and logs, and the processing will end (KDJW71014-E).
- Specify single-byte alphanumeric characters and underscores for a service name. If you specify other than the single-byte alphanumeric characters and underscores, the operations are not guaranteed.
- We recommend that you specify a service name in accordance with the identifier naming rules stipulated in Java. If the specified service name is not in accordance with the identifier naming rules stipulated in Java, a compilation error occurs when you execute the `cjwsimport` command to develop the Web Service client.

Notes for specifying the -portname option

Code the `-portname` option in `QName` format. The following is a coding example of the `-portname` option:

```
{Namespace URI}Port name
```

Namespace

- Enclose the namespace URI within a parenthesis ({ }). If you omit the namespace or do not enclose within parentheses ({ }), an error message will output to the standard error output and logs, and the processing will end (KDJW71016-E).
- If you do not close the parentheses, an error message will output to the standard error output and logs, and the processing will end (KDJW71015-E).
- If you do not enter a value within the parentheses ({ }), an error message will output to the standard error output and logs, and the processing will end (KDJW71015-E).
- Specify a namespace URI same as that of the `service` element of the WSDL file. If you specify a namespace URI different from that of the `service` element of the WSDL file, an error message will output to the standard error output and logs, and the processing will end (KDJW71022-E).
- Specify single-byte alphanumeric characters for the namespace URI. If you specify other than the single-byte alphanumeric characters, the operations are not guaranteed.

Protocol

- Only a domain name beginning with `http://` or `urn:` is valid as the namespace URI to be coded in the `-portname` option. The namespace URIs beginning with `https://` and `file://` are handled as invalid. If you specify a namespace beginning with other than `http://` or `urn:`, an error message will output to the standard error output and logs, and the processing will end (KDJW71018-E).
- You cannot specify a relative path for the namespace URI to be coded in the `-portname` option. If you specify a relative path for the namespace URI, an error message will output to the standard error output and logs, and the processing will end (KDJW71019-E).

Information that you cannot specify

You cannot code query strings, anchors, port numbers, user names, and passwords in the namespace URI to be coded in the `-portname` option. If you specify this information in the namespace URI, an error message will output to the standard error output and logs, and the processing will end (KDJW71020-E).

Character strings that you can code

In a segment demarcated with delimiters such as a forward slash (/) or a period (.), you can code a character string that satisfies all the conditions described in the following table:

Table 14-8: Conditions for character strings that you can code in a namespace (When specifying the **-portname option)**

No.	Condition	Examples of invalid character strings	Operation when an invalid character string is specified

1	Character strings using only single-byte alphanumeric characters (0 to 9, A to Z, and a to z)	`http://hitachi.com/`
		`http://133.145.224.19/`
		`http://[1080:2C14;D30:BA04:275:806:270C:418A]/`
			The operation is not guaranteed (no error message is displayed).

No.	Condition	Examples of invalid character strings	Operation when an invalid character string is specified
2	Character strings containing other than reserved terminology of Java	`http://hitachi.com/abstract`	The operation is not guaranteed.
3	Character strings that do not begin with a numeric character	`http://1hitachi.com`	

Port name

- If you omit the port name, an error message will output to the standard error output and logs, and the processing will end (KDJW71021-E).
- Specify single-byte alphanumeric characters and underscores for the port name. If you specify other than the single-byte alphanumeric characters and underscores, the operations are not guaranteed.

Notes for specifying `-soap12binding`

You can specify only "DEFAULT" or "WSI_BP20_TRANSPORT" for the `-soap12binding` option. If you specify any other value, an error message will be output to the standard error output and logs, while **Help** will be output to the standard output, and the processing will end (KDJW71030-E).

The following table describes the relationship between the `-soap12binding` option and the `transport` attribute value of the `soap12:binding` element of WSDL generated by the `cjwsgen` command.

Table 14-9: Relationship between the `-soap12binding` option and `transport` attribute value

No.	Option specification	Option specified value	Set value of the `transport` attribute
1	Not specified	None	`http://www.w3.org/2003/05/soap/bindings/HTTP/`
2	Specified	DEFAULT	
3		WSI_BP20_TRANSPORT	`http://schemas.xmlsoap.org/soap/http`

Notes for specifying the -classpath and -cp options

The following are the notes related to the specified values of the `-classpath` option and the `-cp` option:

- If you omit the options, the environment variable `CLASSPATH` is used as the class path. If you specify the options, the environment variable `CLASSPATH` is ignored.
- The value you specify in the environment variable `CLASSPATH` is used as it is, and therefore, even if the value includes a blank space, you need not enclose the value within double quotation marks (""). If you specify a value in such a format, in which the value is enclosed within double quotation marks (""), the operations are not guaranteed.
- If you omit the options and also do not specify the environment variable `CLASSPATH`, the current directory will be used as the class path.
- You can specify either a relative path or an absolute path for the class path.
- You can also specify a JAR file as the class path.
- When specifying more than one class path, code a semicolon (;) between two class paths.
- Do not use the following characters in the class path to be specified. If you use the characters, the operations are not guaranteed.

  `# % & ^`
- If the specified class path is invalid, an error message will output to the standard error output and logs, and the processing will end (KDJW71026-E).

Combination of the -d, -s, and -keep options and the output destination directory and the work directory

If the `-d` option is specified, the already compiled class file (`*.class`) will output to the directory specified in the `-d` option, and if the `-d` option is not specified, the already compiled class file (`*.class`) will output in the current directory.

The following table describes the combination of options and provides the information about whether to output source file (`.java`) or if the source file is output, the output destination directory, and the work directory used by

the cjwsgen command. Note that if you want to use only the resource file, it is all right if the source file is not output.

Table 14-10: Availability of the source file output and the output destination directory

Whether the option is specified			Whether to output the source file The output destination directory and work directory
-d	-s	-keep	
Y	Y	Y	The directory specified by the -s option is used as the work directory and the source file is output.
Y	Y	N	
Y	N	Y	The directory specified by the -d option is used as the work directory and the source file is output.
Y	N	N	The directory specified by the -d option is used as the work directory. The source file is not output.
N	Y	Y	The directory specified by the -s option is used as the work directory and the source file is output.
N	Y	N	
N	N	Y	The current directory is used as the work directory and the source file is output.
N	N	N	The current directory is used as the work directory. The source file is not output.

Legends:

Y: Indicates that the option is specified.

N: Indicates that the option is not specified.

Combinations and output destination directory of the -d, -r, -soap, -servicename, -portname, -soap12binding, and -wsdl options

The following table describes the combination of options and provides the information about whether to output resource files (*.wsdl and *.xsd)) or if the resource files are output, the location of the output destination directory:

Table 14-11: Availability of resource file output and the output destination directory

Availability of option and specification				Output or not	Output destination
-d option	-r option	-soap, -servicename, -portname, and -soap12binding option	-wsdl option		
Specified	Specified	Specified	Specified	Y	Directory specified in the -r option
			Not specified		
		Not specified	Specified		
			Not specified		
	Not specified	Specified	Specified	Y	Directory specified in the -d option
			Not specified		
		Not specified	Specified	Y	Directory specified in the -d option
			Not specified	N	--
Not specified	Specified	Specified	Specified	Y	Directory specified in the -r option
			Not specified		
		Not specified	Specified		

Availability of option and specification				Output or not	Output destination
-d option	-r option	-soap, -servicename, -portname, and -soap12binding option	-wsdl option		
Not specified	Specified	Not specified	Not specified	Y	Directory specified in the -r option
	Not specified	Specified	Specified	Y	Current directory
			Not specified		
		Not specified	Specified	Y	Current directory
			Not specified	N	--

Legend:

--: Not applicable because the resource files are not output.

Y: Indicates that the resource files are output.

N: Indicates that the resource files are not output.

(3) Generated files

The following table lists and describes the files generated when you execute the `cjwsgen` command:

Table 14-12: List of files generated when executing the cjwsgen command

No.	Generated file	Contents
1	Request bean class	This is the JavaBeans class for the request message. This file is output when the generated Service Implementation Class has the `wrapper` format.
2	Response bean class	This is the JavaBeans class for the response message. This file is output when the generated Service Implementation Class has the `wrapper` format.
3	Fault bean class	This is the JavaBeans class corresponding to the fault. This file is output when the wrapper exception class is specified in the Java code that you have specified, and the fault bean does not exist.
4	WSDL	This is a WSDL file.
5	XSD	This is an XML schema definition file.

Creating a directory for file generation

If you execute the `cjwsgen` command, a directory corresponding to the package name of the generated files is created in the specified output destination directory, and files are output to this directory.

The following are the specification example and the output destination:

- Command specification example
  ```
  cjwsgen -d ./output -s ./output -keep -cp . com.example.UserInfoImpl
  ```
- Output destination
  If a JavaBean class exists in the class file of the Service Implementation Class specified in the `cjwsgen` command, the source file and the already compiled class file of the JavaBean class are output to the following `jaxws` sub package (except in cases where the package name is customized using annotations):
  ```
  ./output/com/example/jaxws/
  ```

Also, the files generated under the resource file are output to the directory specified in the argument of the `cjwsgen` command. The following are the specification example and the output destination:

- Command specification example
  ```
  cjwsgen -r ./output -cp . com.example.UserInfoImpl
  ```
- Output destination

```
./output/
```

Notes for file generation

If a file with the same name exists in the output destination directory of the generated file, the file is overwritten.

Output of header information of Javadoc

In the generated file, the Cosminexus-related information is output in the header information as `Javadoc`.

(4) Relationship between the Input Service Implementation Class and output resource files

The following table describes the relationship between the Input Service Implementation Classes and output resource files:

Table 14-13: Relationship between the Input Service Implementation Class and output resource files

Input Service Implementation Class	Output resource			
	WSDL		XSD	
	Number of files	File name	Number of files#1	File name
Without SEI	1	`name` attribute value of the `wsdl:service` element	1 to N#2	`name` attribute value of the `wsdl:service` element + suffix (_schemaN)#2
With SEI (same namespace as the Service Implementation Class)	1	`name` attribute value of the `wsdl:service` element	1 to N#2	`name` attribute value of the `wsdl:service` element + suffix (_schemaN)#2
With SEI (different namespace from the Service Implementation Class)	2	• For an abstract WSDL file#3 `name` attribute value of the `wsdl:portType` element • For an implementation WSDL file#4 `name` attribute value of the `wsdl:service` element	1 to N#2	`name` attribute value of the `wsdl:portType` element + suffix (_schemaN)#2

#1

The file is generated, if the namespace of the schema is different.

#2

N is the number of namespaces of the schema. The upper-limit count of files that you can generate depends on the OS.

#3

Abstract WSDL indicates WSDL of '`wsdl:types` element, `wsdl:message` element, and `wsdl:portType` element'.

#4

Implementation WSDL indicates WSDL of '`wsdl:binding` element and `wsdl:service` element'.

(5) Operation during processing

If you execute the `cjwsgen` command, a message (KDJW70001-I) indicating the command execution will output to the standard output and log, and processes such as the generation and deletion of JavaBeans, and generation of WSDL and XSD will be executed. The following is the description about each process:

Process for generating JavaBeans

When the generation of JavaBeans starts, a message indicating the start of the generation process will output to the standard output and the log (KDJW70004-I). If an attempt to generate JavaBeans fails, an error message indicating the cause of the error will output to the standard error output and the log (KDJW70005-E).

Process for generating WSDL and XSD

When the generation of the resource files (WSDL and XSD) corresponding to the contents of the generated JavaBeans starts, a message indicating the start of the generation process will output to the standard output and log (KDJW70006-I). If an attempt to generate the resource files fails, an error message reporting the cause of the error will output to the standard error output and log (KDJW70007-E).

Process for deleting JavaBeans

Delete the source file of the generated JavaBeans. However, in some cases, the source file might not be deleted depending on the specification contents of the options. For details about the options, see the section *14.3(2) List of options*.

(6) End codes

The following are the *end codes of the cjwsgen command*:

`0`: Normal termination

Unless an error that cannot be allowed to continue during the processing is detected, a message indicating the termination with standard output and logs will be displayed, and the processing will end (KDJW70002-I).

`1`: Abnormal termination

- If a minor error that allows the continuation of the process is detected midway, a warning message will output, and the processing will continue.
- If an error that does not allow the continuation of the process is detected midway, a message indicating termination will be output to the standard output and log, and the processing will end (KDJW70003-E). For details about the action to be taken for the abnormal termination, see *14.3(7) Action for abnormal termination*.
- If multiple errors are detected, the error detected first will be displayed, and a message indicating termination will output to the standard output and log, and the processing will end.
- If you execute the `cjwsgen` command, the `apt` command is invoked. Therefore, error messages of the `apt` command might be output.
- The directories and files generated prior to the detection of the error are not deleted and are retained, even when the command terminates abnormally.

Note that depending on the output level (importance) that you have set, the log might not be output. For details on the settings of the output level of a log, see *10.1.2 Settings for the common definition file*.

(7) Action for abnormal termination

If the `cjwsgen` command terminates abnormally during the execution, an error message will output and the processing will end. In such cases, remove the cause of the error that is output and re-execute the `cjwsgen` command.

Even when multiple errors occur, the error that is detected first will be displayed. In such cases, repeatedly execute the `cjwsgen` command and remove the causes of the displayed error one by one.

Note that if the command terminates abnormally due to inaccuracy of the class file of the Service Implementation Class, you need to modify the Java sources that are the generated source of the class file, and then revise the compilation.

14.4 Notes on using a command line interface in Windows with enabled UAC

This section describes the notes on executing the `cjwsimport` command, `apt` command, and `cjwsgen` command when the OS is Windows and *UAC (User Account Control)* is enabled.

14.4.1 When the administrator uses a command line interface

When the administrator uses a command line interface, do not take any precaution that is to be taken during the installation.

If the administrator executes a command line interface after the installation, the administrator must start the Command Prompt. For details about how to start the Command Prompt by escalating privileges to the administrator, reference the OS documentation.

14.4.2 When a user other than the administrator uses a command line interface

This subsection describes the notes for using a command line interface by a user other than the administrator.

(1) Notes for installation

When a user other than the administrator uses a command line interface, install Cosminexus in the default installation directory. If you install Cosminexus in a directory other than the default installation directory, you must set up the access privileges in such a way so that even a user other than the administrator who is executing the command line interface can write in all the log output destination directories of the command line interface. The administrator sets the access privileges. For details about how to specify the access privileges, reference the OS documentation.

(2) Notes for command execution

The following are the notes for executing a command line interface by a user other than the administrator:

- For the current directory, specify a directory that is not protected by UAC.
- For the output destination of files generated according to the option settings, specify a directory that is not protected by UAC.
- If you have installed the Cosminexus server in the default installation directory, the log of the command line interface is redirected to the following directory:

  The corresponding directory under the `%LoadlAppData%\VirtualStore\Program Files` directory

  For details on redirecting, reference the OS documentation.

15 Mapping from WSDL to Java

When you execute the `cjwsimport` command, WSDL is mapped to the Java source in accordance with the JAX-WS 2. 2 specifications.

This chapter describes the default mapping and customized mapping of WSDL to Java.

15.1 Default mapping from WSDL to Java

When the `cjwsimport` command is executed, WSDL is mapped to the Java source. The following table describes the correspondence relationship in this case:

Table 15-1: List of mapping of WSDL to Java source

No.	WSDL	Java source	Reference
1	Namespace	Package name	*15.1.1*
2	Port type	SEI name	*15.1.2*
3	Operation	Method name	*15.1.3*
4	Part	Parameter and return value	*15.1.4*, *15.1.5*
5	Type	Parameter and return value	*15.1.6*
6	Fault	Exception class	*15.1.7*
7	Binding	`javax.jws.soap.SOAPBinding` annotation	*15.1.8*
8	Service	The `serviceName` attribute of the `javax.jws.WebService` annotation	*15.1.9*

15.1.1 Mapping a namespace to a package name

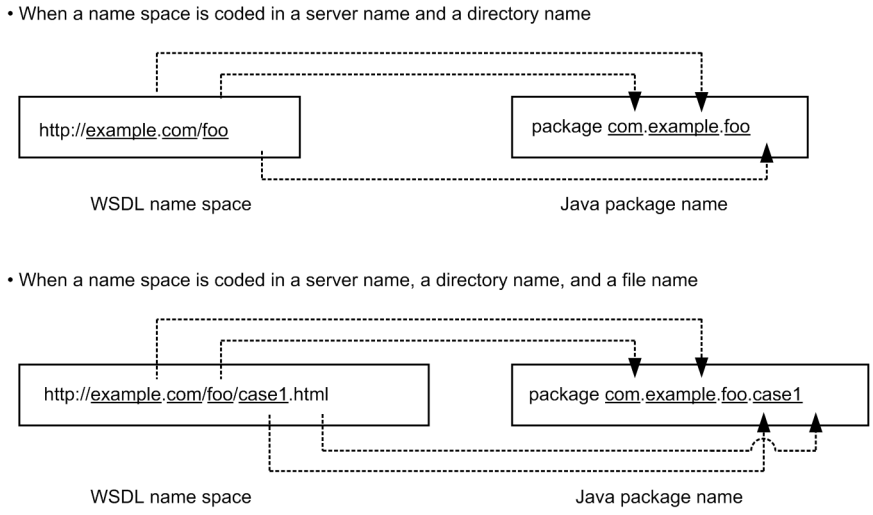
This subsection describes the mapping of a WSDL namespace (`targetNamespace` attribute of the `wsdl:definitions` element) to a package name.

(1) Mapping

The WSDL namespace and package name are mapped in accordance with the JAX-WS 2.2 specifications. For details, see the *JAX-WS 2.2 specifications*.

The following figure shows an example of mapping:

Figure 15-1: Example of mapping the namespace to the package name



(2) Conditions for the namespace

This point describes the conditions for the namespace coded in WSDL.

- **Protocol**

  Code the namespace using `http://` or `urn: protocols`. If you code the namespace using a protocol other than `http://` or `urn:` (such as `https://`, `file ://`), an error message is output in the standard error output and log and the processing ends (KDJW51002-E).

  Also, when the relative path is used for coding, an error message is output in the standard error output and log and the processing ends (KDJW51003-E).

- **Namespace coding format**

  The following formats cannot be coded in the namespace. If namespace is coded in the following formats, an error message is output in the standard error output and log and the processing ends (KDJW51004-E):

  - Null character string
  - Query string (example) `http://example.com/?a=b`
  - Anchor (example) `http://example.com/index.html#anchor`
  - Port number (example) `http://example.com:8080/`
  - User name/ password (example) `http://user:password@example.com`

- **Strings that can be coded**

  In a segment demarcated with delimiters such as a forward slash (`/`) or period (`.`), you can code a character string that satisfies all the conditions described in the following table. However, when customizing with the binding declaration, you can code the strings in accordance with the rules in the RFC 2396 specifications.

  Table 15-2: Conditions for strings that can be coded in the namespace

No.	Conditions	Examples of invalid strings	Operations when invalid strings are specified
1	Strings using only one-byte alphanumeric characters (0 to 9, A to Z, a to z)	`http://Hitachi.com`	The operation might not function properly (error message is not displayed).
2	Strings other than Java reserved words	`http://hitachi.com/abstract`	When mapping to the Java package name, an underscore (_) is added at the beginning of the Java reserved word. Example: `com.hitachi._abstract`
3	Strings that do not begin with numeric characters	`http://1hitachi.com`	When mapping to the Java package name, an underscore (_) is added at the beginning of the string that begins with a numeric character. Example: `com._1hitachi`

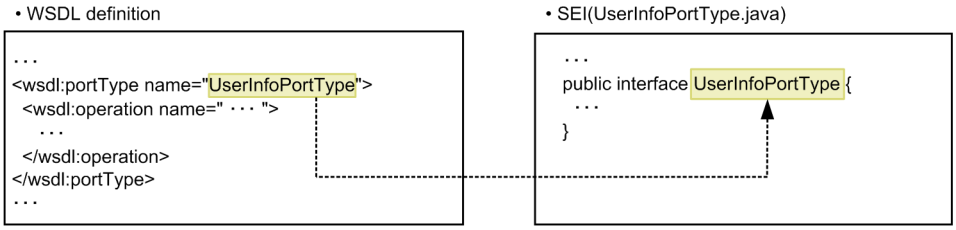
15.1.2 Mapping a port type to a SEI name

This subsection describes the mapping of a WSDL port type name (`name` attribute of the `wsdl:portType` element) to a SEI name.

(1) Mapping

The WSDL port type and SEI name are mapped in accordance with the JAX-WS2.2 specifications. The following figure shows an example of mapping:

Figure 15-2: Example of mapping the port type to the SEI name



During mapping, the first character of the WSDL port type name is converted into an upper case character. The following is an example of conversion:

Before conversion: `portTypeName`

After conversion: `PortTypeName`

(2) Conditions for the port type name

When specifying a port type name and a namespace for a WSDL, confirm that the SEI name, including the package name, is not changed to `javax.xml.ws.Provider`. Therefore, do not specify `Provider` or `provider` in the port type name. Also, do not specify `http://ws.xml.javax` in the namespace.

In the port type, you can code a string that fulfills all the conditions described in the following table. However, when customizing with the binding declaration, you can code the strings that can be used as the `xsd:NCName` type of the XML Schema specification.

Table 15-3: Conditions for strings that can be coded in the port type

No.	Conditions	Examples of invalid strings	Operations when invalid strings are specified
1	Strings using only one-byte alphanumeric characters (0 to 9, A to Z, a to z) and underscore (_)	`Hitachi_portType`	The operation might not function properly (error message is not displayed).
2	Strings that do not begin with numeric characters	`1User_portType`	An error message is output in the standard error output and log and the processing ends (KDJW51029-E).

(3) Number of port types coded

You can code 1 to 255 port types in WSDL. The following table describes the relationship between the number of port types coded and the operations:

Table 15-4: Relationship between the number of port types coded and the operations

No.	Elements	Number of coding	Operations when invalid strings are specified
1	`wsdl:portType`	0	An error message is output in the standard error output and log and the processing ends (KDJW51008-E).
2		1 to 255	Terminates normally.
3		256 or more	An error message is output in the standard error output and log and the processing ends (KDJW51008-E).

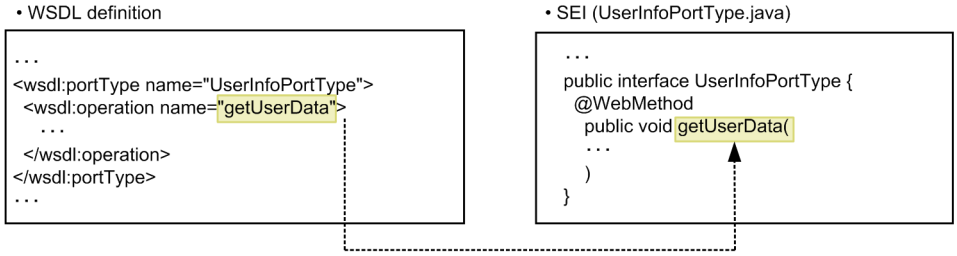
15.1.3 Mapping from an operation to a method name

This subsection describes the mapping of WSDL operations (`name` attribute of the `wsdl:operation` element) to Java method names.

(1) Mapping

The WSDL operations and Java method names are mapped in accordance with the JAX-WS 2.2 specifications. The following figure shows an example of mapping:

Figure 15-3: Example of mapping operations to method names



During the mapping, the first character of the WSDL operation name is converted into a lower case character. The prefixes `get` and `set` are not added. The following is an example of conversion:

Before conversion: `OperationName`

After conversion: `operationName`

(2) Conditions for the operation name

In the operation name, you can code a string that fulfills all the conditions described in the following table. However, when customizing with the binding declaration, you can code the strings that can be used as `xsd:NCName` type of the XML Schema specification.

Table 15-5: Conditions for strings that can be coded in the operation name

No.	Conditions	Examples of invalid strings	Operations when invalid strings are specified
1	Strings using only one-byte alphanumeric characters (0 to 9, A to Z, a to z) and underscore (_)	`Hitachi_operation`	The operation might not function properly (error message is not displayed).
2	Strings other than Java reserved words#	`abstract`	An error message is output in the standard error output and log and the processing ends (KDJW51007-E).
3	Strings that do not begin with numeric characters	`1User_operation`	An error message is output in the standard error output and log and the processing ends (KDJW51029-E).

#

You cannot code strings such as 'Abstract' where the first character of the Java reserved word is in upper case (since the first character is converted into a lower case character due to mapping).

(3) Number of operations and its child elements coded

You can code 1 to 255 operations for one port type in WSDL. Also, in the child element of the operation, you can code one `wsdl: input` element, zero or one `wsdl:output` element, and 0 to 255 `wsdl:fault` elements.

The following table describes the relationship between the number of operations coded and the operations:

Table 15-6: Relationship between the number of operations coded and the operations

No.	Elements	Number of coding	Operations when invalid strings are specified
1	`wsdl:operation`	0	An error message is output in the standard error output and log and the processing ends (KDJW51029-E).
2		1 to 255	Terminates normally.

No.	Elements	Number of coding	Operations when invalid strings are specified
3	`wsdl:operation`	256 or more	An error message is output in the standard error output and log and the processing ends (KDJW51029-E).

The following table describes the relationship between the number of child elements coded for the operation and the operations:

Table 15-7: Relationship between the number of child elements coded for the operation and the operations

No.	Elements	Number of coding	Operations when invalid strings are specified
1	`wsdl:input`	0	An error message is output in the standard error output and log and the processing ends (KDJW51029-E).
2		1	Terminates normally.
3		2 or more	An error message is output in the standard error output and log and the processing ends (KDJW51029-E).
4	`wsdl:output`	0	If zero `wsdl:fault` elements are coded, the `wsdl:output` element is mapped to a one-way operation and the process ends successfully. If one or more `wsdl:fault` elements are coded, an error message is output in the standard error output and log and the processing ends (KDJW51029-E).
5		1	The `wsdl:output` element is mapped to the Request-response operation and the process ends successfully.
6		2 or more	An error message is output in the standard error output and log and the processing ends (KDJW51029-E).
7	`wsdl:fault`	0 to 255	Terminates normally.
8		256 or more	An error message is output in the standard error output and log and the processing ends (KDJW51029-E).

15.1.4 Mapping a message part to a parameter and return value (For wrapper style)

This subsection describes the mapping of WSDL message parts (`wsdl:part` child element of the `wsdl:message` element) to parameters and return values of Java methods.

This subsection describes the mapping for the wrapper style.

- **Conditions for the wrapper style**

  A wrapper style that fulfills all the following conditions is handled as a wrapper style. A wrapper style that does not fulfill the conditions is handled as a non-wrapper style.

  - The input message referenced from the `soap:body` element of the WSDL operation includes only 1 part.

    If two or more parts are included, an error message is output in the standard error output and log and the processing ends (KDJW51019-E).
  - The output message (if present) referenced from the WSDL operation includes only one part.

    If two or more parts are included, an error message is output in the standard error output and log and the processing ends (KDJW51020-E).
  - The input message part references the global element where the local name is equal to the operation name.
  - The output message (if present) part references the global element.
  - The element type referenced from the input message and output message (if present) part is `xsd:complexType` defined in `xsd:sequence`.

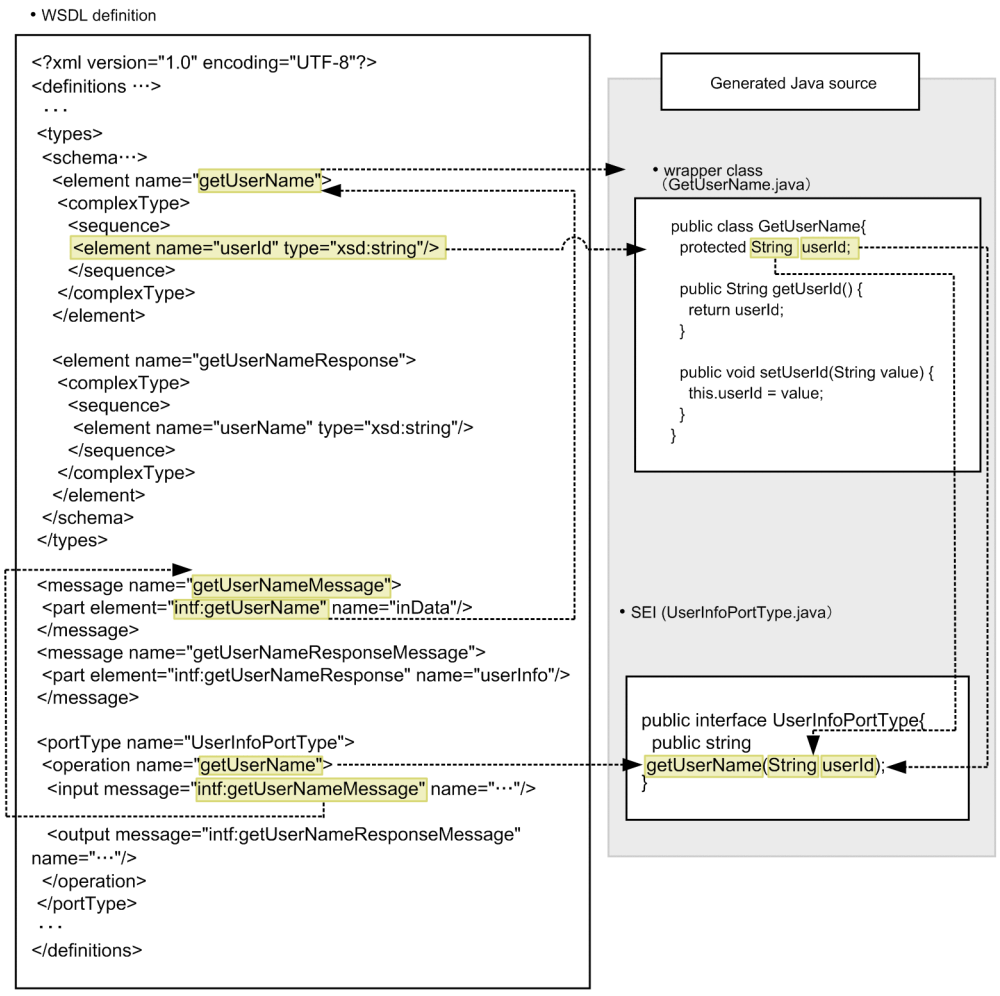
- The wrapper element only includes the child elements and does not include the other components such as the `xsd:any` element, the `xsd:anyAttribute` attribute, the `xsd:choise` element, the `substitutionGroup` attribute, or the `attribute` element.
- The wrapper element that is not `nillable`.

(1) Mapping

- **For mapping from a WSDL of request-response operations**

  The wrapper child element to be referenced from the message part of a WSDL (define one `wsdl:input` element and one `wsdl:output` element , and zero or more `wsdl:fault` elements) of the request-response operations and the return values and parameters of the Java method are mapped. The following figure shows the example of mapping.

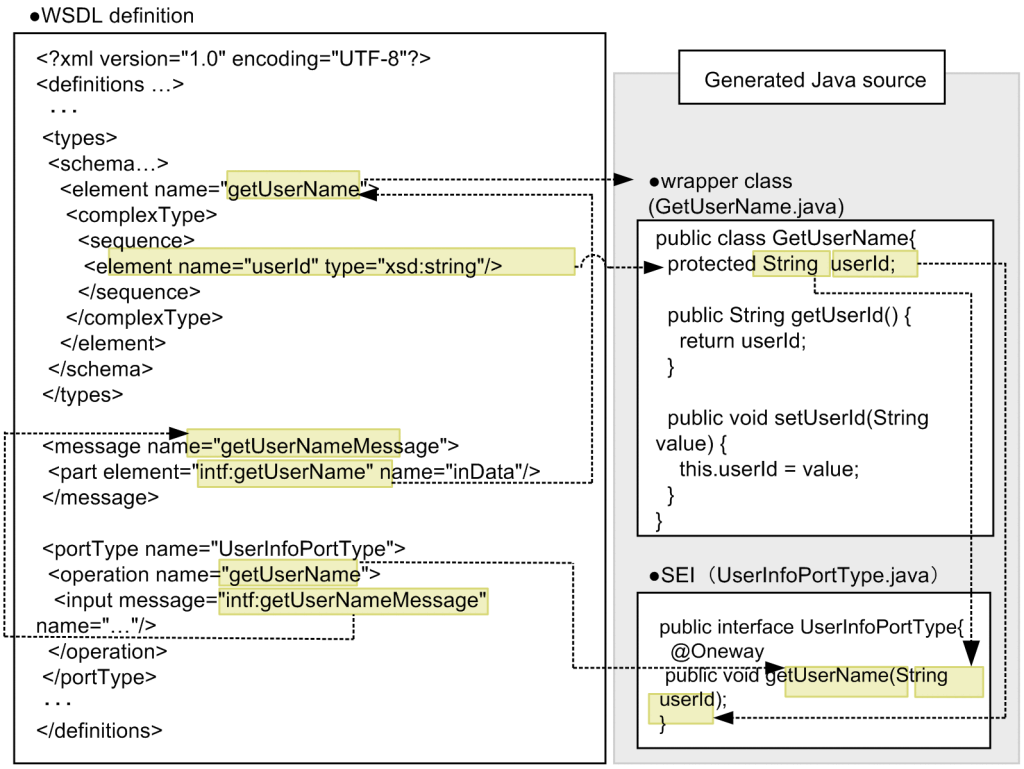
  Figure 15–4: Example of mapping message parts to parameters and return values (request-response operation)



- **For mapping from a WSDL of one-way operations**

  The wrapper child element to be referenced from the message part of a WSDL (define only one `wsdl:input` element) of the one-way operation and the parameters of the Java method are mapped. The following figure shows an example of mapping.

Figure 15-5: Example of mapping of message parts and parameters (one-way operations)



During mapping, the first letter of the wrapper child elements names of the WSDL is converted to lower-case letter in both, the request-response operations and one-way operations.

Before conversion: `WrapperName`

After conversion: `wrapperName`

- **Relationship between part types and mapping to Java source**

  The following table describes the relationship between the part types (`in`, `inout`, `out`) and the mapping to Java source:

Table 15-8: Mapping of part types to Java sources (Wrapper style)

No.	WSDL part types	Mapping to Java	
		Mapped to	Mapping method
1	`in`	Parameter	Not mapped using `javax.xml.ws.Holder<T>` class. Mapped using classes such as `java.lang.String`.
2	`inout`	Parameter	Mapped using `javax.xml.ws.Holder<T>` class. #
3	`out`	Parameter	Mapped using `javax.xml.ws.Holder<T>` class. #
4		Return value	Not mapped using `javax.xml.ws.Holder<T>` class. Mapped using classes such as `java.lang.String`.

#

If the wrapper child element type is a type that is mapped with JAXB specifications to a Java primitive type such as `xsd:int`, since the type corresponds to primitive, set up a non-null value for the `Holder` instance for sending.

(2) Conditions for the wrapper child element name

In the wrapper child element name, you can code a string that fulfills all the conditions described in the following table. However, when customizing with the binding declaration, you can code the strings that can be used as `xsd:NCName` type of the XML Schema specification.

Table 15-9: Conditions for strings that can be coded in the wrapper child element name (wrapper style)

No.	Conditions	Examples of invalid strings	Operations when invalid strings are specified
1	Strings using only one-byte alphanumeric characters (0 to 9, A to Z, a to z) and underscore (_)	`Hitachi_wrapper`	The operation might not function properly (error message is not displayed).
2	Strings other than Java reserved words #	`abstract`	An error message is output in the standard error output and log and the processing ends (KDJW51018-E).
3	Strings that do not begin with numeric characters	`1User_wrapper`	An error message is output in the standard error output and log and the processing ends.

#

You cannot code strings such as `Abstract` where the first character of the Java reserved word is in upper case (since the first character is converted into a lower case character due to mapping).

(3) Conditions for handling multiple wrapper child elements as the same wrapper child element

When the wrapper child element that appears in the input message or output message is coded several times in WSDL, handling of the wrapper child elements differs depending on whether the local name and the XML Schema type of wrapper child elements are same or different, as described in the following table:

Table 15-10: Different handling of wrapper child elements depending on local name and XML Schema type of wrapper child elements

No.	Local name of wrapper child element	XML Schema type of wrapper child element	Handling of wrapper child element
1	When the local name is same	When the XML Schema type is same	Handled as the same wrapper child element, when each of the wrapper child elements indirectly reference to the same global element using the `ref` attribute of the `xsd:element` element.
2		When the XML Schema type is different	Handled as separate wrapper child elements.
3	When the local name is different	When the XML Schema type is the same	Handled as separate wrapper child elements.
4		When the XML Schema type is different	

(4) Notes for coding multiple wrapper child elements

When you define same wrapper child elements and different wrapper child elements multiple times in a WSDL file as the composite-type child elements, and execute the `cjwsimport` command by specifying this WSDL file, SEI will be mapped with the non-wrapper style.

The following is an example of the WSDL file where SEI is mapped with the non-wrapper style:

```
<wsdl:definitions name="TestJaxWsService"
                  xmlns:soap=http://schemas.xmlsoap.org/wsdl/soap/
                  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
                  xmlns:tns="http://example.com/example"
```

```
                  targetNamespace="http://example.com/example">

    <xsd:element name="getUserData" type="tns:getUserData"/>
     ...
    <xsd:complexType name="getUserData">
      <xsd:sequence>
        <xsd:element name="in0"  type="xsd:string"/>
        <xsd:element name="in0"  type="xsd:string"/>
        <xsd:element name="hoge" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
    ...
  </xsd:schema>
 </wsdl:types>
...
 <wsdl:message name="getUserDataRequest">
  <wsdl:part name="inputParameters" element="tns:getUserData"/>
 </wsdl:message>
 ...
</wsdl:definitions>
```

(5) Precautions for mapping to parameters

- When you map the wrapper child element to Java, even if the parameter types of the method are different, if the parameter names are the same, an error message is output in the standard error output and log and the processing ends.
- The parameters mapped from the wrapper child elements of `in` and `inout` are mapped in the order of appearance of the corresponding wrapper child elements in the wrapper element. The parameters mapped from the wrapper child element of `out` are mapped in the order of appearance of the corresponding wrapper child elements in the wrapper element.
- If the wrapper child elements of `in`, `inout`, and `out` are mixed, the wrapper child elements of `in` and `inout` are mapped in the order of appearance of the corresponding wrapper child elements in the wrapper element. Then the wrapper child element of `out` is mapped in the order of appearance of the corresponding wrapper child elements in the wrapper element.
- The parameters of `out` (excluding the parameters mapped to the return value) and `inout` that are Java primitive type, Java array type, and user-defined type are mapped to the `Holder` type (`javax.xml.ws.Holder<T>`) in the Java source. The example is as follows:

  Example:

  Data type of the part of `out` and `inout`: `java.lang.String`

  Data type after mapping to Java: `javax.xml.ws.Holder<java.lang.String>`
- Specify 0 to 254 as the number of parameters after mapping to Java. If you specify 255 or more parameters, an error message is output in the standard error output and log and the processing ends (KDJW51016-E).

(6) Precautions for mapping to the return values

If there is 1 wrapper child element of `out` or if the local name of the wrapper child element of `out` is "return", that value is mapped to the return value of the method. However, even if the types are different, if you code multiple wrapper child elements with the local name "return", an error message is output in the standard error output and log and the processing ends.

15.1.5 Mapping the message part to the parameter and return value (For non-wrapper style)

This subsection describes the mapping of the WSDL message part (`wsdl:part` child element of the `wsdl:message` element) to the parameters and return values of the Java methods.

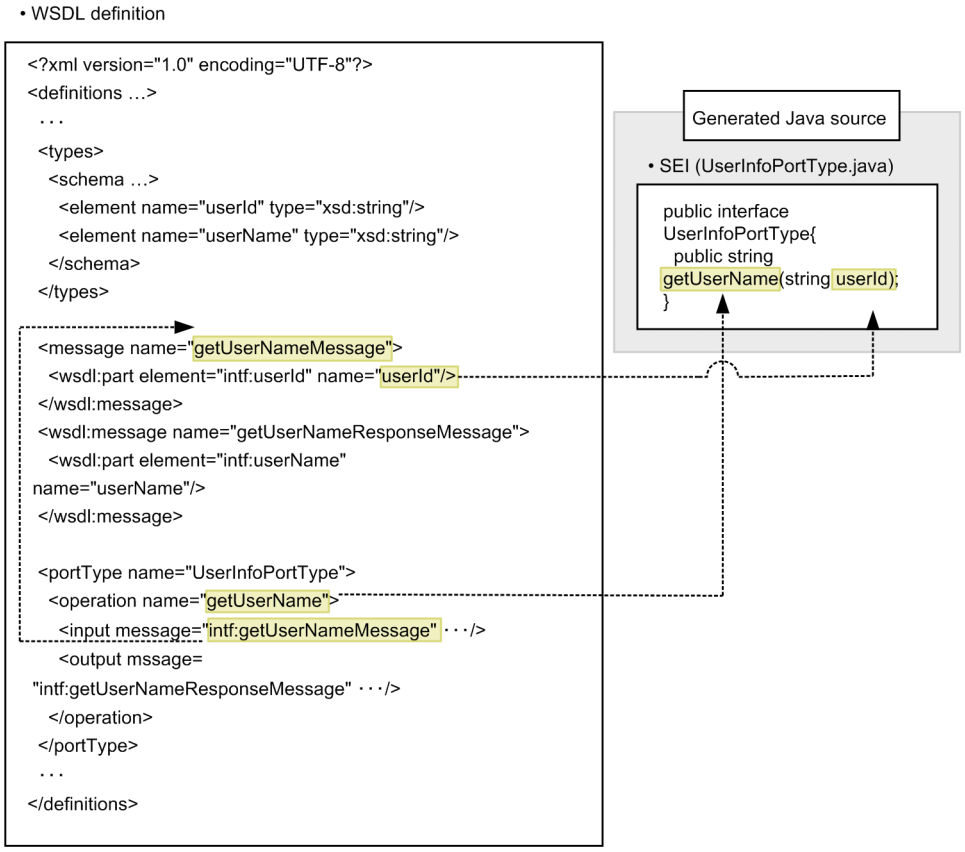
This subsection describes the mapping for the non-wrapper style.

(1) Mapping

- **For mapping from a WSDL of a request-response operation**

  The following figure shows an example of mapping of parameters and return values of the Java methods and the message part of a WSDL (define one `wsdl:input` element and one `wsdl:output` element , and zero or more `wsdl:fault` elements) of a request-response operation.

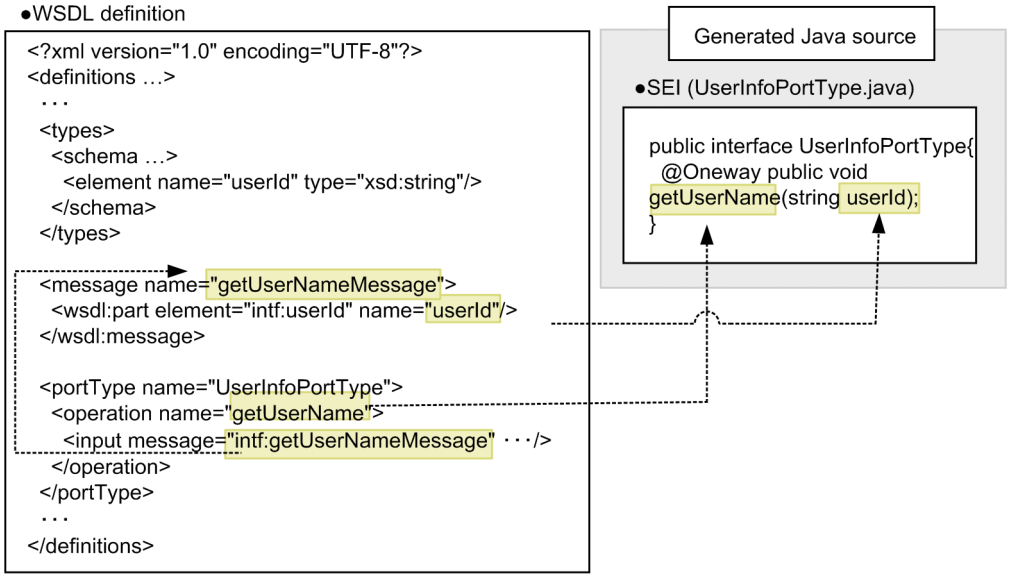
  Figure 15-6: Example of mapping the message part to the parameters and return values (request-response operation)



- **For mapping from a WSDL of a one-way operation**

  The following figure shows an example of mapping of parameters of the Java method and the message part of a WSDL (define only one `wsdl:input` element) of a one-way operation. Also, annotate `javax.jws.Oneway` in the mapped Java method.

Figure 15-7: Example of mapping of message parts and parameter (one-way operation)



During the mapping, the first character of the message part name (`name` attribute of the `wsdl:part` element) is converted into a lower-case character in both, the request-response operations and one-way operations.

Before conversion: `PartName`

After conversion: `partName`

- **Relationship between the part types and the mapping to Java source**

  The following table describes the relationship between the part types (`in`, `inout`, `out`) and the mapping to Java source:

  Table 15-11: Relationship between the part types and the mapping to Java source (Non-wrapper style)

No.	WSDL part types	Mapping to Java	
		Mapped to	Mapping method
1	`in`	Parameter	Not mapped using `javax.xml.ws.Holder<T>` class. Mapped using classes such as `java.lang.String`.
2	`inout`	Parameter	Mapped using `javax.xml.ws.Holder<T>` class.
3	`out`	Parameter	Mapped using `javax.xml.ws.Holder<T>` class.
4		Return value	Not mapped using `javax.xml.ws.Holder<T>` class. Mapped using classes such as `java.lang.String`.

(2) Conditions for part names

In the part name, you can code a string that fulfills all the conditions described in the following table. However, when customizing with the binding declaration, you can code the strings that can be used as `xsd:NCName` type of the XML Schema specification.

Table 15-12: Conditions for strings that can be coded in the part name (non-wrapper style)

No.	Conditions	Examples of invalid strings	Operations when invalid strings are specified
1	Strings using only one-byte alphanumeric characters (0 to 9, A to Z, a to z) and underscore (_)	`Hitachi_part`	The operation might not function properly (error message is not displayed).

No.	Conditions	Examples of invalid strings	Operations when invalid strings are specified
2	Strings other than Java reserved words #	`abstract`	An error message is output in the standard error output and log and the processing ends (KDJW51017-E).
3	Strings that do not begin with numeric characters	`1User_part`	An error message is output in the standard error output and log and the processing ends (KDJW51029-E).

#

You cannot code strings such as `Abstract` where the first character of the Java reserved word is in upper case (since the first character is converted into a lower case character due to mapping).

(3) Handling when multiple parts reference the same global element

When the part that appears in the input message or output message is coded several times in WSDL, the part is handled as the same part only when the part name is the same and the global element being referenced is the same.

If either the part name or the global element being referenced is different, the part is handled as a different part.

(4) Precautions for mapping to the parameters

- The part of `in` and `inout` is mapped in the order of appearance of the corresponding part in the input message. The part of `out` is mapped in the order of appearance of the corresponding part in the output message.
- If the part of `in`, `inout`, and `out` are mixed, the part of `in` and `inout` is mapped in the order of appearance of the corresponding part in the input message and then the part of `out` is mapped in the order of appearance of the corresponding part in the output message.
- The part of `out` (excluding the parameters mapped to the return value) and `inout` that are Java primitive type and Java array type, user-defined type are mapped to the `Holder` type (`javax.xml.ws.Holder<T>`) in the Java source. The example is as follows:

  Example:

  Data type of the part of `out` and `inout`: `java.lang.String`

  Data type after mapping to Java: `javax.xml.ws.Holder<java.lang.String>`

(5) Precautions for mapping to the return values

If there is only 1 part of `out` in the output message, the part is mapped to the return value of the method. If there are 2 or more parts of `out`, the parts are mapped to the return value.

15.1.6 Mapping the schema type to the Java type

This subsection describes the mapping of the types defined in the WSDL schema (`xsd:schema` child element of the `wsdl:types` element) to the Java types.

(1) Mapping

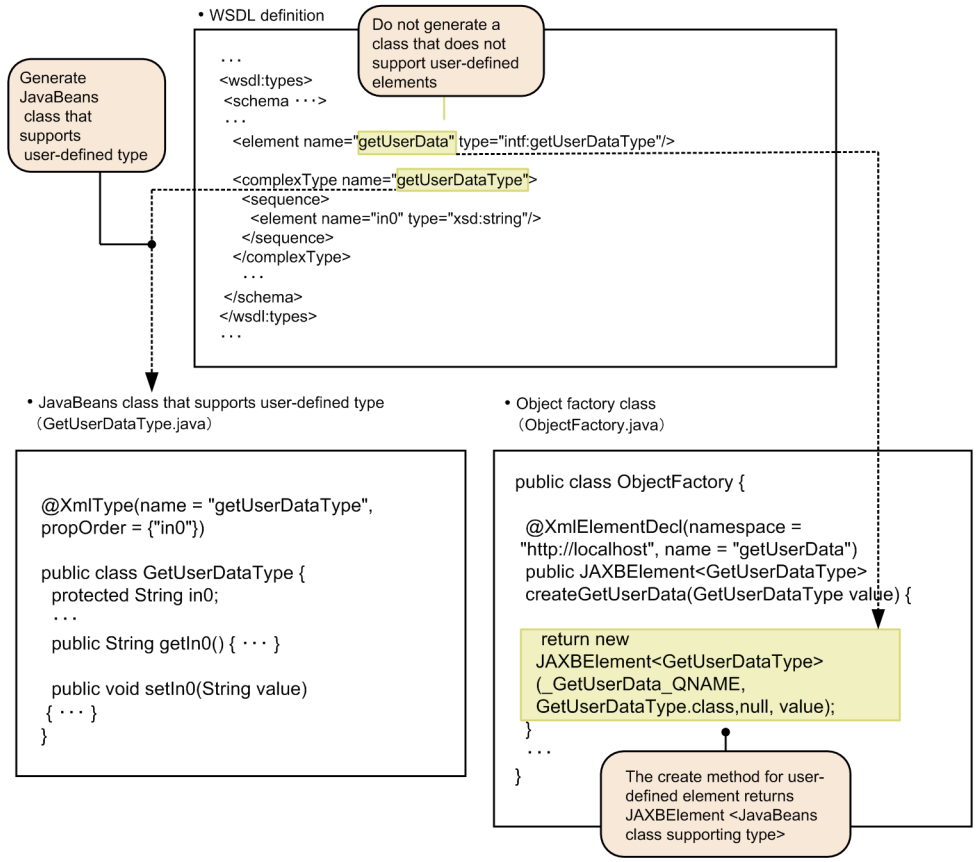
The WSDL schema type and Java type are mapped in accordance with the JAXB 2.2 specifications.

The class-based mapping is performed when the schema type is mapped to the Java type. The operations of class-based mapping are as follows:

- A `JavaBeans` class corresponding to the user-defined type is generated.
- A class corresponding to the user-defined element is not generated.
- The `ObjectFactory` class is output. Also, the corresponding `JAXBElement`*JavaBeans-class-for-the-type* is returned using the `create` method for the user-defined element.

The following figure shows an example of mapping:

Figure 15-8: Example of mapping the schema type to the Java type



Reference note

Class-based mapping

Indicates that the mapping is the same as if the `generateValueClass` attribute of the `globalBindings` element of JAXB is true and the `generateElementClass` attribute is false (default state of no specification).

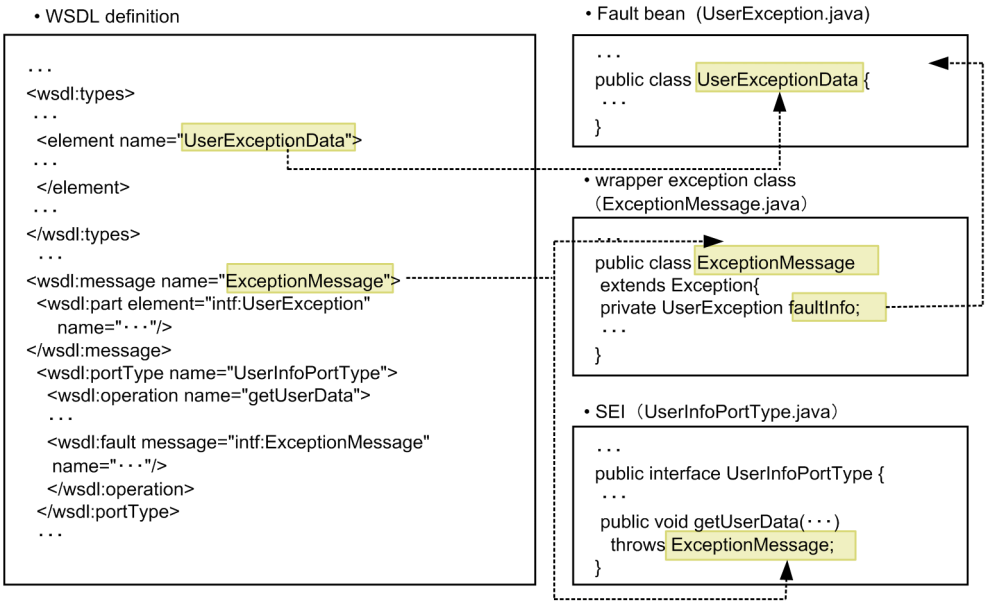
15.1.7 Mapping the fault to the exception class

This subsection describes the mapping of the WSDL fault (`name` attribute of the `wsdl:message` element referenced from the `wsdl:fault` element) to the exception class.

(1) Mapping

When the `cjwsimport` command is executed, the WSDL fault is mapped to the Java type in accordance with the JAX-WS 2.2 specifications. The following figure shows an example of mapping:

Figure 15-9: Example of mapping the fault to the exception class



- **Mapping to the fault bean**

  A fault bean is generated in accordance with the JAX-WS 2.2 specifications. The global element declaration referenced from the fault part is mapped to the fault bean.

- **Generated wrapper exception class**

  The wrapper exception class is generated in accordance with the JAX-WS 2.2 specifications. The generated wrapper exception class inherits the `java.lang.Exception` class and has the `javax.xml.ws.WebFault` annotation. The generated wrapper exception class also has the following methods:

  - Constructor of `FaultMessageName(String message, FaultBean faultInfo)`#

    This method has the message string and fault bean class as the arguments. Also, the constructor of the parent class `javax.xml.ws.WebFault` is invoked in this constructor.

  - Constructor of `FaultMessageName(String message, FaultBean faultInfo, Throwable cause)`#

    This method has the message string and fault bean class, and protocol-specific exception information as the argument. Also, the constructor of the parent class `javax.xml.ws.WebFault` is invoked in this constructor.

  - `getFaultInfo()` method

    This method does not have arguments. The return value is the fault bean class.

  #

  "`FaultMessageName`" indicates the message name (`name` attribute of the `wsdl:message` element) referenced from the fault. Also, the argument "`FaultBean`" indicates the name of the fault bean class.

(2) Conditions for the fault name

In the fault name, you can code a string that fulfills all the conditions described in the following table. However, when customizing with the binding declaration, you can code the strings that can be used as `xsd:NCName` type of the XML Schema specification.

Table 15-13: Conditions for strings that can be coded in the fault name

No.	Conditions	Examples of invalid strings	Operations when invalid strings are specified
1	Strings using only one-byte alphanumeric characters (0 to 9, A to Z, a to z) and underscore (_)	`Hitachi_fault`	The operation might not function properly (error message is not displayed).
2	Strings that do not begin with numeric characters	`1User_fault`	An error message is output in the standard error output and log and the processing ends (KDJW51029-E).

(3) Number of parts of the messages referenced from the fault

The fault can reference messages with only 1 part coded. The following table describes the number of parts of the messages referenced from the fault and the operations:

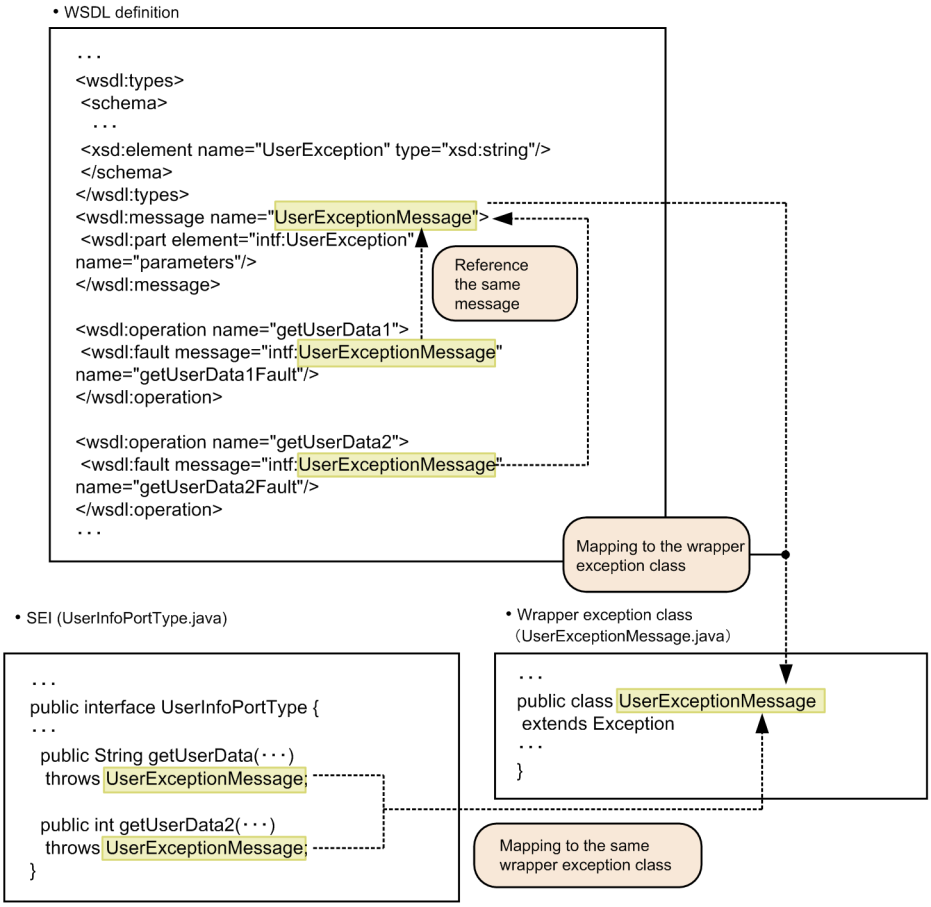
Table 15-14: Number of parts of the messages referenced from the fault and the operations

No.	Number of coding	Operations
1	0	An error message is output in the standard error output and log and the processing ends (KDJW51025-E).
2	1	Terminates normally.
3	2 or more	An error message is output in the standard error output and log and the processing continues (KDJW51025-E).

(4) Handling when the same operation fault references the same message

When many different operation faults reference the same message, all the faults are handled as the same fault. Therefore, when mapping to Java, a common wrapper exception class is assumed. The following figure shows an example:

Figure 15-10: Example of mapping when the same operation fault references the same message



The following table describes the relationship between the operation coding the fault and the messages referenced from the fault:

Table 15-15: Relationship between the operation coding the fault and the referenced messages

No.	Operation coding the fault	Messages referenced from the fault	Handling of the fault
1	Different	Same	Handled as the same fault.
2		Different	Handled as a different fault.
3	Same	Same	An error message is output in the standard error output and log and the processing ends (KDJW51026-E).
4		Different	Handled as a different fault.

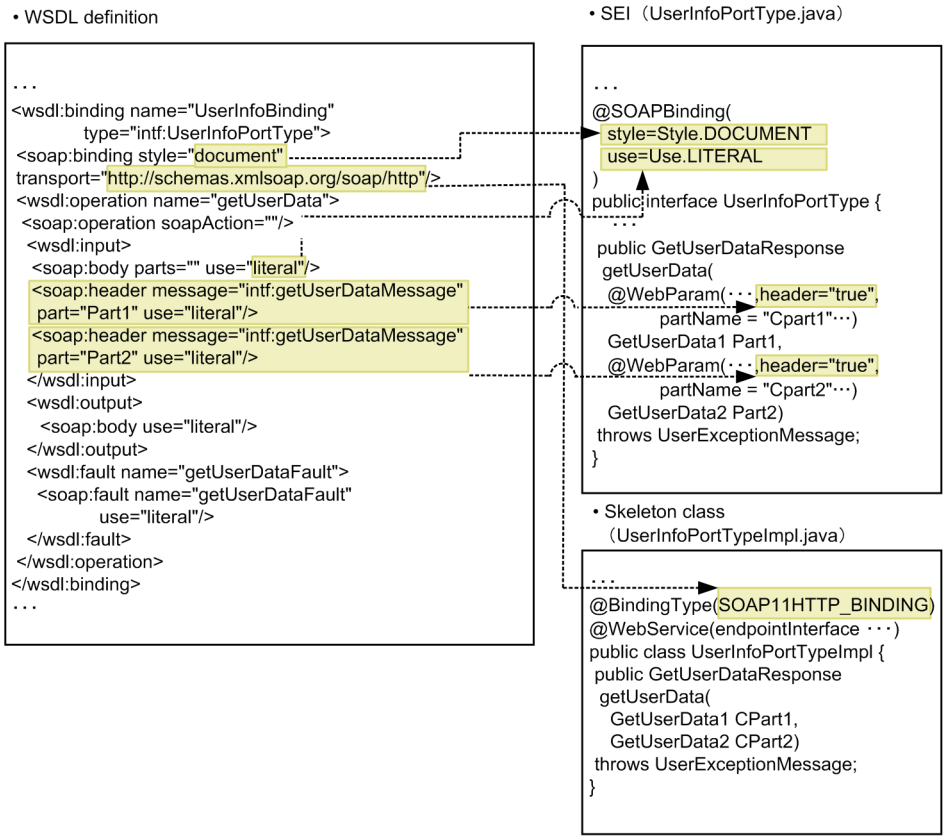
15.1.8 Mapping the binding extension element to the parameter

This subsection describes the mapping of the binding extension element (`wsdl:binding` element) of WSDL binding to the method parameters.

(1) Mapping

The binding extension elements of the WSDL binding and the Java method parameters are mapped in accordance with the JAX-WS 2.2 specifications. The following figure shows an example of mapping:

Figure 15-11: Example of mapping the binding extension elements to the parameters



- **SOAP binding**

  You can code SOAP binding in the binding extension element.

  ! Important note

  When multiple `soap:header` elements are coded, make sure that the local name of the global element referenced from the message corresponding to each `soap:header` element is unique. If the local name is the same, an error message is output in the standard error output and log and the processing ends (KDJW51205-E).

Mapping from the `transport` attribute value of the `soap:binding` or `soap12:binding` elements to the `javax.xml.ws.BindingType` annotation

The following table describes the mapping from the `transport` attribute value of the `soap:binding` element or the `soap12:binding` element, which is a child element of the `wsdl:binding` element of WSDL to the `javax.xml.ws.BindingType` annotation.

Table 15-16: Mapping the `transport` attribute value to `javax.xml.ws.BindingType`

Sr.No	SOAP Version	`transport` attribute value	BindingType annotation value
1	SOAP 1.1	`http://schemas.xmlsoap.org/soap/http`	`http://schemas.xmlsoap.org/soap/http`[#1]
2		`http://schemas.xmlsoap.org/wsdl/soap/http?mtom=true`	`--`[#2]
3	SOAP 1.2	`http://schemas.xmlsoap.org/soap/http#3`	`http://www.w3.org/2003/05/soap/bindings/HTTP/`

Sr.No	SOAP Version	transport attribute value	BindingType annotation value
4	SOAP 1.2	http://www.w3.org/2003/05/soap/bindings/HTTP/	http://www.w3.org/2003/05/soap/bindings/HTTP/
5		http://www.w3.org/2003/05/soap/bindings/HTTP/?mtom=true	--#2

Legend:
--: None.

#1
Being the same value as the default value, in reality, the javax.xml.ws.BindingType annotation is omitted.

#2
Error occurs when the cjwsimport command is executed (KDJW51147-E).

#3
Since the general specifications of the transport attribute are ambiguous, you can use the following URL in JAX-WS.

- **MIME binding**
  MIME binding is not supported.

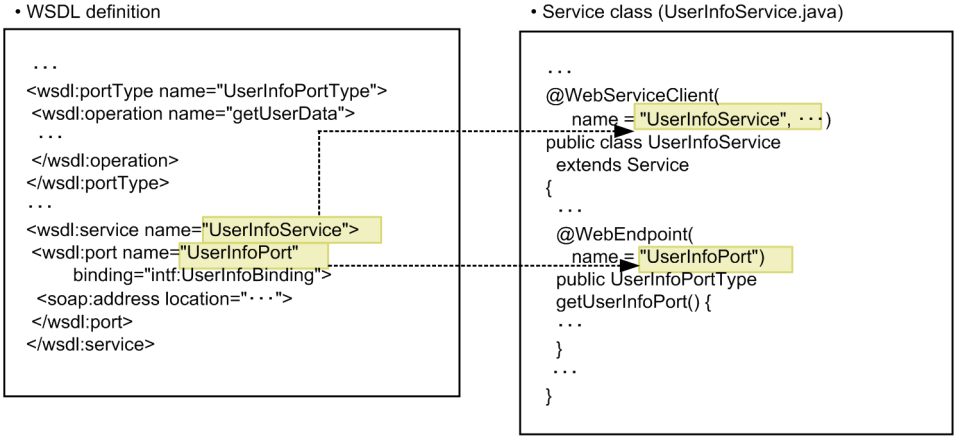
15.1.9 Mapping the service and port to the service class

This subsection describes the mapping of the WSDL service (name attribute of the wsdl:service element) and port (name attribute of the wsdl:port element) to the service class.

(1) Mapping

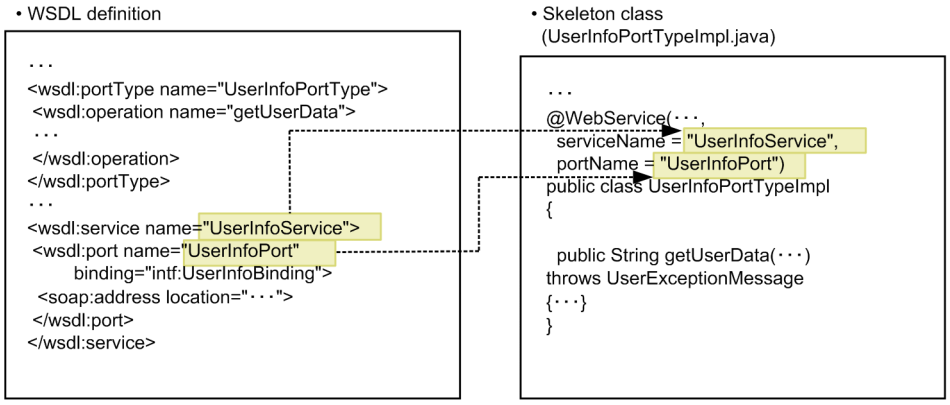
The WSDL service and port and the service class are mapped in accordance with the JAX-WS 2.2 specifications. The following figure shows an example of mapping:

Figure 15-12: Example of mapping the service and port to the service class



The WSDL service and port are also mapped to the skeleton class. The following figure shows an example of mapping:

Figure 15-13: Example of mapping the service and port to the skeleton class



- **Generated service class**

  The generated service class inherits the `javax.xml.ws.Service` class and has the `javax.xml.ws.WebServiceClient` annotation. The generated service class also has the following methods.

  Table 15-17: The methods included in a service class

No.	Type of the return value	Method Name or Description		Support
1	--	`ServiceName()`[#1]		Y
		Description	A constructor of the parent class `javax.xml.ws.Service`[#2] (`java.net.URL wsdlDocumentLocation, javax.xml.namespace.QName serviceName`) is called in this constructor. For `wsdlDocumentLocation` and `serviceName`, use the definitions mapped from a WSDL. When the catalog functionality is enabled, map the definition to a different URI specified in the catalog file and then use the definition mapped from the WSDL as `wsdlDocumentLocation`. For details on the catalog functionality, see *27. Catalog Functionality*.
		Exception	`javax.xml.ws.WebServiceException`
2	--	`ServiceName(javax.xml.ws.WebServiceFeature... features)`[#1]		N
3	--	`ServiceName(java.net.URL wsdlLocation)`[#1]		Y
		Description	A constructor of the parent class `javax.xml.ws.Service`[#2] (`java.net.URL wsdlDocumentLocation, javax.xml.namespace.QName serviceName`) is called in this constructor. For `ServiceName`, use a definition mapped from a WSDL. If the catalog functionality is enabled, map the URL pointing to the WSDL location specified in this argument to the URI pointing to a different WSDL location. For details on the catalog functionality, see *27. Catalog Functionality*.
		Argument	wsdlLocation: A URL indicating the WSDL Location.

No.	Type of the return value	Method Name or Description		Support
3	--	Exception	`javax.xml.ws.WebServiceException`	Y
4	--	`ServiceName(java.net.URL wsdlLocation, javax.xml.ws.WebServiceFeature... features)`[#1]		N
5	--	`ServiceName(java.net.URL wsdlLocation, javax.xml.namespace.QName serviceName)`[#1]		Y
		Description	A constructor of the parent class `javax.xml.ws.Service`[#2] (`java.net.URL wsdlDocumentLocation, javax.xml.namespace.QName serviceName`) is called in this constructor.	
		Argument	wsdlLocation: A URL indicating the WSDL Location. If the catalog functionality is enabled, map the URL pointing to the WSDL location specified in this argument to the URI pointing to a different WSDL location. For details on the catalog functionality, see *27.Catalog Functionality*. serviceName: QName of the service.	
		Exception	`javax.xml.ws.WebServiceException`	
6	--	`ServiceName(java.net.URL wsdlLocation, javax.xml.namespace.QName serviceName, javax.xml.ws.WebServiceFeature... features)`		N
7	Proxy that implements the SEI	`getPortName()`[#3]		Y
		Description	`The getPort`[#2] (`javax.xml.namespace.QName portName, java.lang.Class<T> serviceEndpointInterface`) method of the parent class `javax.xml.ws.Service` is called. For `portName` and `serviceEndpointInterface`, use the definitions mapped from a WSDL. This method is annotated by `javax.xml.ws.WebEndpoint`.	
		Exception	`javax.xml.ws.WebServiceException`	
8	Proxy that implements the SEI	`getPortName(javax.xml.ws.WebServiceFeature... features)`[#3]		Y
		Explanation	The `getPort`[#2] (`javax.xml.namespace.QName portName, java.lang.Class<T> serviceEndpointInterface, WebServiceFeature... features`) method of the parent class `javax.xml.ws.Service` is called. For `portName` and `serviceEndpointInterface`, use the definitions mapped from a WSDL. This method is annotated by `javax.xml.ws.WebEndpoint`.	
		Argument	Features: The `javax.xml.ws.WebServiceFeature type of variable length`.	

No.	Type of the return value	Method Name or Description		Support
8	Proxy that implements the SEI	Exception	`javax.xml.ws.WebServiceException`	Y

Legend:

--: Indicates that the type of return value is not available.

Y: Supported by the JAX-WS functionality of Cosminexus.

N: Not supported by the JAX-WS functionality of Cosminexus.

#1

`ServiceName` represents the service class name (the `name` attribute of the `wsdl:service` element).

#2

For details on the parent class `javax.xml.ws.Service`, see *19.2.2(4) javax.xml.ws.Service class*.

#3

`PortName` represents a name with the first letter of the port name in uppercase (the `name` attribute of the `wsdl: port` element).

(2) Conditions for the service name and port name

In the service name and port name, you can code a string that fulfills all the conditions described in the following table. However, when customizing with the binding declaration, you can code the strings that can be used as `xsd:NCName` type of the XML Schema specification.

Table 15-18: Conditions for strings that can be coded in the service name and port name

No.	Conditions	Examples of invalid strings	Operations when invalid strings are specified
1	Strings using only one-byte alphanumeric characters (0 to 9, A to Z, a to z) and underscore (_)	`Hitachi_service` `Hitachi_port`	The operation might not function properly (error message is not displayed).
2	Strings that do not begin with numeric characters	`1User_service` `1User_port`	An error message is output in the standard error output and log and the processing ends (KDJW51029-E).

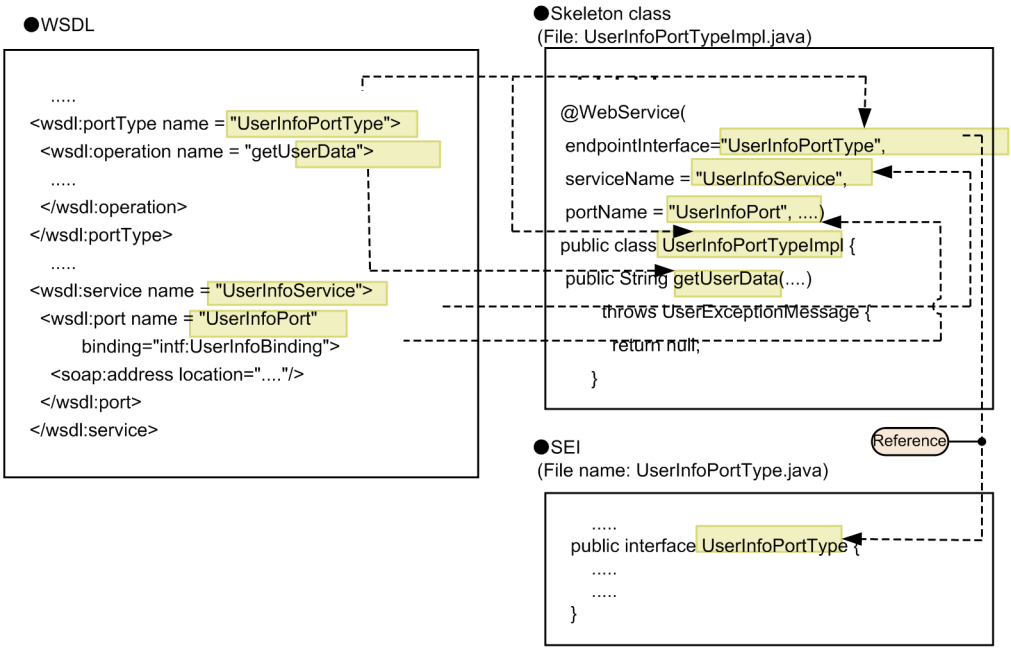
15.1.10 Mapping to the skeleton class

This section describes the skeleton class that implements SEI.

(1) Mapping

You can use the `cjwsimport` command to map the WSDL services and ports to the skeleton class. The following figure shows an example of mapping the WSDL services and ports to the skeleton class.

Figure 15-14: Example of mapping the WSDL services and ports to the skeleton class



When mapping to the skeleton class, append the suffix `Impl` to the WSDL port type name (the `name` attribute of the `wsdl:portType` element).

15.1.11 Precautions on mapping from WSDL to Java

This subsection describes the precautions on mapping from WSDL to Java.

(1) Overloading of Java methods

When multiple operations are coded in 1 port type, all the operation names must be unique. Therefore, when mapping WSDL to Java, you cannot overload the Java methods. If the operation name is duplicated, customize the name and specify a unique name for each operation.

(2) Mapping for name conflict

When the `cjwsimport` command is executed, a name conflict might occur in the SEI name, class name, method name, and parameter name. This point describes the mapping for name conflict.

(a) Mapping for conflict between the SEI name and class name

When mapping WSDL to the Java source, if a name conflict occurs in the SEI name and class name (non-exception Java class name, exception class name, service class name, and skeleton class name), the name conflict is resolved according to the priority order.

The following table describes the priority order and solutions for a name conflict. *No.* indicates the priority order (No. 1 is the highest).

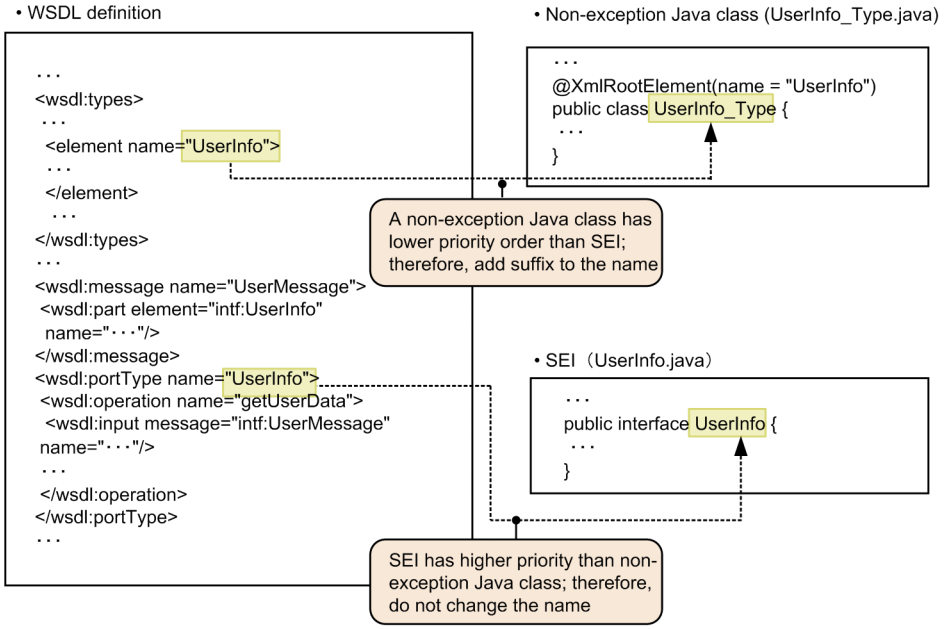
Table 15-19: Priority order and solutions for name conflict

No.	Types	Solutions for name conflict
1	SEI name	The priority order is highest, so the name is not changed.
2	Non-exception Java class name	Suffix `_Type` is added to the class name.
3	Exception class name	Suffix `_Exception` is added to the class name.

No.	Types	Solutions for name conflict
4	Service class name	Suffix `_Service` is added to the class name.
5	Skeleton class name	Suffix `_Impl` is added to the class name.

The following figure shows an example of name resolution when a conflict occurs between the SEI name and the non-exception Java class name:

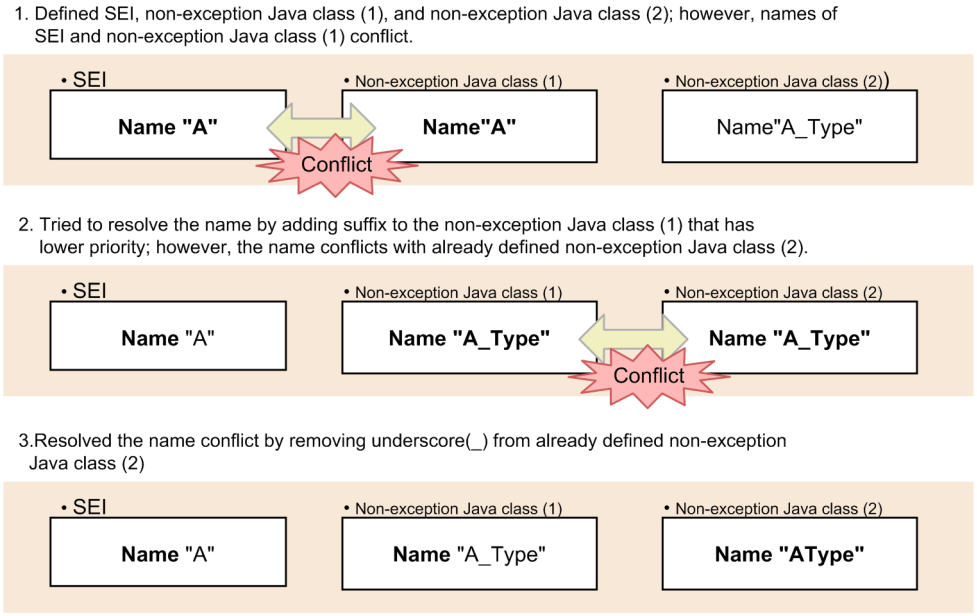
Figure 15-15: Name resolution when a conflict occurs between the SEI name and the non-exception Java class name



If a class name with the same name as the class name to which a suffix is added due to name resolution is defined, name conflict occurs again. In this case, the conflict is resolved by deleting the underscore from the defined class name.

The following figure shows an example of name resolution when a name conflict occurs after a suffix is added:

Figure 15-16: Example of name resolution when a name conflict occurs after a suffix is added



Moreover, if a class name with the same name as the class name that was changed by the deletion of the underscore is defined, an error message is output in the standard error output and log and the processing ends (KDJW51030-E#).

#

For a non-exception Java class, a different message is output.

(b) Mapping for conflict between the method name and parameter name

When mapping WSDL to the Java source, if a name conflict occurs in the methods and method parameters, the name conflict is not resolved and an error occurs.

(3) Supporting JAXB annotations

The `cjwsimport` command supports Comformance 2.17 of the JAX-WS 2.2 specifications. When you execute the `cjwsimport` command, the following JAXB annotations are added to SEI as required:

- `javax.xml.bind.annotation.XmlAttachmentRef`
- `javax.xml.bind.annotation.XmlList`
- `javax.xml.bind.annotation.adapters.XmlJavaTypeAdapter`
- `javax.xml.bind.annotation.XmlMimeType`

Note that the MIME binding is not supported in the Cosminexus JAX-WS functionality, so the `javax.xml.bind.annotation.XmlMimeType` annotation is not added. If MIME binding is specified, an error message is output in the standard error output and log and the processing ends (KDJW51188-E).

15.2 Customized mapping of WSDL to Java

You can customize the mapping of WSDL to the Java source by using the binding declaration. The methods of customization using the binding declaration are as follows:

- Customizations in the embedded binding declaration
- Customizations with the external binding file

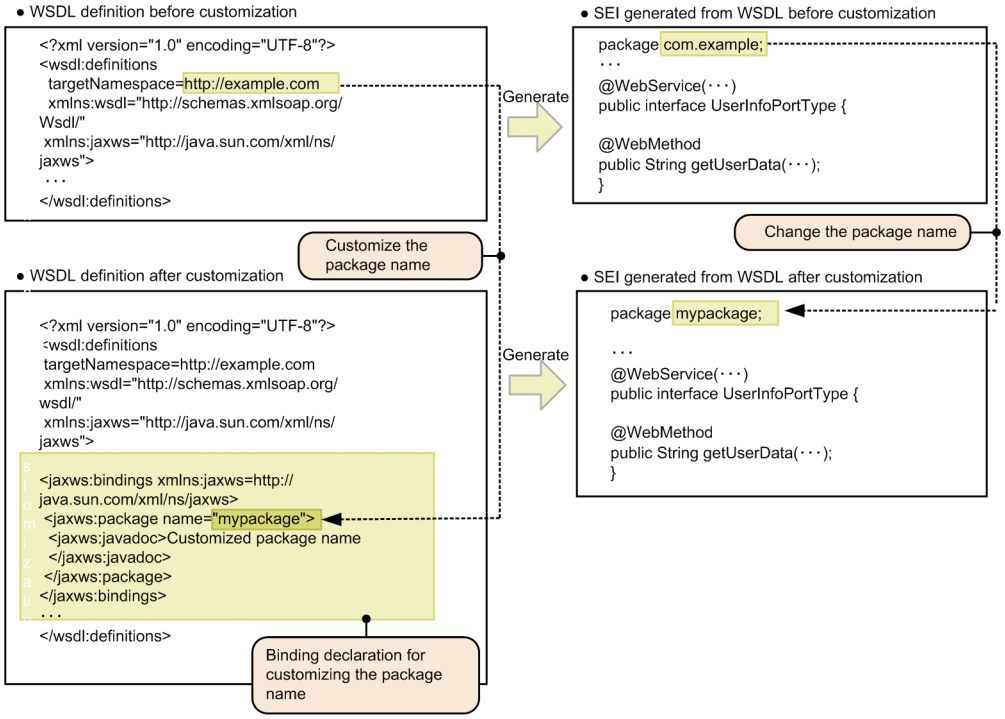
This section describes the customization methods and precautions for each of the above methods.

15.2.1 Customizations in the embedded binding declaration

To customize with the embedded binding declaration, use the `jaxws:bindings` element, code the binding declaration directly in the WSDL document, and customize.

The following figure shows an example of using the embedded binding declaration to customize the package name:

Figure 15‑17: Example of package name customization (Embedded binding declaration)



The following points describe the points to remember when you customize with the embedded binding declaration:

(1) Specifying the jaxws:bindings element

The `jaxws:bindings` element is used as the container of the embedded binding declaration.

However, you cannot code the `jaxws:bindings` element as the child element of the `jaxws:bindings` element. If coded as the child element of the `jaxws:bindings` element, an error message is output in the standard error output and log and the processing ends (KDJW51034-E).

The following table describes the attributes of the `jaxws:bindings` element and the operations depending on whether the attribute is specified:

Table 15-20: Relationship between the attributes of the jaxws:bindings element and the operations depending on whether the attribute is specified (Embedded binding declaration)

No.	Elements	Specification of attribute	Operations
1	`wsdlLocation`	Yes	The attribute cannot be specified. Even if the attribute is specified, it is ignored.
2		No	Terminates normally.
3	`Node`	Yes	The attribute cannot be specified. Even if the attribute is specified, it is ignored.
4		No	Terminates normally.
5	`Version`	Yes	Only `2.0` can be specified in the attribute. If a value other than `2.0` is specified, it is ignored and `2.0` is assumed.
6		No	Terminates normally.

(2) Available binding declarations

The following table lists the binding declarations that can be used in the Cosminexus JAX-WS functionality when you use the embedded binding declaration. For details about each binding declaration, see *JAX-WS 2.2 specifications*.

Table 15-21: Available binding declarations (Embedded binding declaration)

Element name	Attribute name	Description
`wsdl:definitions` and `jaxws:bindings` elements	`version`	In the `version` attribute, you code the WSDL customization version.
Child element of `jaxws:bindings` element	--	This is the child element of the `jaxws:bindings` element.
`jaxws:package`	`name`	In the `name` attribute, you code the Java package name corresponding to the `targetNamespace` attribute of the `wsdl:definitions` element.
`jaxws:javadoc`	--	This is the Javadoc string that is added to the Java package.
`jaxws:enableWrapperStyle`	--	Shows the enabling or disabling of wrapper style for all the WSDL operations.
`jaxws:enableAsyncMapping`	--	Shows the enabling or disabling of asynchronous mapping for all the WSDL operations.
`wsdl:definitions`, `wsdl:portType`, and child element of the `jaxws:bindings` element	--	This is the child element of the `jaxws:bindings` element included in the `wsdl:definitions` and `wsdl:portType` element.
`jaxws:class`	`name`	In the `name` attribute, you code the completely modified name of SEI corresponding to the `wsdl:portType` element.
`jaxws:javadoc`	--	You code the Javadoc string to be added to SEI.
`jaxws:enableWrapperStyle`	--	Shows the enabling or disabling of wrapper style for the `wsdl:portType` element.
`jaxws:enableAsyncMapping`	--	Shows the enabling or disabling of asynchronous mapping for the `wsdl:portType` element.
`wsdl:definitions`/ `wsdl:portType`/ `wsdl:operation`/ Child element of `jaxws:bindings` element	--	This is the child element of the `jaxws:bindings` element included in the

Element name	Attribute name	Description
wsdl:definitions/ wsdl:portType/ wsdl:operation/ Child element of jaxws:bindings element	--	wsdl:definitions, wsdl:portType, and wsdl:operation elements.
jaxws:method	name	In the name attribute, you code the Java method name corresponding to the wsdl:operation element.
jaxws:javadoc	--	This is the Javadoc string that is added to the method.
jaxws:enableWrapperStyle	--	Shows the enabling or disabling of wrapper style for the wsdl:operation element.
jaxws:enableAsyncMapping	--	Shows the enabling or disabling of asynchronous mapping for the wsdl:operation element.
jaxws:parameter	part	In the part attribute, you code the XPath expression that identifies the wsdl:part child element of the wsdl:message element.
	childElementName	In the childElementName attribute, you code the child element name of the global type definition or global element declaration referenced by the wsdl:part element.
	name	In the name attribute, you code the parameter name of the element identified by the part attribute and childElementName attribute.
wsdl:definitions, wsdl:portType, wsdl:operation, wsdl:fault, and child element of the jaxws:bindings element	--	This is the child element of the jaxws:bindings element included in the wsdl:definitions, wsdl:portType, wsdl:operation, and wsdl:fault elements.
jaxws:class	name	In the name attribute, you code the completely modified name of the exception class corresponding to the wsdl:fault element.
jaxws:javadoc	--	This is the Javadoc string that is added to the exception class.
wsdl:definitions, wsdl:binding, wsdl:operation element, and child element of the jaxws:bindings element	--	This is the child element of the jaxws:bindings element included in the wsdl:definitions, wsdl:binding, wsdl:operation elements.
jaxws:parameter	part	In the part attribute, you code the XPath expression that identifies the wsdl:part child element of the wsdl:message element.
	childElementName	In the childElementName attribute, you code the child element name of the global type definition or global element declaration referenced by the wsdl:part element.
	name	In the name attribute, you code the parameter name of the element identified by the part attribute and childElementName attribute.
wsdl:definitions, wsdl:service, child element of jaxws:bindings element	--	This is the child element of the jaxws:bindings element included in the wsdl:definitions, wsdl:service elements.

Element name	Attribute name	Description
jaxws:class	name	In the name attribute, you code the completely modified name of the service class corresponding to the wsdl:service element.
jaxws:javadoc	--	This is the Javadoc string that is added to the service class name.
wsdl:definitions, wsdl:service, wsdl:port, child element of jaxws:bindings element	--	This is the child element of the jaxws:bindings element included in the wsdl:definitions, wsdl:service, wsdl:port elements.
jaxws:method	name	In the name attribute, you code the getter method name corresponding to the wsdl:port element.
jaxws:javadoc	--	This is the Javadoc string that is added to the getter method.
jaxws:provider	--	If you specify true, SEI is not generated. The getter method of the port is omitted in the service interface that is generated. For details about the jaxws:provider element, see the section *15.2.7 Operation when the jaxws:provider element is coded*.

Legend:

--: Indicates that attributes that can be used in the binding declaration do not exist.

As described in the above table, the location where the jaxws:bindings element and its child element can be coded in a WSDL is defined in the JAX-WS 2.2 specifications. If these elements are coded in locations that are not defined, an error message is output in the standard error output and log and the processing ends (KDJW51029- E).

If you code elements that cannot be coded as the child element of a jaxws:bindings element, an error message is output in the standard error output and log and the processing ends (KDJW51040-E).

Furthermore, if you code attributes that cannot be coded in the attributes of the jaxws:bindings element and its child element, an error message is output in the standard error output and log and the processing ends (KDJW51029-E).

(3) Repetition of elements and attributes

If an attribute is coded repeatedly in the jaxws:bindings element and its child element, an error message of Cosminexus XML Processor is output to the standard error output and log and the processing ends.

You cannot specify the customization of the same object by using the embedded binding declaration. Also, you cannot specify the child element of the jaxws:bindings element repeatedly. If specified, the operation might not function properly.

(4) Priority order of jaxws:enableWrapperStyle element

You can code the jaxws:enableWrapperStyle element in the following locations in WSDL. The following is the priority order when the element is coded in multiple locations at the same time:

1. wsdl:portType/wsdl:operation/jaxws:bindings/jaxws:enableWrapperStyle
2. wsdl:portType/jaxws:bindings/jaxws:enableWrapperStyle
3. wsdl:definitions/jaxws:bindings/jaxws:enableWrapperStyle

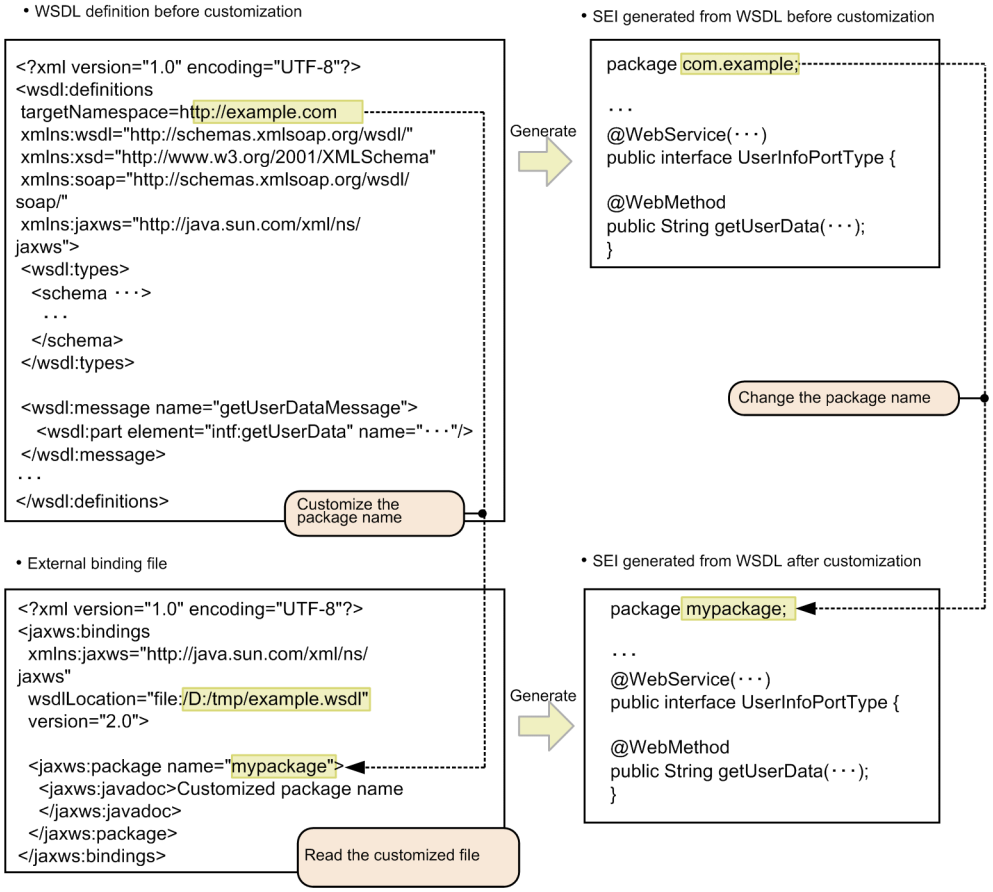
The number shows the priority order. No.1 is the highest and the element value with a high priority order is enabled.

15.2.2 Customizations with the external binding file

To customize with the external binding file, prepare a file that collectively codes the binding declaration separately from WSDL, read that file at the same time as the WSDL document, and customize. The external binding file can be read several times.

The following figure shows an example of using the external binding file to customize the package name:

Figure 15－18: Example of package name customization (external binding file)



The following points describe the points to remember when you customize with the external binding file:

(1) Specifying the jaxws:bindings element

Like the embedded binding declaration, for the external binding file, the `jaxws:bindings` element is used as a container.

However, unlike the embedded binding declaration, you can code the `jaxws:bindings` element as the child element of the `jaxws:bindings` element.

The following table describes the attributes of the `jaxws:bindings` element and the operations depending on whether the attribute is specified:

Table 15－22: Relationship between the attributes of the jaxws:bindings element and the operations depending on whether the attribute is specified (external binding file)

No.	Coding location	Elements	Specification of attribute	Operations
1	Root `jaxws:bindings`[#1]	`wsdlLocation`	Yes	Make sure you specify the attribute. The WSDL at the specified location is assumed to be the WSDL file to be customized. For details on the

No.	Coding location	Elements	Specification of attribute	Operations
1	Root `jaxws:bindings`[#1]	`wsdlLocation`	Yes	method of specification, see *15.2.2(1)(a) Coding format of the wsdlLocation attribute*.
2			No	If the attribute is not specified, the external binding file is ignored (terminates normally without customization).
3		`Node`	Yes	You can specify the attribute in the XPath 1.0 format. The specified element is assumed to be the target for customization. For details about the method of specification, see *15.2.2(1)(b) Coding format of the node attribute*.
4			No	The element to be customized is assumed a WSDL root (`wsdl:definitions` element).
5		`Version`	Yes	"2.0" can be specified. For details about the method of specification, see *15.2.2(1)(c) Coding format of the version attribute*.
6			No	`2.0` is assumed to be specified.
7	Non-root `jaxws:bindings`[#2]	`wsdlLocation`	Yes	The attribute cannot be specified. Even if the attribute is specified, it is ignored.
8			No	Terminates normally.
9		`node`	Yes	Make sure you specify the attribute. The specified element is assumed to be the target for customization. For details about the method of specification, see *15.2.2(1)(b) Coding format of the node attribute*.
10			No	There is no target for customization, so terminates normally without customization.
11		`version`	Yes	The attribute cannot be specified. Even if the attribute is specified, it is ignored.
12			No	Terminates normally.

#1

'Root `jaxws:bindings`' indicates the `jaxws:bindings` element coded at the top of the external binding file.

#2

'Non-root `jaxws:bindings`' indicates the `jaxws:bindings` element coded under the child element of the root `jaxws:bindings`.

(a) Coding format of the wsdlLocation attribute

You specify the value to be specified in the `wsdlLocation` attribute of the `jaxws:bindings` element with a URL. The files that you specify using a URL might be either remote files or local files. You can also specify a local file with a relative path.

If you use the wrong format for coding or if the file does not exist, an error message is output in the standard error output and log and the processing ends (KDJW51043-E).

For the URL, use a string complying with the rules of the RFC 2396 specifications. Perform percent encoding with UTF-8 according to the rules of the RFC 2396 specifications, when a character string not complying with rules of the RFC 2396 specifications is used. However, you cannot use an ampersand (`&`) even when the percent encoding is performed. If the rules of the RFC 2396 specifications are not followed and you specify the characters and character strings that are not encoded, the operation is not guaranteed. Furthermore, the operation might not function properly if a file other than the WSDL file is specified in the `wsdlLocation` attribute.

The following is an example of the correct coding of the `wsdlLocation` attribute:

```
<jaxws:bindings xmlns:jaxws=http://java.sun.com/xml/ns/jaxws
 wsdlLocation="file:///D:/tmp/example.wsdl" version="2.0">
 ...
</jaxws:bindings>
```

(b) Coding format of the node attribute

You specify the value to be specified in the `node` attribute of the `jaxws:bindings` element using the XPath 1.0 format.

If you use the wrong format for coding, an error message is output in the standard error output and log and the processing ends (KDJW51038-E).

The following is an example of coding of the `node` attribute. In this example, the `name` attribute of the `wsdl:definitions`/ `wsdl:portType` elements indicates a binding declaration for the `AddNumbersImpl` element.

```
<jaxws:bindings node="wsdl:definitions/wsdl:portType[@name='AddNumbersImpl']">
 ...
</jaxws:bindings>
```

In the `node` attribute of the non-root `jaxws:bindings` element, you can specify the relative path (XPath 1.0 format) from the elements to be customized, specified in the `node` attribute of the root `jaxws:bindings` element. The following is the coding example in this case:

```
<jaxws:bindings node="wsdl:definitions/wsdl:portType[@name='UserInfoPortType']"
 ...
 <jaxws:bindings node="../wsdl:service[@name='UserInfoService']">
 ...
 </jaxws:bindings>
</jaxws:bindings>
```

(c) Coding format of the version attribute

Specify `2.0` as the value of the `version` attribute of the `jaxws:bindings` element.

If a value other than `2.0` is coded, an error message is output in the standard error output and log and the processing ends (KDJW51039-E).

The following is an example of the correct coding of the `version` attribute:

```
<jaxws:bindings xmlns:jaxws=http://java.sun.com/xml/ns/jaxws
 wsdlLocation="file:///D:/tmp/example.wsdl" version="2.0">
 ...
</jaxws:bindings>
```

(2) Available binding declarations

The following table lists the binding declarations that can be used in the Cosminexus JAX-WS functionality when you use the external binding file. For details about each binding declaration, see *JAX-WS 2.2 specifications*.

Table 15-23: Available binding declarations (External binding file)

Element name	Attribute name	Description
`jaxws:bindings` element	`wsdlLocation`	In the `wsdlLocation` attribute, you code the file path (URL) of the external binding file.
	`node`	In the `node` attribute, you code the element to be customized in WSDL.
	`version`	In the `version` attribute, you code the WSDL customization version.
The child element of the `jaxws:bindings` element	--	This is the child element of the `jaxws:bindings` element.

Element name	Attribute name	Description
jaxws:package	name	In the name attribute, you code the Java package name corresponding to the targetNamespace attribute of the wsdl:definitions element.
jaxws:javadoc	--	This is the Javadoc string that is added to the Java package.
jaxws:enableWrapperStyle	--	Shows the enabling or disabling of wrapper style for each element.
jaxws:enableAsyncMapping	--	Shows the enabling or disabling of asynchronous mapping for each element.
jaxws:class	name	In the name attribute, you code the class name corresponding to each element.
jaxws:javadoc	--	This is the Javadoc string that is added to the class.
jaxws:method	name	In the name attribute, you code the Java method name corresponding to each element.
jaxws:javadoc	--	This is the Javadoc string that is added to the method.
jaxws:parameter	part	In the part attribute, you code the XPath expression that identifies the wsdl:part child element of the wsdl:message element.
	childElementName	In the childElementName attribute, you code the child element name of the global type definition or global element declaration referenced by the wsdl:part element.
	name	In the name attribute, you code the parameter name of the element identified by the part attribute and childElementName attribute.
jaxws:provider	--	If you specify true, SEI is not generated. The getter method of the port is omitted in the service interface that is generated. For details about the jaxws:provider element, see the section *15.2.7 Operation when the jaxws:provider element is coded*.

Legend:

--: Indicates that attributes that can be used in the binding declaration do not exist.

As described in the above table, the location where the jaxws:bindings element and its child element can be coded in a WSDL is defined in the JAX-WS 2.2 specifications. If these elements are coded in locations that are not defined, an error message is output in the standard error output and log and the processing ends (KDJW51029-E).

If you code elements that cannot be coded as the child element of the jaxws:bindings element, an error message is output in the standard error output and log and the processing ends (KDJW51040-E).

Furthermore, if you code attributes that cannot be coded in the attributes of the jaxws:bindings element and its child element, an error message is output in the standard error output and log and the processing ends (KDJW51029-E).

Note that you cannot code the binding declaration of JAXB specifications. If coded, the operations might not function properly.

(3) Repetition of elements and attributes

If an attribute is coded repeatedly in the jaxws:bindings element and its child element, an error message of Cosminexus XML Processor is output in the standard error output and log and the processing ends.

You cannot specify the customization of the same object by using the external binding file. If specified, the operations might not function properly.

(4) Customization of WSDL read in the wsdl:import element

When you want to customize WSDL to be imported with the `wsdl:import` element using the external binding file, specify WSDL to be imported with the `wsdl:import` element using the `wsdlLocation` attribute of the `jaxws:bindings` element.

When customizing WSDL to be imported with the `wsdl:import` element, if WSDL at the `wsdl:import` source is specified by mistake, the target for customization specified in the `node` attribute of the `jaxws:bindings` element is not found. In this case, an error message is output in the standard error output and log and the processing ends (KDJW51187-E).

15.2.3 Concurrent specification of the embedded binding declaration and external binding file

When the targets for customization in the embedded binding declaration and external binding file are different, the respective customization contents are enabled.

If the targets for customization in the embedded binding declaration and external binding file are the same, the embedded binding declaration is disabled.

15.2.4 Value that can be specified in the jaxws:bindings element

The following table lists the elements and attributes that can be specified in the `jaxws:bindings` element:

Table 15-24: Specification of attributes of the jaxws:bindings element

Element name	Attribute name	Specification
The child element of the `wsdl:definitions/jaxws:bindings` element		
`jaxws:package`	`name`	Y
`jaxws:javadoc`	--	Y
`jaxws:enableWrapperStyle`	--	Y
`jaxws:enableAsyncMapping`	--	Y
`jaxws:enableMIMEContent`	--	N
The child element of the `wsdl:portType/jaxws:bindings` element		
`jaxws:class`	`name`	Y
`jaxws:javadoc`	--	Y
`jaxws:enableWrapperStyle`	--	Y
`jaxws:enableAsyncMapping`	--	Y
The child element of the `wsdl:portType/wsdl:operation/jaxws:bindings` element		
`jaxws:method`	`name`	Y
`jaxws:javadoc`	--	Y
`jaxws:enableWrapperStyle`	--	Y
`jaxws:enableAsyncMapping`	--	Y
`jaxws:parameter`	`part`	Y
	`childElementName`	Y

Element name	Attribute name	Specification
jaxws:parameter	name	Y
The child element of the wsdl:portType/wsdl:operation/wsdl:fault/jaxws:bindings element		
jaxws:class	name#	Y
jaxws:javadoc	--	Y
The child element of the wsdl:binding/jaxws:bindings element		
jaxws:enableMIMEContent	--	N
The child element of the wsdl:binding/wsdl:operation/jaxws:bindings element		
jaxws:enableMIMEContent	--	N
jaxws:parameter	part	Y
	childElementName	Y
	name	Y
The child element of the wsdl:service/jaxws:bindings element		
jaxws:class	name	Y
jaxws:javadoc	--	Y
The child element of the wsdl:service/wsdl:port/jaxws:bindings element		
jaxws:method	name	Y
jaxws:javadoc	--	Y
jaxws:provider	--	Y

Legend:

--: Indicates that attributes that can be used in the binding declaration do not exist.

Y: Indicates that the elements and attributes can be specified.

N: Indicates that the elements and attributes cannot be specified (not supported).

#

When you customize the fault name, make sure that the customized fault name is not duplicated with other fault names. If the name is repeated, the operations might not function properly.

If you specify a binding element that cannot be specified, an error message is output in the standard error output and log and the processing ends (KDJW51188-E).

The following points describe the value that can be specified in the elements and attributes:

(1) Values that can be specified in the name attribute

In the name attribute, you can only specify the following values. If other value and strings are specified, the operations might not function properly.

- Value that can be used as Java identifiers
- One-byte alphanumeric characters (0 to 9, A to Z, a to z)
- Underscore (_), dollar mark ($)
- Period (.) #

#

You can use this only when customizing the wsdl: portType element, wsdl:fault element, or wsdl:service element.

When you customize the class name and method name using the `name` attribute, an error occurs if the customized name is duplicated with another name, so make sure that the name is not repeated.

When you customize the SEI name corresponding to the `Wsdl: portType` element, the exception class name corresponding to the `wsdl: fault` element or the service name corresponding to the `wsdl: service` element with the `jaxws: class` element, if you specify a class name that includes the package name, the specified class name serves as a Java class of the specified package name. If you specify a class name that does not include the package name, the specified class name serves as a Java class with the package name that is mapped from the WSDL name space (the `targetNamespace` attribute of the `wsdl:definitions` element).

(2) Values that can be specified in the jaxws:javadoc element

In the `jaxws:javadoc` element, you can only specify the following values. If other values are specified, the operations might not function properly.

- One-byte alphanumeric characters (0 to 9, A to Z, a to z)
- One-byte signs ("/*", "//", "\", "\n\r")[#1]
- One-byte Katakana
- Two-byte Hiragana, Two-byte Katakana, two-byte alphanumeric characters
- Two-byte level-1 Kanji set
- Java reserved words
- Spaces and null characters[#2]

#1

The one-byte sign ("*/") is handled as the end of the Javadoc, so this sign cannot be used.

#2

The line with spaces and null characters is deleted during source code generation.

(3) Values that can be specified in the jaxws:enableWrapperStyle element

In the `jaxws:enableWrapperStyle` element, you can specify the boolean value (`true` or `false`). If other values are specified, the operations might not function properly.

The wrapper style is enabled only when WSDL is coded in the wrapper style and the value of this element is specified as `true`.

(4) Values that can be specified in the jaxws:enableAsyncMapping element

In the `jaxws:enableAsyncMapping` element, you can only specify the boolean value (`false`). If other values are specified, the operations might not function properly.

(5) Values that can be specified in the part attribute

In the `part` attribute, you can only specify the following values. If other values are specified, the operations might not function properly.

- XPath expression of the `wsdl:part` child element of the existing `wsdl:message` element
- XPath expression of the `wsdl:part` child element of the `wsdl:message` referenced from the existing `soap:header`
- XPath expression of the `wsdl:part` child element of the `wsdl:message` element that is not used in the `wsdl:operation` element

(6) Values that can be specified in the childElementName attribute

You can only specify QName of the qualified name of an existing type definition. If QName of the qualified name of a non-existent type definition is specified, the operation terminates normally without resulting in an error. In this case, the target for customization is ignored.

(7) Values that can be specified in the jaxws:provider element

You can specify a boolean value (`true` or `false`). If you specify any other values, the operation is not guaranteed.

(8) Un-supported elements

You cannot specify the binding elements that are not supported in the Cosminexus JAX-WS functionality. If un-supported binding elements are specified, the operations might not function properly.

15.2.5 Values of the elements to be customized

The values of the elements in the WSDL document forming the target of customization by the embedded binding declaration or external binding file are not mapped to Java. Therefore, if the characters can be coded in the target of customization as WSDL 1.1 specifications, you might code any character.

15.2.6 Dealing with the name conflict

By customization, a name conflict might occur in the SEI name, class name, method name, and parameter name. The method of resolving the name conflict (such as priority order) is similar to that in default mapping. For details about the method of resolving the name conflict, see *15.1.11 (2) Mapping for name conflict*.

However, for the SEI name and class name, when you customize the one with a lower priority, the name is resolved when name conflict occurs, but when you customize the one with a higher priority, the name conflict is not resolved when the name conflict occurs. In this case, an error message is output in the standard error output and log and the processing ends (KDJW51030-E#).

#

For a non-exception Java class, a different message is output.

15.2.7 Operations when the jaxws:provider element is coded

When you code the `jaxws:provider` element in the `wsdl:port` element, and then execute the `cjwsimport` command, the SEI for the `wsdl:port` element is not generated. A warning message is output and the processing continues (`KDJW51206-W`). In such cases, use the `javax.xml.ws.Provider` interface to generate a Provider Implementation Class.

Also, the `getter` method of the `wsdl:port` element is omitted in the generated service class.

When you specify the `-generateService` option in the `cjwsimport` command, and code the `jaxws:provider` element in the `wsdl:port` element, the skeleton class is not generated. A warning message is output and the processing continues (`KDJW51207-W`).

15.2.8 Notes on customizing the SEI name

To customize under the following conditions, specify a value other than `Provider` in the SEI name and a value other than `http://ws.xml.javax` in the package name, and then confirm that the SEI name is not changed to `javax.xml.ws.Provider` by the including the package name:

- Package name: `jaxws:package` element
- SEI name: `jaxws:class` element

Use `javax.xml.ws.Provider` as a keyword, when coding the `jaxws:provider` element in the `wsdl:port` element, and specify for using the `javax.xml.ws.Provider` interface in the Service Implementation Class.

15.2.9 Notes on customizing inout parameter name in the jaxws: parameter element

Customize both the `input message` and `output message` parameter names when customizing inout parameter names in the `jaxws:parameter` element. Behavior is not guaranteed if only one of the parameter names is customized.

15.2.10 Skeleton class name when the SEI name is customized with the jaxws:class element

When you customize the SEI name with the `jaxws:class` element, the skeleton class name is mapped by assigning the `Impl` suffix to the customized SEI name.

16 Mapping from Java to WSDL

When you execute the `apt` command or the `cjwsgen` command, the Java source is mapped to a WSDL in accordance with the JAX-WS 2.2 specifications.

This chapter describes the default mapping and customized mapping of Java to WSDL.

16.1 Default mapping of Java to WSDL

The following table describes the corresponding relationship, when mapping a Java source with a WSDL:

Table 16-1: List of mapping of Java source to WSDL

No.	Java source	WSDL	Reference
1	Package name	WSDL name space	*16.1.1*
2	SEI name	Port type	*16.1.3*
3	Name of the method of SEI	Operation	*16.1.4*
4	Parameter and return value of the method of SEI	Part	*16.1.5*, *16.1.6*
5	SEI wrapper exception class	Fault	*16.1.7*
6	SEI and Web Service Implementation Class	Binding	*16.1.8*
7	Web Service Implementation Class	Service and port	*16.1.9*

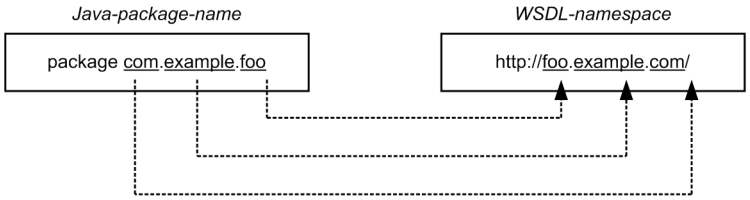
16.1.1 Mapping the package name to the name space

This subsection describes the mapping of the Java package name to the WSDL name space (`targetNamespace` attribute of the `wsdl:definitions` element).

(1) Mapping

The package name of the SEI and Web Service Implementation Class and the WSDL name space are mapped in accordance with the JAX-WS 2.2 specifications. The following figure shows an example of mapping:

Figure 16-1: Example of mapping the Java package name to the name space



(2) Conditions for the package name

You can code the strings by fulfilling all the conditions described in the following table in the Java package name segments separated by period (.). However, when using `targetNamespace` element of the `javax.jws.WebService` annotation, you can code strings complying with the naming rules of the Java identifiers provided in the Java language specifications.

Table 16-2: Conditions for strings that can be coded in the Java package name segments

No.	Condition	Examples of invalid string	Operation when an invalid string is specified
1	Strings using only one-byte alphanumeric characters (0 to 9, A to Z, a to z)	`package com.Hitachi`	The operations might not function properly (error message is not displayed).
2	Strings complying with the naming rules of the Java identifiers provided in the Java language specifications	`package com.abstract;`	When the `apt` command is executed, a compilation error occurs and the processing ends. For details, see the *JDK documentation*.

(3) Using the targetNamespace element of the javax.jws.WebService annotation

If the SEI or Web Services Implementation Class that you enter is the default package, code the name space name in the `targetNamespace` element of the `javax.jws.WebService` annotation.

If the name space name is not coded in the `targetNamespace element` of the `javax.jws.WebService` annotation, an error message is output to the standard error output and log (KDJW61004-E).

For details about the `targetNamespace` element of the `javax.jws.WebService` annotation, see *16.2.9 javax.jws.WebService annotation*.

16.1.2 Mapping the Web Service Implementation Class to SEI

This subsection describes the preconditions and points to remember when you map the Web Service Implementation Class to SEI.

(1) Conditions for Web Service Implementation Class

The following are the conditions for the Web Service Implementation Class:

- You must implement all the methods of SEI. If all the methods are not implemented, an error message is output to the standard error output and log (KDJW61011-E).
- As a Web Service operation, you cannot define the `finalize` method that overrides the `finalize` method of the `Object` class. If such a method is defined, an error message is output to the standard error output and log (KDJW61012-E).
- You must define a public default constructor. If such a constructor is not defined, an error message is output to the standard error output and log and the error is notified to the `apt` command (KDJW61013-E).
- You must code the `javax.jws.WebService` annotation. If the `javax.jws.WebService` annotation is not coded, the class is determined as a non-Web Service Implementation Class. If it is determined that the Web Service Implementation Class does not exist, a warning message is output (KDJW61001-W).
- The `javax.jws.WebService` annotation can be defined using a static inner class. If a non-static inner class is used, an error message is output to the standard error output and log (KDJW61015-E).
- Set the access modifier of the Web Service Implementation Class to `public`. You cannot specify final and abstract. If a value other than `public` is specified, an error message is output to the standard error output and log (KDJW61016-E).
- The Web Service Implementation Class might perform `implements` for SEI. If you are writing `implements`, and the `-sourcepath` option is specified when the `apt` command is executed, you must specify both SEI and the Web Service Implementation Class in the argument. If `implements` is not performed and SEI is not specified in the argument of the `apt` command, the SEI information is only referenced by the `endpointInterface` element of the `javax.jws.WebService` annotation, but is not included as a target for compilation, so the SEI class file is not generated.

(2) Using the endpointInterface element of the javax.jws.WebService annotation

In the mapping of the Web Service Implementation Class to SEI, you can use the `endpointInterface` element of the `javax.jws.WebService` annotation to link the Web Service Implementation Class and SEI.

When only the Web Service Implementation Class is defined, the `endpointInterface` element is not used. In this case, the abstract information defined in SEI is extracted from the information in the Web Service Implementation Class and is assumed that a virtual SEI exists (implicit SEI).

For details about the `endpointInterface` element of the `javax.jws.WebService` annotation, see *16.2.9 javax.jws.WebService annotation*.

16.1.3 Mapping the SEI name to the port type

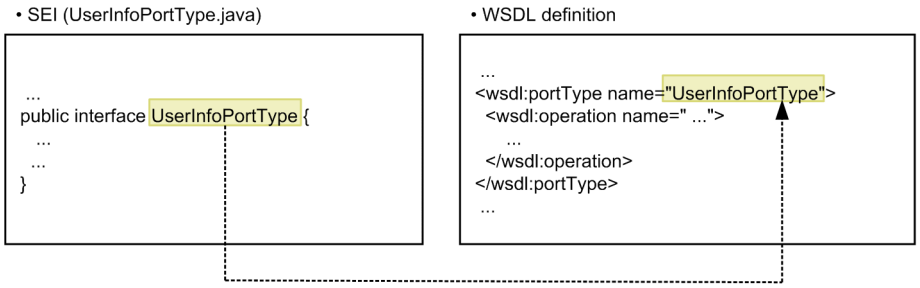
This subsection describes the mapping of the Java SEI name to the WSDL port type name (`name` attribute of the `wsdl:portType` element).

(1) Mapping

The Java SEI name and the WSDL port type are mapped in accordance with the JAX-WS 2. 2 specifications.

If the `endpointInterface` element is not used in the `javax.jws.WebService` annotation of the service implementation class, implicit SEI with the same name as the Web Service Implementation Class name is assumed to be exist and is mapped to the WSDL port type. The following figure shows an example of mapping:

Figure 16-2: Example of mapping SEI name to the port type



(2) Conditions for SEI

The following are the conditions for the Web Service Implementation Class:

- The `javax.jws.WebService` annotation must be coded. If the annotation is not coded, an error message is output to the standard error output and log (KDJW61020-E).
- The `java.rmi.Remote` interface might be inherited.

(3) Conditions for SEI name

In the SEI name, you can code strings that fulfill all the conditions described in the following table. However, when using the `name` element of the `javax.jws.WebService` annotation, you can code strings complying with the naming rules of the Java identifiers provided in the Java language specifications.

Table 16-3: Conditions for strings that can be coded in the SEI name

No.	Condition	Example of invalid string	Operation when an invalid string is specified
1	Strings using only one-byte alphanumeric characters (0 to 9, A to Z, a to z) and underscore (_)	`Hitachi_sei`	The operations might not function properly (error message is not displayed).
2	Strings complying with the naming rules of the Java identifiers provided in the Java language specifications	`abstract`	When the `apt` command is executed, a compilation error occurs and the processing ends. For details, see the *JDK documentation*.

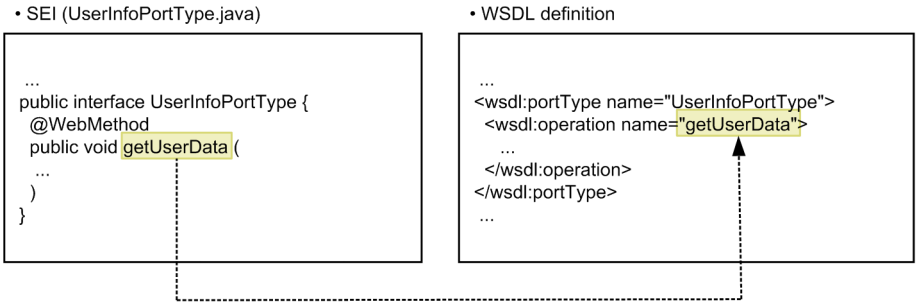
16.1.4 Mapping the name of method of SEI to an operation

This subsection describes the mapping the name of method of SEI and WSDL operation (`name` attribute of the `wsdl:operation` element).

(1) Mapping

The name of the method of SEI and the WSDL operation are mapped in accordance with the JAX-WS 2.2 specifications. The following figure shows an example of mapping:

Figure 16-3: Example of mapping the name of method of SEI to an operation



The following conditions must be fulfilled to make the SEI method public:

- The access modifier is public
- A static modifier or final modifier has not been applied
- If annotated by the `javax.jws.WebMethod` annotation, the `exclude` element of the `javax.jws.WebMethod` annotation is not `true`

The following are the rules for mapping the name of method of SEI to the operation:

- Regardless of whether the `javax.jws.WebMethod` annotation is present, all the public methods of SEI are mapped to the WSDL operation.
- If the `endpointInterface` element of the `javax.jws.WebService` annotation is not used, it is considered that an implicit SEI has the public method of the Web Service Implementation Class and is mapped to the WSDL operations. For details about the methods mapped to the implicit SEI, see *16.2.6 javax.jws.WebMethod annotation*.
- When a Web Service Implementation Class inherits another Web Service Implementation Class, all the methods meeting the following conditions are mapped to WSDL operations:

  (Conditions)

  The public method in the Web Service Implementation Class and parent Web Service Implementation Class, wherein the `exclude` element of the `javax.jws.WebMethod` annotation is not `true`.
- If a Web Service Implementation Class inherits another Web Service Implementation Class and overrides the parent class methods, the public methods overridden in the Web Service Implementation Class are mapped to the WSDL operations. The methods overridden in the parent class are not mapped.
- In the public methods that can be defined in SEI and the public methods of Web Service Implementation Class, you can define 255 public methods. If 256 or more public methods are defined, a warning message is output to the standard error output and log and the processing continues (KDJW61026-W).
- The value of the `name` attribute of the `wsdl:operation` element mapped from the name of method of SEI must be unique in WSDL. If a name conflict occurs, an error message is output to the standard error output and log (KDJW61060-E).
- If no public method exists, the system outputs an error message to the standard error output and log, and ends the operation (KDJW61093-E).

(2) Conditions for method name

In the method name, you can code strings that fulfill all the conditions described in the following table:

Table 16-4: Conditions for strings that can be coded in the method name

No.	Condition	Example of invalid string	Operation when an invalid string is specified
1	Strings using only one-byte alphanumeric characters (0 to 9, A to Z, a to z) and underscore (_)	`Hitachi_sei`	The operations might not function properly (error message is not displayed).

No.	Condition	Example of invalid string	Operation when an invalid string is specified
2	Strings complying with the naming rules of the Java identifiers provided in the Java language specifications	`Abstract`	When the `apt` command is executed, a compilation error occurs and the processing ends. For details, see the *JDK documentation*.

However, when all the following annotations are used, you can code strings complying with the naming rules of the Java identifiers provided in the Java language specifications:

- The `operationName` element of the `javax.jws.WebMethod` annotation
- The `localName` element and `className` element of the `javax.xml.ws.RequestWrapper` annotation (for wrapper style)
- The `localName` element and `className` element of the `javax.xml.ws.ResponseWrapper` annotation (for wrapper style)
- The `name` element of the `javax.jws.WebParam` annotation (for non-wrapper style)
- The `name` element of the `javax.jws.WebResult` annotation (for non-wrapper style)

(3) Name conflict due to overloading

When using the method overloading, a name conflict occurs in the default mapping; therefore, you must customize the name using annotation so that the name is unique.

The following table describes the locations where the name conflict occurs due to overloading and their references:

Table 16-5: Location where the name conflict occurs due to overloading and references

No.	Location where the name conflict occur	Annotation reference	
		Wrapper style	Non-wrapper style
1	Operation name	*16.2.6(2)*	
2	Global element of input#	*16.2.16(1)*, *16.2.16(2)*	*16.2.7(2)*, *16.2.7(5)*
3	Global element of output#	*16.2.17(1)*, *16.2.17(2)*	*16.2.8(2)*, *16.2.8(4)*
4	Request bean class name	*16.2.16(3)*	
5	Response bean class name	*16.2.17(3)*	

#

If you customize either the local name or name space, the name conflict will not occur.

16.1.5 Mapping the parameter and return value to the message part (For wrapper style)

This subsection describes the mapping of parameters of method of SEI to WSDL (`name` attribute of the `wsdl:part` element).

This subsection describes the mapping for the wrapper style.

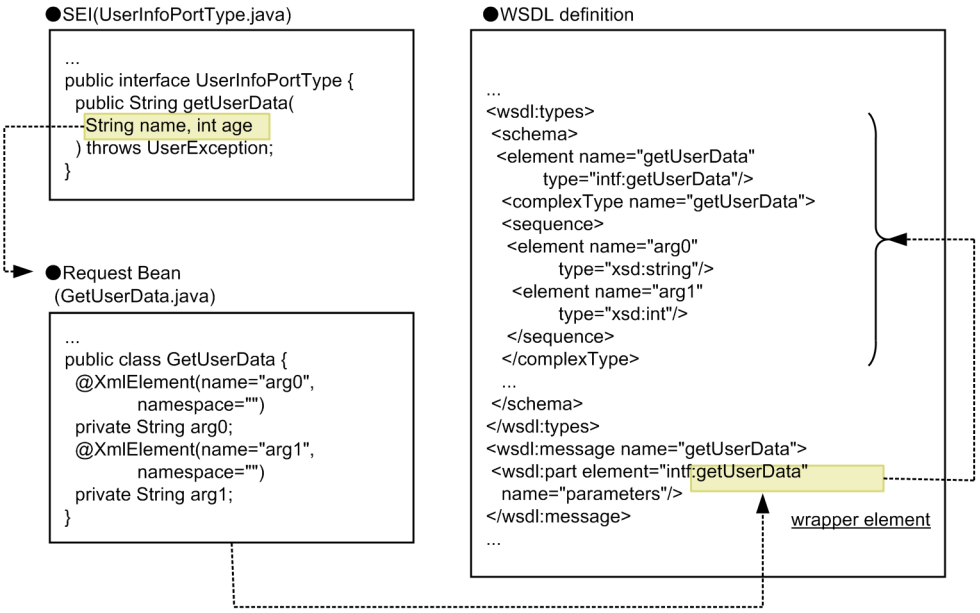
(1) Mapping

For the wrapper style, the name of the method of SEI and a request bean with the same name as the name of the method of SEI will be generated. A response bean with the suffix 'Response' added to it will also be generated. The request and response beans can either be generated automatically by using the `apt` command or `cjwsgen` command, or you can dynamically generate them when starting Web Services. When dynamically generating the request and response beans, for a preliminary error check, you can execute the `cjwsgen` command for the compiled Web

Services Implementation Class to avoid the occurrence of an error while starting Web Services. For details, see *10.23(1) Using the cjwsgen command for checking errors*.

The following figure shows an example of mapping the method parameters to the message parts.

Figure 16–4: Example of mapping the method parameters to the message parts (Wrapper style)



The following figure shows an example of mapping the method return values to the message parts:

Figure 16–5: Example of mapping the method return values to the message parts (Wrapper style)

The following are the rules for mapping the method parameters and return values to the message parts:

- The parameters and return values are mapped using an empty name space ("") as the child element of the wrapper element. The wrapper element is mapped with a name space same as that for SEI.
- The `in` parameter and the `inout` parameter are mapped as a request bean property using the name `argN`#.
- The `out` parameter and the `inout` parameter are mapped as a response bean property using the name `argN`#.

Furthermore, the return value is mapped as a response bean property using the name "return". At this time, an underscore (_) is prefixed to the field name such that the name does not become a reserved word.

- The `javax.xml.bind.annotation.XmlElement` annotation is annotated in the properties of the mapped request bean and the response bean.
- In the annotated `javax.xml.bind.annotation.XmlElement` annotation, in the request bean properties, a name called *argN*# is set for the `name` element and a blank name space ("") is set for the `namespace` element. In the response bean properties, a name called *return* is set for the `name` element and a blank name space ("") is set for the `namespace` element.
- When mapping from a parameter to a WSDL part, the mapping to a part name of the `input` message is done with the fixed value `parameters`.
- When mapping from a return value to a WSDL part, the mapping to a name of the `output` message is done with the fixed value `parameters`.

#

N in `argN` indicates 0 or a higher integer that depends on the order of the parameters.

For details on the `javax.xml.bind.annotation.XmlElement` annotation, see *16.2.10 javax.xml.bind.annotation.XmlElement annotation*.

(2) Java types that can be specified in the parameters

This point describes the conditions and precautions for specifying a Java type other than the `Holder(javax.xml.ws.Holder)` type and `Holder` type.

(a) Java type other than Holder type

- The Java types other than the `Holder` type are mapped to the WSDL schema types in accordance with the JAXB 2.2 specifications. The precautions for specifying the Java types other than the `Holder` type are as follows:
- The Java primitive type cannot be specified in the `out` and `inout` parameters. If specified, an error message is output to the standard error output and log (KDJW61035-E).
- Also, the Java primitive type cannot be specified as the type parameter of the `javax.xml.ws.Holder` class. If specified, an error occurs when executing the `apt` command and the processing ends.
- You can specify a Java type in the `out` and `inout` parameter as the type parameter of the `javax.xml.ws.Holder` class. If the Java type is specified using any other method, an error message is output to the standard error output and logs (KDJW61035-E).

(b) javax.xml.ws.Holder type

The precautions for specifying the `javax.xml.ws.Holder` type are as follows:

- When you specify the `javax.jws.WebParam` annotation in the type parameter of the `javax.xml.ws.Holder` class, you must specify `Mode.OUT` or `Mode.INOUT` in the `mode` element. If the `mode` element is not specified or if `Mode.IN` is specified in the `mode` element, an error message is output to the standard error output and logs (KDJW61031-E). Also, if the `javax.jws.WebParam` annotation is not specified, the type parameter of `javax.xml.ws.Holder` is interpreted as the `inout` parameter.
- The operations might not function properly in the following cases:
  - If the `javax.xml.ws.Holder` class is specified anywhere except in the method argument
  - If the array of the `javax.xml.ws.Holder` class is used
  - If the type is not specified in the type parameter of the `javax.xml.ws.Holder` class
  - If the `javax.xml.ws.Holder` class or a class inheriting is specified in the type parameter of the `javax.xml.ws.Holder` class
  - If a multi-dimensional array of a type other than the `byte` type and three or more dimensional array of the `byte` type are specified in the `type` parameter of the `javax.xml.ws.Holder` class when dynamically generating the request and response beans during the Web Services startup

(c) Java type mapping

The precautions for mapping the Java types are as follows:

- If Java types are not customized in the `mode` element of the `javax.jws.WebParam` annotation, the arguments other than the type parameter of the `javax.xml.ws.Holder` class are mapped as the `in` parameter and the type parameter of the `javax.xml.ws.Holder` class is mapped as the `inout` parameter. For details about the mapping as the `out` parameter, see *16.2.7(4) mode element (javax.jws.WebParam)*.
- The input message name is mapped using the operation name. The output message name is mapped using a value with 'Response' suffixed to the operation name.
- You can define up to 254 parameters of the method of SEI according to the Java language specifications. If 255 or more parameters are defined, a compilation error occurs during the execution of the `apt` command and the processing ends.

(3) Conditions for Java method parameters

The parameter names of the Java methods are not mapped to WSDL; therefore, code the Java method parameters in accordance with the Java identifier naming rules provided in the Java language specifications.

(4) Combinations of parameters and return values

You can combine and code the `in` parameter, `inout` parameter, `out` parameter, and the return value that you want.

(5) Operations during name conflict

This point describes the rules about the wrapper bean class name and global element name and the operations for a name conflict.

- **Wrapper bean class name**

  The wrapper bean class name that is generated must be a unique name in the package. However, the differences in uppercase and lower case are ignored. If the request bean or response bean have a name conflict with another `JavaBean` class generated in the package concurrently, an error message is output to the standard error output and logs (KDJW61083-E).

  If the name is duplicated with an existing class, the name is overwritten. However, if that class is included in the `apt` command argument, the error is checked when the `apt` command is executed.

- **Global element (local name and name space)**

  The global element (local name and name space) must be unique in WSDL. If the name is not unique, the operations might not function properly.

(6) Using the java.util.Map class

When you use the `java.util.Map` class in the SEI argument or return value, the following operations must be performed for the argument or return value of the `java.util.Map` type of SEI:

1. Create a value type.

   Create the `value` type (a `JavaBean` class that can be marshaled or un-marshalled) for `java.util.Map(bound type)` in accordance with the JAXB 2.2 specifications.

2. Create an adapter.

   Create an adapter that interconverts the `java.util.Map(bound type)` that inherits `javax.xml.bind.annotation.adapters.XmlAdapter` and the `value` type, and implement the `unmarshal` method and `marshal` method.

3. Annotate with the javax.xml.bind.annotation.adapters.XmlJavaTypeAdapter annotation.

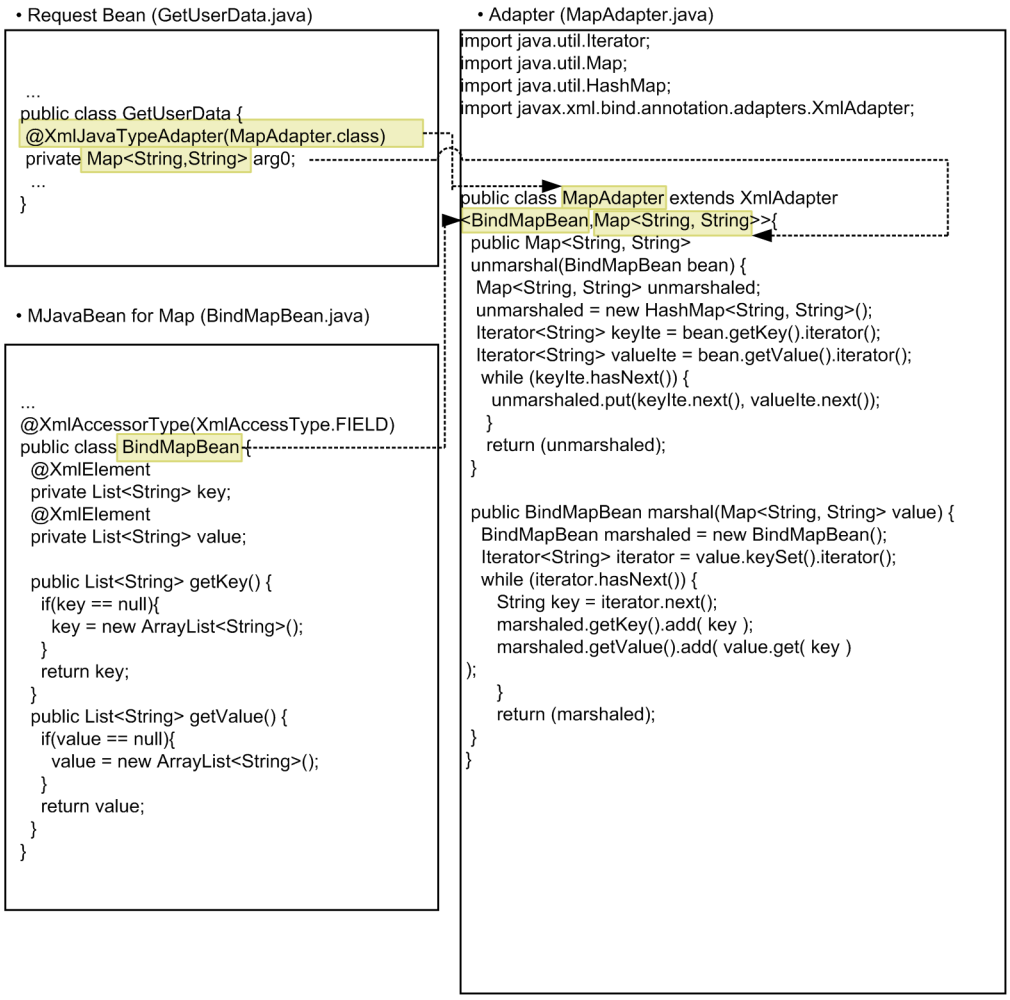
   Annotate the argument or return value of the `java.util.Map` type with the `XmlJavaTypeAdapter` annotation that has the adapter created in 2 as the value.

4. Execute the apt command.

   Interpret the annotated SEI with the `apt` command.

The following is an example of the relationship and implementation of the request bean class and response bean class in which the `xmlJavaTypeAdapter` annotation is applied through the `value` type, adapter, and the `apt` command:

Figure 16-6: Example of using java.util.Map



16.1.6 Mapping the parameter and return value to the message part (For non-wrapper style)

This subsection describes the mapping of the parameters of the method of SEI to WSDL (`name` attribute of the `wsdl:part` element).

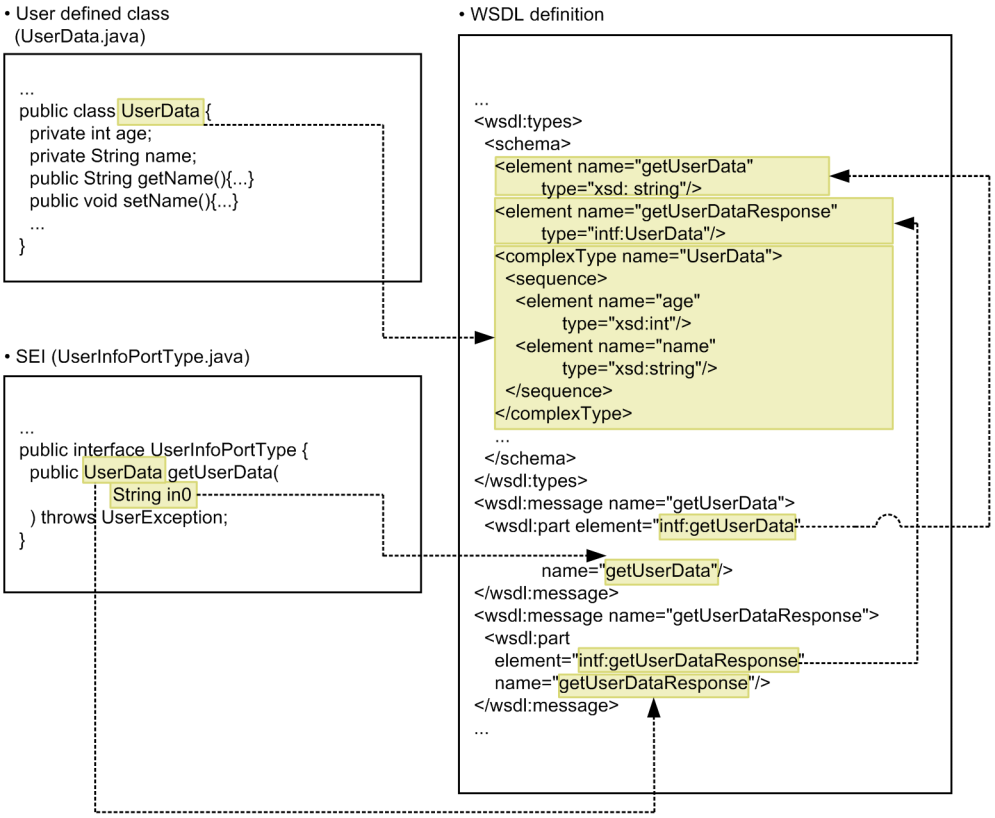
This subsection describes the mapping for the non-wrapper style.

(1) Mapping

For the non-wrapper style, the method parameters of SEI are mapped to the WSDL part and global elements using the same name as the operation name. The return value is mapped to the WSDL part and global element using the name with "Response" suffixed to the operation name. The non-wrapper style does not require the request and response beans. Therefore, the `apt` and `cjwsgen` commands do not automatically generate the request and response beans. However, you can check for errors beforehand by executing the `cjwsgen` command for the compiled Web resource Implementation Class. For details, see *10.23(1) Using the* `cjwsgen` *command for checking errors*.

The following figure shows an example of mapping:

Figure 16-7: Example of mapping the method parameters and return values to the message parts (Non-wrapper style)



(2) Java types that can be specified in the parameters

The Java types that can be specified in the parameters are the same as for the wrapper type. For details, see *16.1.5(2) Java types that can be specified in the parameters*.

(3) Conditions for the parameter names

The parameter names of the Java methods are not mapped to WSDL; therefore, code the Java method parameters in accordance with the Java identifier naming rules provided in the Java language specifications.

(4) Combinations of parameters and return values

You can specify any number of method parameters that are SOAP Header. However, as far as the method parameters that are the SOAP Body are concerned, the specifiability and the specified number are determined by the relationship with the return value. The following table describes the conditions for the return values of the methods and the specification method of the parameters:

Table 16-6: Conditions for the return values of the methods and the specification method of the parameters

No.	Conditions for the return values		Specification method of the parameters
	Presence of return values	Specified location	
1	No	--	• You can specify either one `in` parameter or one `inout` parameter. #1 • You can specify either one `out` parameter or one `inout` parameter. #2

No.	Conditions for the return values		Specification method of the parameters
	Presence of return values	Specified location	
2	Yes	SOAP header	• You can specify either one `in` parameter or one `inout` parameter. #1 • You can specify either one `out` parameter or one `inout` parameter. #2
3	Yes	SOAP body	• You can specify one `in` parameter. #1 • You cannot specify the `out` parameter and `inout` parameter. #3

#1
If 0 or 2 or more are specified in total, an error message is output to the standard error output and logs (KDJW61056-E).

#2
If 0 or 2 or more are specified in total, an error message is output to the standard error output and logs (KDJW61057-E).

#3
If the `out` parameter or `inout` parameter is specified, an error message is output to the standard error output and logs (KDJW61059-E).

If the `out` parameter or `inout` parameter is used in the non-wrapper style, you must specify the `name` element of the `javax.jws.WebParam` annotation. If the `name` element of the `javax.jws.WebParam` annotation is not specified, an error message is output to the standard error output and logs (KDJW61085-E).

(5) Relationship between the specification of the annotation and the mapping to the part name

The mapping of the method parameter to the WSDL part (`name` attribute of the `wsdl:part` element) depends on the element values of the `javax.jws.WebParam` annotation and `javax.jws.WebMethod` annotation.

The following table describes the relationship between the specification of the annotation and the mapping methods:

Table 16–7: Relationship between the specification of annotations and the mapping to the WSDL part name

No.	Specification of annotation elements			Method of mapping to the WSDL part name
	`javax.jws.WebParam` `@partName`	`javax.jws.WebParam` `@name`	`javax.jws.WebMethod` `@operationName`	
1	Specified	--	--	The element value of the specified annotation is mapped to the `name` attribute of the `wsdl:part` element.
2	Not specified	Specified	--	
3		Not specified	Specified	The Java method name is mapped to the `name` attribute of the `wsdl:part` element.
4			Not specified	

Legend:
--: Indicates that the presence or absence of specification for the element does not affect the mapping (the mapping method is the same whether the element is specified or not).

(6) Operations during name conflict

This point describes the operations for a name conflict of the part names and global elements.

- Conflict of part names
  In the non-wrapper style, the name of the `wsdl:part` element of WSDL mapped from the Java source must be unique in WSDL. If the name is not unique, the operations might not function properly.
- Conflict of global elements (local name and name space)

The global element (local name and name space) must be unique in WSDL. If the global element is not unique, the operations might not function properly.

(7) Using the java.util.Map class

For details about using the `java.util.Map` class, see *16.1.5(6) Using the java.util.Map class*.

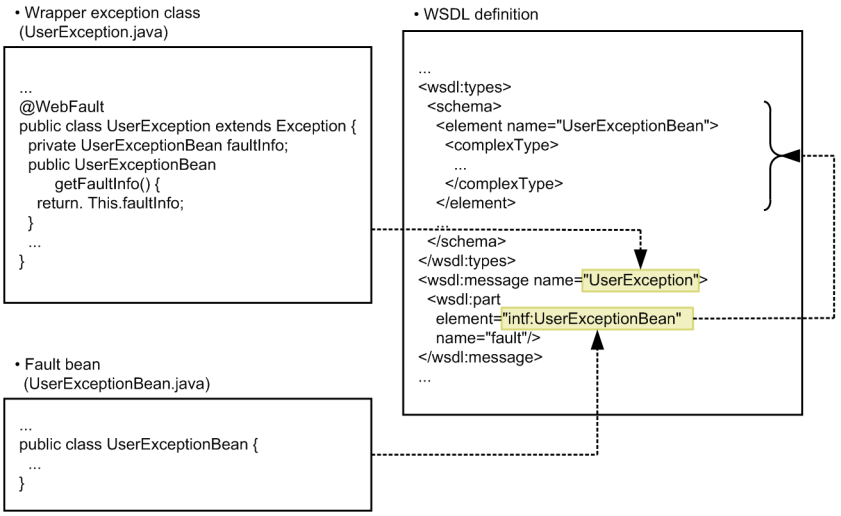
16.1.7 Mapping the Java wrapper exception class to the fault

This subsection describes the mapping of the Java wrapper exception class to the WSDL fault (`wsdl:fault` element, `wsdl:message` element with one `wsdl:part` child element, and the global element declaration of the XML Schema).

(1) Mapping

The Java wrapper exception class and the fault are mapped in accordance with the JAX-WS 2.2 specifications. The following figure shows an example of mapping:

Figure 16-8: Example of mapping the exception class to the fault



The following are the rules for mapping the wrapper exception class to the fault:

- If the wrapper exception class has the `javax.xml.ws.WebFault` annotation and the `getFaultInfo` method that returns the fault bean, the fault bean is not generated when you execute the command as the fault bean already exists.
- If the wrapper exception class does not have the `javax.xml.ws.WebFault` annotation and the `getFaultInfo` method, a fault bean is generated with a name in which "Bean" is added as a suffix to the wrapper exception class name.
- The generated fault bean has the same type or name properties that the wrapper exception class and its parent class have and the type or name properties similar to all the `getter` except the `getter` named `getCause`, `getLocalizedMessage`, and `getStackTrace` inherited from `Throwable` and the `getter` named `getClass` inherited from `java.lang.Object`.
- The `javax.xml.bind.annotation.XmlType` annotation is annotated in the generated fault bean.

  In the annotated `javax.xml.bind.annotation.XmlType` annotation, an exception class name is set in the `name` element, an SEI name space is set in the `namespace` element, and all the property names that the wrapper exception class has are set in the `propOrder` element. The property name of the `propOrder` element is set with the String type array sorted in an ascending order according to the Unicode value of each character.

  For details on the `javax.xml.bind.annotation.XmlType` annotation, see *16.2.12 javax.xml.bind.annotation.XmlType annotation*.

- The fault message name is mapped with the same name as the wrapper exception class name. Also, the fault message part name is mapped with the fixed value `fault`.
- You can define up to 255 exceptions to be thrown in one method. If 256 or more exceptions are defined, a warning message is output to the standard error output and log and the processing continues (KDJW61027-W).
- If the exception class thrown by the method is not found, a compilation error occurs.

(2) Conditions for the wrapper exception class

The following are the conditions for the wrapper exception class:

- The wrapper exception class might inherit the exception classes of `java.lang.Exception`, `java.lang.RuntimeException`, and `java.rmi.RemoteException`. However, the `java.lang.RuntimeException` and `java.rmi.RemoteException`, and its sub-classes are not handled as the wrapper exception class.
- The same wrapper exception class might be thrown by multiple methods in the same SEI.

(3) Conditions for the wrapper exception class name

In the wrapper exception class name, you can code strings that fulfill all the conditions described in the following table. The wrapper exception class name is used in WSDL even though an annotation is specified, so the name must comply with the conditions described in the following table:

Table 16–8: Conditions for strings that can be coded in the wrapper exception class name

No.	Condition	Example of invalid string	Operation when an invalid string is specified
1	Strings using only one-byte alphanumeric characters (0 to 9, A to Z, a to z) and underscore (_)	`Hitachi_exception`	The operations might not function properly (error message is not displayed).
2	Strings complying with the naming rules of the Java identifiers provided in the Java language specifications	`Abstract`	When the `apt` command is executed, a compilation error occurs and the processing ends. For details, see the *JDK documentation*.

(4) Operations during name conflict

The fault bean name must be unique in the package. However, the uppercase or lowercase differences are ignored. If a name conflict occurs between the fault bean and the JavaBean generated concurrently in the package, an error message is output to the standard error output and logs (KDJW61065-E).

If the name is duplicated with an existing class, the name is overwritten. However, if that class is included in the argument of the `apt` command, the error is checked with the `apt` command.

The global element (local name and name space) mapped from the fault bean must be unique in WSDL. If the global element is not unique, the operations might not function properly.

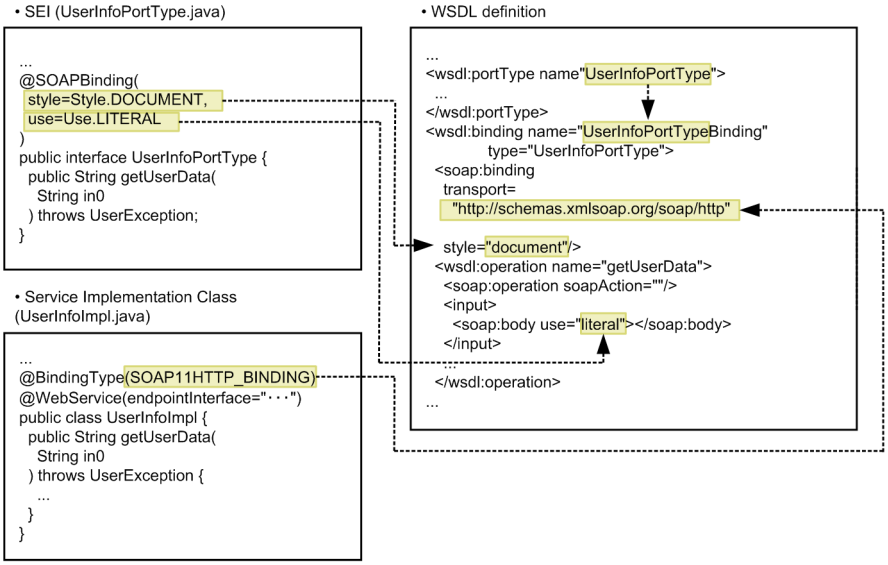
16.1.8 Mapping SEI to binding

This subsection describes the mapping of Java SEI to WSDL binding (`name` attribute of the `wsdl:binding` element).

(1) Mapping

The Java SEI and Web Service Implementation Class and WSDL binding are mapped in accordance with the JAX-WS 2.2 specifications. The following figure shows an example of mapping:

Figure 16-9: Example of mapping SEI to binding



The following are the rules for mapping SEI to binding:

- The Java SEI and Web Service Implementation Class are mapped to one `wsdl:binding` element and 0 or more `wsdl:port` extension elements of WSDL.
- The WSDL binding name is the name with "Binding" prefixed to the port type name. For details about the format of the port type name, see *16.1.3 Mapping the SEI name to the port type*.

(2) SOAP transport and transfer binding

When the `javax.xml.ws.BindingType` annotation is not used for customization, binding is performed using `SOAP 1.1overHTTP` for default mapping.

This means that `http://schemas.xmlsoap.org/soap/http` is specified in the `transport` attribute of the `soap:binding` element that is the child element of the `wsdl:binding` element.

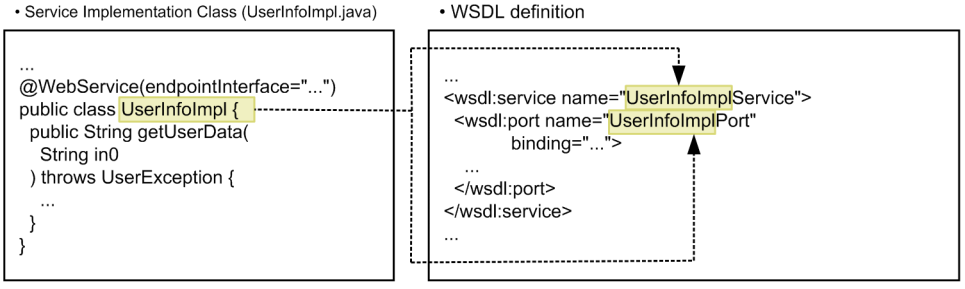
16.1.9 Mapping the Web Service implementation class to the service and port

This subsection describes the mapping of the Web Service Implementation Class to the WSDL service (`name` attribute of the `wsdl:service` element) and port (`name` attribute of the `wsdl:port` element).

(1) Mapping

The Web Service Implementation Class and the WSDL service and port are mapped in accordance with the JAX-WS 2.2 specifications. The following figure shows an example of mapping:

Figure 16-10: Example of mapping the Web Service Implementation Class to the service and port



The following are the rules for mapping the Web Service Implementation Class to the service and port:

- If the WSDL service name is not customized using the `serviceName` element of the `javax.jws.WebService` annotation, the value with "Service" suffixed to the name of the Web Service Implementation Class is assumed as the value of the `name` attribute of the `wsdl:service` element.
- If the WSDL port name is not customized using the `portName` element of the `javax.jws.WebService` annotation, the value with "Port" suffixed to the value of the `name` element of the `javax.jws.WebService` annotation is assumed as the value of the `name` attribute of the `wsdl:port` element.
- If the `portName` attribute and `name` attribute of the `javax.jws.WebService` annotation are not used for customization, the value with "Port" suffixed to the name of the Web Service Implementation Class is assumed as the value of the `name` attribute of the `wsdl:port` element.

(2) Conditions for the Web Service Implementation Class name

In the Web Service Implementation Class name, you can code a string that fulfills all the conditions described in the following table:

Table 16-9: Conditions for strings that can be coded in the Web Service Implementation Class name

No.	Conditions	Examples of invalid strings	Operation when an invalid string is specified
1	Strings using only one-byte alphanumeric characters (0 to 9, A to Z, a to z) and underscore (_)	`Hitachi_service`	The operations might not function properly (error message is not displayed).
2	Strings complying with the naming rules of the Java identifiers provided in the Java language specifications	`abstract`	When the `apt` command is executed, a compilation error occurs and the processing ends. For details, see the *JDK documentation*.

16.1.10 Precautions for mapping from Java to WSDL

This subsection describes the precautions for mapping Java to WSDL.

(1) Deleting the generics type

When you generate a JavaBean (request bean, response bean, and fault bean), the generics type is deleted. The following table describes an example of deleting the generics type:

Table 16-10: Example of deleting the generics type

Before type is deleted	After type is deleted
`T`#	`NumbersData`
`List<E>`	`List<java.lang.Object>`
`List<? extends NumbersData>`	`List<NumbersData>`
`List<? super NumbersData>`	`List< java.lang.Object >`
`Map<K, V>`	`Map< java.lang.Object, java.lang.Object >`
`Map<? extends NumbersKey, ? extends NumbersData>`	`Map<NumbersKey, NumbersData>`
`Map<? super NumbersKey, ? super NumbersData>`	`Map< java.lang.Object, java.lang.Object >`
`Iterator<E>`	`Iterator< java.lang.Object >`
`Iterator<? extends NumbersData>`	`Iterator<NumbersData>`

Before type is deleted	After type is deleted
`Iterator<? super NumbersData>`	`Iterator< java.lang.Object >`
`List<List<? extends NumbersData>>`	`List<List<NumbersData>`

#

The T type definition indicates the case of `<T extends NumbersData>`.

Even if the method argument and return value is customized using the `javax.jws.WebParam` and `javax.jws.WebResult` annotations, the generics type is deleted (customization is also enabled). Also, regardless of whether the method argument and return value is the wrapper style or non-wrapper style, the generics type is deleted. Note that when the method argument and the return value are of the non-wrapper style, the generics type is not deleted.

(2) Supporting JAXB annotations

The Cosminexus JAX-WS functionality complies with Comformance 3.14 of the JAX-WS 2.2 specifications. During the execution of the command, the following JAXB annotations are interpreted as and when required:

- `javax.xml.bind.annotation.XmlAttachmentRef`
- `javax.xml.bind.annotation.XmlList`
- `javax.xml.bind.annotation.adapters.XmlJavaTypeAdapter`
- `javax.xml.bind.annotation.XmlMimeType`

The Cosminexus JAX-WS functionality does not support the MIME binding.

In the annotation processor provided by the JAX-WS functionality of Cosminexus, the `javax.xml.bind.annotation.XmlMimeType` annotation is interpreted for the wrapper style, but not for the non-wrapper style.

If arguments and return values other than the SEI and service implementation class are annotated using these JAXB annotations, the operations might not function properly.

In the annotation processor provided by the Cosminexus JAX-WS functionality, the `javax.xml.bind.annotation.XmlJavaTypeAdapter` annotation and `javax.xml.bind.annotation.XmlMimeType` annotation are interpreted when the argument and return value of the SEI or service implementation class or JavaBean fields are annotated. If a package, interface, or class is annotated, the operations might not function properly.

When you invoke a developed Web Service, if a sub-class of the Web Service arguments and return values is used, an error occurs. To invoke a Web Service normally, the sub-class must be linked by the `javax.xml.bind.annotation.XmlSeeAlso` annotation when you define the SEI and Web Service Implementation Class.

(3) Limitation of the generics type

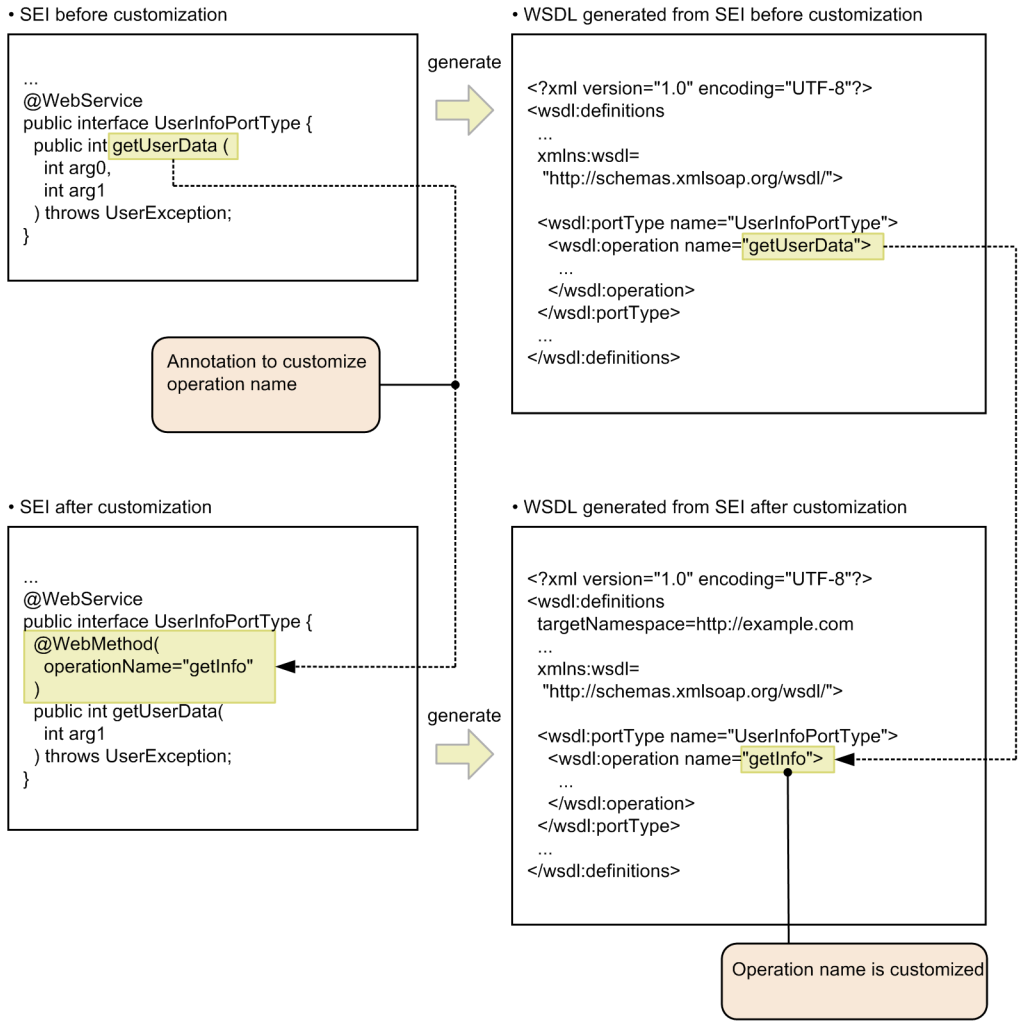
You can use the generics type in the method argument and the return value for the wrapper style, but not for the non-wrapper style.

16.2 Customized mapping from Java to WSDL

You can customize the mapping of Java to WSDL by using annotations.

The following figure shows an example of customization using annotations:

Figure 16-11: Example of customization using annotations



When you execute the `apt` command or the `cjwsgen` command, the `javax.xml.ws.WebEndpoint` and `javax.xml.ws.WebServiceClient` annotation automatically given by the `cjwsimport` command are ignored (warning message is not output). Also, if an un-supported annotation is specified, the annotation is ignored. At this time, if the annotation is not supported even in the annotation processor provided by functionality other than the Cosminexus JAX-WS functionality, a warning message of the `apt` command is output.

The annotations include annotations that can be defined in SEI, annotations that can be defined in the Web Service Implementation Class, and the annotations that can be defined in both. However, if the `endpointInterface` element of the `javax.jws.WebService` annotation is not used, the abstract information is extracted from the information of the Web Service Implementation Class and it is considered that implicit SEI is present. Only in this case, the definition of annotations defined in SEI is allowed in the Web Service Implementation Class.

Even if the annotation element is explicitly customized using the same value as the default value, the annotation is processed as if the element value is not specified.

16.2.1 List of annotations

The following table describes the annotations that can be used for customization and the auto-generated annotations. For details about the annotations, see the *JAX-WS 2.2 specifications*.

Table 16-11: List of annotations of JAX-WS used for customization

No.	Annotations		Description	Definition
	Annotation name	Element name		
1	com.sun.xml.ws.developer.StreamingAttachment	dir	This is a name of the directory that generates a temporary file when you want to output the MIME body that is included in the SOAP message containing the received attachments as temporary files.	Y
		parseEagerly	Sets whether to perform the detailed parsing for the SOAP message that contains the received attachments.	Y
		memoryThreshold	This is the threshold to decide whether to extract the MIME body that is included in the SOAP message containing the received attachments, in the memory.	Y
2	javax.jws.HandlerChain	file	URL and relative path indicating the location of the handler chain file.	Y
3	javax.jws.Oneway	--	Annotation to be specified when using the one-way operation in Web Services	Y
4	javax.jws.soap.SOAPBinding	parameterStyle	The style of the method parameters (Document wrapped or Document Bare style).	Y
		style	The encoding style of the message. In the Cosminexus JAX-WS functionality, only document can be specified.	Y
		use	The format style of the message. In the Cosminexus JAX-WS functionality, only literal can be specified.	Y
5	javax.jws.WebMethod	action	String that determines the SOAP action.	Y
		exclude	Specifies whether the Web method can be made public.	Y
		operationName	The wsdl:operation element name matching with the method.	Y
6	javax.jws.WebParam	header	Indicates whether the parameter is obtained from the message header.	Y
		mode	Direction of the flow of parameters (in, inout, and out).	Y
		name	Local name of the XML element indicating the parameter.	Y
		partName	The wsdl:part element name indicating the parameter.	Y
		targetNamespace	XML name space name.	Y
7	javax.jws.WebResult	header	Indicates whether to obtain the return value from the message header.	Y

No.	Annotations		Description	Definition
	Annotation name	Element name		
7	javax.jws.WebResult	name	Local name of the XML element indicating the return value.	Y
		partName	The wsdl:part element name indicating the return value.	Y
		targetNamespace	XML name space name.	Y
8	javax.jws.WebService	endpointInterface	The fully qualified name of SEI that defines the abstract Web Service contract of the Web Service.	Y
		name	The wsdl:portType element name indicating the Web Service name.	Y
		portName	The wsdl:port element name indicating the Web Service port name.	Y
		serviceName	The wsdl:service element name indicating the service name of the Web Service.	Y
		targetNamespace	The name space name of the Web Service.	Y
		wsdlLocation	URL indicating WSDL of the Web Service.	Y#1,#3
9	javax.xml.ws.Action	fault	This is a value of the wsa:Action element of the addressing header. This element is required, when the Web Service sends a fault message	Y
		input	This is a value of the wsa:Action element of the addressing header. This element is required, when the Web Service receives a request message.	Y
		output	This is a value of the wsa:Action element of the addressing header. This element is required, when the Web Service sends a response message.	Y
10	javax.xml.ws.BindingType	value	Binding used for making SEI public.	Y
11	javax.xml.ws.FaultAction	className	This is an exception class name.	Y
		value	This is a value of the wsa:Action element of the addressing header. This element is required, when sending the fault message corresponding to the exception class name that is specified in className from the Web Service.	Y
12	javax.xml.ws.RequestWrapper	className	The request bean class name.	Y
		localName	The local name of the target element.	Y
		targetNamespace	The name space name of the target element.	Y
		partName	The part name of the input message that references the request wrapper element.	Y
13	javax.xml.ws.ResponseWrapper	className	The response bean class name.	Y

No.	Annotations		Description	Definition
	Annotation name	Element name		
13	javax.xml.ws.ResponseWrapper	`localName`	The local name of the target element.	Y
		`targetNamespace`	The name space name of the target element.	Y
		`partName`	The part name of the output message that references the response `wrapper` element.	Y
14	javax.xml.ws.ServiceMode	`value`	This is the service mode.	Y
15	javax.xml.ws.soap.Addressing	`enabled`	Specifies whether to enable the addressing functionality.	Y
		`required`	Specifies whether the addressing header is required, when invoking the Web Service.	
		`response`	Sets whether to make the anonymous or non-anonymous response mandatory for the endpoint.	
16	javax.xml.ws.soap.MTOM	`enabled`	Sets whether to use the attachments with the MTOM/XOP specification format.	Y
		`threshold`	Sets the threshold for sending the binary data as attachments with the MTOM/XOP specification format.	Y
17	javax.xml.ws.WebFault	`faultBean`	The fault bean class name.	Y
		`name`	The local name of the target element.	Y
		`targetNamespace`	The name space name of the target element.	Y
		`messageName`	The fault message name.	Y
18	javax.xml.ws.WebServiceProvider	`targetNamespace`	This is the name space name of the Web Service.	Y
		`portName`	This is the port name of the Web Service.	Y
		`serviceName`	This is the service name of the Web Service.	Y
		`wsdlLocation`	This is the URL indicating the WSDL of the Web Service.	Y[#3]
19	javax.xml.ws.RespectBinding[#1]	`enabled`	Indicates whether the contents of the `wsdl:binding` element are enabled.	Y[#1]
20	javax.xml.ws.WebEndpoint[#2]	`name`	The local name of port.	N[#1]
21	javax.xml.ws.WebServiceClient[#2]	`name`	The local name of Web Service.	N[#2]
		`targetNamespace`	The name space name of the Web Service.	N[#2]
		`wsdlLocation`	URL indicating WSDL of the Web Service.	N[#2,#3]

Legend:

--: Indicates that the annotation does not have any element

Y: Indicates that you can specify an annotation and an element.

N: Indicates that you cannot specify an annotation and an element (not supported).

#1

The specified value is ignored (the warning message is not displayed).

#2

Cannot be specified since the annotation is automatically given to the class generated from WSDL.

#3

The annotations do not support mapping in the catalog functionality.

The following is the list of the JAXB annotations that can be defined with the Web Services of Cosminexus. For details on each annotation, see *Standard specifications of JAXB*.

- `javax.xml.bind.annotation.XmlAttachmentRef`
- `javax.xml.bind.annotation.XmlList`
- `javax.xml.bind.annotation.adapters.XmlJavaTypeAdapter`
- `javax.xml.bind.annotation.XmlMimeType`
- `javax.xml.bind.annotation.XmlSeeAlso`
- `javax.xml.bind.annotation.XmlElement`
- `javax.xml.bind.annotation.XmlType`

Of the annotations in the JSR-181 specifications and JAX-WS 2.2 specifications, you cannot use the annotations indicated in the following table. If the following annotations are specified, the operations might not function properly.

Table 16–12: List of un-supported annotations

No.	Annotations	Remarks
1	`javax.xml.ws.WebServiceRefs`	The functionality related to JSR109 specifications is not supported.
2	`javax.xml.ws.spi.WebServiceFeatureAnnotation`	The API (method that takes the features in the parameter) for using this annotation is not supported.
3	`javax.jws.soap.InitParam`	Not recommended in JSR181 specifications (will be removed).
4	`javax.jws.soap.SOAPMessageHandler`	Not recommended in JSR181 specifications (will be removed).
5	`javax.jws.soap.SOAPMessageHandlers`	Not recommended in JSR181 specifications (will be removed).

16.2.2 com.sun.xml.ws.developer.StreamingAttachment annotation

The description of the `com.sun.xml.ws.developer.StreamingAttachment` annotation is as follows.

Web Services

This annotation is to be specified in the Web Service using streaming. You can specify the annotation only in the Web Service Implementation Class. If the annotation is specified in SEI, the specification is ignored. If the annotation is specified in the Provider Implementation Class, the operation is not guaranteed.

Web Services client

This annotation is to be specified in the `setter` method or a field that injects the port. For details, see *10.21.1(4) Enabling features*. The annotation will be ignored if you specify the annotation in any other fields or methods.

When creating the Web Service using streaming with the help of the skeleton class of the Web Service Implementation Class generated by the `cjwsimport` command, the `com.sun.xml.ws.developer.StreamingAttachment` annotation is not mapped in the Web Service Implementation Class. Therefore, you must specify the `com.sun.xml.ws.developer.StreamingAttachment` annotation. Note that even if the `com.sun.xml.ws.developer.StreamingAttachment` annotation is specified in the Web Service Implementation Class, an element or attribute that indicates the usage of streaming does not appear in the WSDL file issued by the Web Service JAX-WS engine or WSDL file generated by the `cjwsgen` command.

The com.sun.xml.ws.developer.StreamingAttachment annotation is referenced when starting the Web Services. Therefore, this annotation is not interpreted when executing the apt command and the cjwsgen command.

The following figure shows an example of using the com.sun.xml.ws.developer.StreamingAttachment annotation:

Figure 16-12: Example of using the com.sun.xml.ws.developer.StreamingAttachment annotation (Web Services)

• Web Service Implementation Class

```
.....
@StreamingAttachment(dir="C:/TMP", parseEagerly=true,
memoryThreshold=50000L)
@WebService(endpointInterface =
"jaxwstp.example.service.ExamplePortType", targetNamespace = "http://
service.example.jaxwstp/", serviceName = "ExampleService", portName =
"ExamplePort")
public class ExampleBinding implements ExamplePortType {
```

(1) dir element (com.sun.xml.ws.developer.StreamingAttachment)

The dir element specifies a directory that is used when you want to output the MIME body that is included in the SOAP message of MIME Multipart/Related structure containing the received attachments when streaming is used in the Web Service, as a temporary file. The default value is a null character ("").

When a null character ("") or null is specified as a default value of the dir element, the output destination for the temporary files is the default temporary file directory for Java (the value corresponding to the java.io.tmpdir key in the system property). Also, if you specify a directory name that does not exist, or a directory name without access rights, or an existing file name, a message is output when starting the Web Service, and the MIME body that is included in the SOAP message of MIME Multipart/Related structure containing the received attachments is extracted to the memory (KDJW10026-W).

(2) parseEagerly element (com.sun.xml.ws.developer.StreamingAttachment)

The parseEagerly element specifies whether to perform the detailed parsing for the SOAP message of MIME Multipart/Related structure containing the received attachments when streaming is used in the Web Service. The default value is false.

(3) memoryThreshold element (com.sun.xml.ws.developer.StreamingAttachment)

The memoryThreshold element specifies the threshold (unit: byte) to decide whether to output the MIME body that is included in the SOAP message of MIME Multipart/Related structure containing the received attachments, as temporary files, when streaming is used in the Web Service. In the memoryThreshold element, specify a value greater than 16KB (16384L) or -1. The operation is not guaranteed if any other value is specified. When -1 is specified, the MIME body is always extracted to the memory. The default value is 1048576L.

For deciding whether to extract the MIME body that is included in the received SOAP message of the MIME Multipart/Related structure to the memory, see *32.3 Temporary files (streaming)* .

16.2.3 javax.jws.HandlerChain annotation

The javax.jws.HandlerChain annotation is enabled when defined in the class or interface declaration.

If the javax.jws.HandlerChain annotation is specified at the same time as the SEI and Web Service Implementation Class, the annotation specified in the Web Service Implementation Class gets priority. At this time, a warning message is output to the standard output and log and the processing continues (KDJW61076-W).

(1) file element (javax.jws.HandlerChain)

You specify the handler chain setup file in the file element. Use the relative path from the class annotated with the `javax.jws.HandlerChain` annotation or the interface to specify the file. In the Cosminexus JAX-WS functionality, the specification in the URL format is not supported.

If you specify a path that cannot be referenced and opened, an error message is output to the standard error output and logs during Web Service initialization (KDJW00010-E).

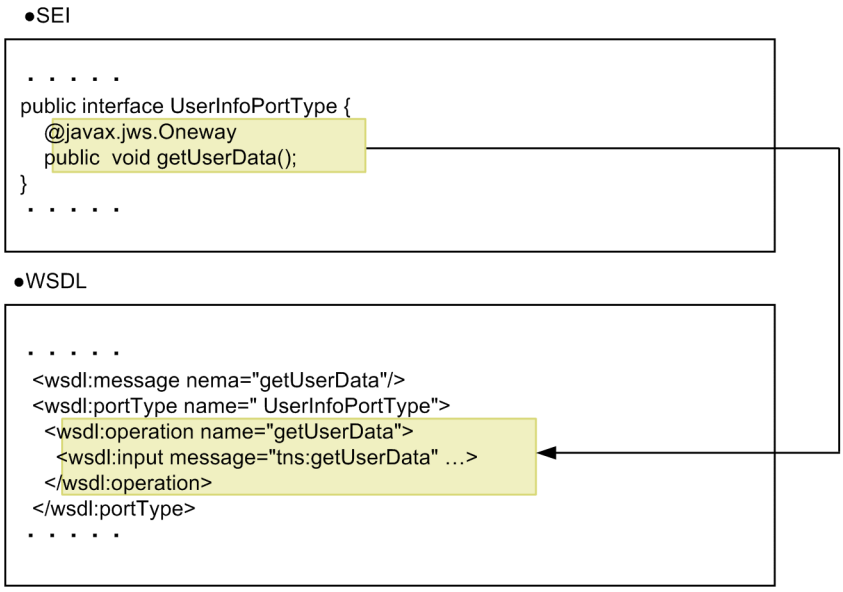
Note that the `file` element is a value referred when the Web Service is invoked, so the element is not interpreted when the `apt` command or the `cjwsgen` command is executed. For details about the examples of using the `file` elements, see *36.9.1 Setting the handler chain in Web Services*.

16.2.4 javax.jws.Oneway annotation

Specify the `javax.jws.Oneway` annotation when using the one-way operation in Web Services. The method of Web Services Implementation Class for which the `javax.jws.Oneway` annotation is annotated, has only input messages and no output messages.

The following figure shows an example of mapping by using the `javax.jws.Oneway` annotation.

Figure 16−13: Example of mapping by using the javax.jws.Oneway annotation



A Web Services Implementation Class method annotated with the `javax.jws.Oneway` annotation must fulfill the following conditions:

- Specify `void` for the type of the return value. An error message (KDJW61108-E) is output to the standard error output and log if a type other than `void` is specified.
- Specify only one `in` parameter other than the SOAP header for Web Services of the BARE style. An error message (KDJW61111-E) is output to the standard error output and log if zero, or two or more `in` parameters are specified.
- Use `javax.jws.WebParam.Mode.IN` when using the Mode element of the `javax.jws.WebParam` annotation. An error message (KDJW61110-E) is output to the standard error output and log if any other annotation is used.
- Confirm that you declared no exceptions other than `java.lang.RuntimeException` and `java.rmi.RemoteException` and their subclasses in the method of Web Services Implementation Class. An error (KDJW61109-E) is output to the standard error output and log if one or more exceptions are specified.

- Do not use a holder type to specify the type of an argument. An error message is wrapped and output (KDJW79999-E) to the standard error output and log if a holder type is specified. The message KDJW20041-E is output.

Notes when using the `javax.jws.Oneway` annotation are as follows:

- Do not combine the `javax.jws.Oneway` and `javax.xml.ws.soap.Addressing` annotations in Web Services Implementation Class. The response messages do not exist for the one-way operations. The addressing functionality, however, specifies destinations for the response messages. Therefore, the operation is not guaranteed when the aforementioned annotations are concurrently used.
- Because the compliant SOAP messages do not exist for the one-way operations, implement the method annotated with the `javax.jws.Oneway` annotation of Web Services Implementation Class in such a way that an exception is not explicitly thrown.

16.2.5 javax.jws.soap.SOAPBinding annotation

The javax.jws.soap.SOAPBinding annotation can be used to customize the mapping of the SOAP Message protocols.

(1) style element (javax.jws.soap.SOAPBinding)

Only the DOCUMENT/LITERAL style can be used in the annotation processor provided in the Cosminexus JAX-WS functionality. Therefore, if `SOAPBinding.Style.RPC` is specified in the `style` element, an error message is output to the standard error output and logs (KDJW61063-E). Also, if a value other than `SOAPBinding.Style.DOCUMENT` and `SOAPBinding.Style.RPC` is specified, a compilation error occurs during the execution of the `apt` command.

If the `javax.jws.soap.SOAPBinding` annotation is specified in the class or interface declaration and the method declaration at the same time, the value specified in the method declaration gets priority.

(2) use element (javax.jws.soap.SOAPBinding)

Only the DOCUMENT/LITERAL style can be used in the annotation processor provided in the Cosminexus JAX-WS functionality. Therefore, if SOAPBinding.Use.ENCODED is specified in the `use` element, an error message is output to the log and the error is notified to the `apt` command (KDJW61063-E). Also, if a value other than `SOAPBinding.Use.LITERAL` and `SOAPBinding.Use.ENCODED` is specified, a compilation error occurs during the execution of the `apt` command.

(3) parameterStyle element (javax.jws.soap.SOAPBinding)

For the wrapper style, you specify `SOAPBinding.ParameterStyle.WRAPPED` in the `parameterStyle` element. Also, for the non-wrapper style, you specify `SOAPBinding.ParameterStyle.BARE` in the `parameterStyle` element.

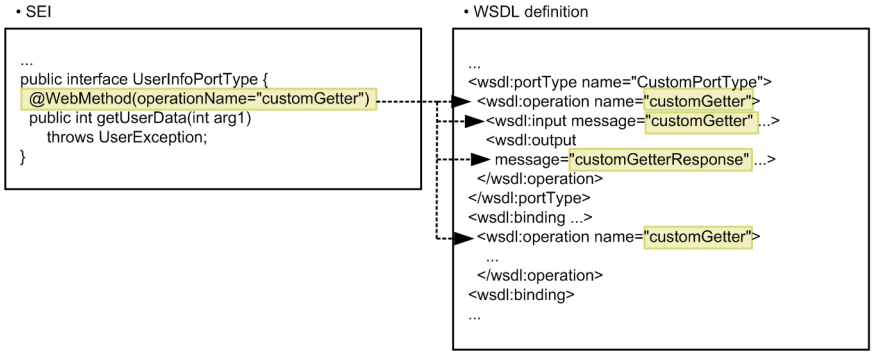
If a value other than `SOAPBinding.ParameterStyle.WRAPPED` and `SOAPBinding.ParameterStyle.BARE` is specified in the `parameterStyle` element, a compilation error occurs during the execution of the `apt` command.

16.2.6 javax.jws.WebMethod annotation

The `javax.jws.WebMethod` annotation can be used to customize the mapping of operations.

The following figure shows an example of customization with the `javax.jws.WebMethod` annotation:

Figure 16-14: Example of customization with the javax.jws.WebMethod annotation



(1) exclude element (javax.jws.WebMethod)

Of the public methods in the Web Service Implementation Class, the public methods that you do not want to make public as an operation can be excluded from the mapped operations by annotating the method with the `javax.jws.WebMethod` annotation where the element value of the `exclude` element is `true`. For methods in an inherited parent class, that method is overridden and can be excluded from the mapped operations by annotating the method with the `javax.jws.WebMethod` annotation where the element value of the `exclude` element is `true`.

The precautions for specifying the `exclude` element are as follows:

- If public methods do not exist due to exclusion from the mapped operations, an error message is output to the standard error output and log and the processing ends (KDJW61093-E).
- If `true` is specified in the element value of this element in the Web Service Implementation Class, other elements cannot be specified in the `javax.jws.WebMethod` annotation of the Web Service Implementation Class. If other elements are specified, an error message is output to the standard error output and logs (KDJW61010-E).
- If this element is specified in SEI, specify `false` in the element value. If you specify `true`, an error message is output to the standard error output and logs (KDJW61021-E).
- If the parent class methods are overridden in the Web Service Implementation Class and if this element is specified in both the parent class and the Web Service Implementation Class (child class), the element specified in the Web Service Implementation Class (child class) gets priority.
- If the `endpointInterface` element of the `javax.jws.WebService` annotation is not specified in the Web Service Implementation Class, all the public methods, to which the static modifier or final modifier of the Web Service Implementation Class is not applied, are mapped to an implicit SEI. Also, even if one `javax.jws.WebMethod` annotation where the `exclude` element is not `true` is specified in the public methods, to which the static modifier or final modifier of the Web Service Implementation Class is not applied, the methods of the Web Service Implementation Class that fulfill the conditions in the following table are mapped to an implicit SEI.

  Table 16-13: Specification of annotations and mapping to implicit SEI

No.	Methods of the Web Service Implementation Class (WebMethod annotation)	Methods in implicit SEI
1	No	Not mapped.
2	Yes (without `exclude` element)	Not mapped.
3	`exclude=false`	Not mapped.
4	`exclude=true`	Not mapped.

- If the `endpointInterface` element of the `javax.jws.WebService` annotation is not specified in the Web Service Implementation Class and if the `javax.jws.WebMethod` annotation where the `exclude` element is not `true` is specified in the public methods, to which the static modifier or final modifier of the Web Service Implementation Class has been applied, the error message (KDJW61102-E) is output to the standard error output and log.

(2) OperationName element (javax.jws.WebMethod)

The `operationName` element is specified when you customize the mapping for the operation name. If you specify the element value in the `operationName` element, you can customize the message name that has the operation name in the default mapping value. For the non-wrapper style, you can also customize the global element name and part name.

The precautions for specifying the `operationName` element are as follows:

- Specify the `operationName` element using one-byte alphanumeric characters and underscore (_). If other characters are specified, the operations might not function properly (error message is not output).
- Specify a value complying with the Java identifier naming rules provided in the Java language specifications. If the value does not comply with the Java identifier naming rules, a compilation error occurs during the execution of the `cjwsimport` command in the Web Service client development.
- For the wrapper style, the `localName` element of the `javax.xml.ws.RequestWrapper` annotation must be the same as the 'operation name'. If the element is not the same, an error message is output to the standard error output and logs (KDJW61047-E). However, if the `javax.xml.ws.RequestWrapper` annotation does not exist or if a blank character ("") is specified in the `localName` element of the `javax.xml.ws.RequestWrapper` annotation, a comparison is not done to check whether the names are the same and hence no message is output.

(3) action element (javax.jws.WebMethod)

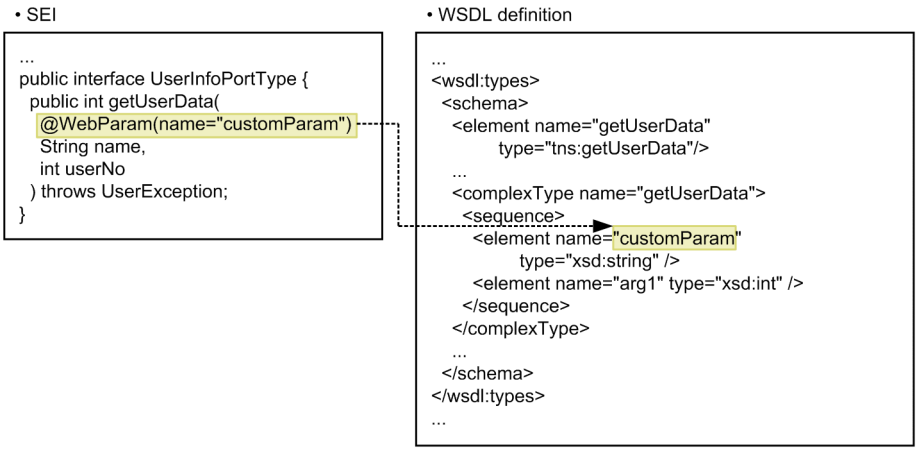
The `action` element is mapped to the action of the SOAP operation. In the `action` element, you can specify characters satisfying `xsd:anyURI` defined in RFC 2396, but the specified value is ignored at runtime.

16.2.7 javax.jws.WebParam annotation

The `javax.jws.WebParam` annotation can be used to customize the mapping of arguments.

The following figure shows an example of customization with the `javax.jws.WebParam` annotation:

Figure 16－15: Example of customization with the javax.jws.WebParam annotation



(1) header element (javax.jws.WebParam)

To map an argument as a `header` parameter, specify `true` in the element value of the `header` element.

You can specify the `header` element in the non-wrapper style. If specified in the wrapper style, an error message is output to the standard error output and logs (KDJW61037-E).

(2) name element (javax.jws.WebParam)

The `name` element is used to customize the child element name of the wrapper element that is mapped from the argument in the wrapper style. For the non-wrapper style, the `name` element is used to customize the local name of the global element that is mapped from the argument. If the `partName` element is not specified in the non-wrapper style, you can also customize the part name by specifying the element value of the `name` element.

The precautions for specifying the `name` element are as follows:

- Specify the `name` element using one-byte alphanumeric characters and underscore (_). If other characters are specified, the operations might not function properly (error message is not output).
- For the wrapper style, specify a value complying with the Java identifier naming rules provided in the Java language specifications. If the value does not comply with the Java identifier naming rules, a compilation error occurs during the execution of the `cjwsimport` command in the Web Service client development.

(3) partName element (javax.jws.WebParam)

You specify the `partName` element to customize the mapping of the part names.

The precautions for specifying the `partName` element are as follows:

- The `partName` element is enabled in the non-wrapper style. If the `partName` element is specified in the wrapper style, the element is ignored.
- If the `partName` element and `name` element are specified at the same time, the enabled elements are as follows:
  - In the wrapper style: The value of the `name` element is enabled.
  - In the non-wrapper style: The value of the `partName` element is enabled.
- Specify the `partName` element using one-byte alphanumeric characters and underscore (_). If other characters are specified, the operations might not function properly (error message is not output).
- For the non-wrapper style, specify a value complying with the Java identifier naming rules provided in the Java language specifications. If the value does not comply with the Java identifier naming rules, a compilation error occurs during the execution of the `cjwsimport` command in the Web Service client development.

(4) mode element (javax.jws.WebParam)

In the `mode` element, you specify the value indicating the direction of the flow of parameters. The values that can be specified are as follows:

- `WebParam.Mode.IN`
- `WebParam.Mode.OUT`
- `WebParam.Mode.INOUT`

(5) targetNamespace element (javax.jws.WebParam)

Use the `targetNamespace` element to customize the name space of the global element that is mapped from the argument.

In the `targetNamespace` element, the `http://` or `urn:` protocol is specified as a name space. The following name space formats and strings can be specified:

- **Protocol**
  Code the name space protocol using the `http://` or `urn:` protocol. If a protocol other than `http://` or `urn:` (such as `https://`, `file://`) is coded, an error message is output to the standard error output and logs (KDJW61087-E).
  Also, if you code the name space using the relative path, an error message is output to the standard error output and logs (KDJW61088-E).
- **Name space coding format**
  The following formats cannot be coded in the name space. If the following formats are used for coding, an error message is output to the standard error output and logs (KDJW61089-E):

- Query string (example) `http://example.com/?a=b`
- Anchor (example) `http://example.com/index.html#anchor`
- Port number (example) `http://example.com:8080/`
- User name or password (example) `http://user:password@example.com`

- **Strings that can be coded**

  In the segments separated using the separation characters, forward slash (/) or period (.), you can code a string that fulfills all the conditions described in the following table:

  Table 16-14: Conditions for strings that can be coded in the name space (javax.jws.WebParam)

No.	Conditions	Examples of invalid strings	Operation when an invalid string is specified

1	Strings using only one-byte alphanumeric characters (0 to 9, A to Z, a to z)	`http://Hitachi.com`
		`http://133.145.224.19/`
		`http://[1080:2C14;D30:BA04:275:806:270C:418A]/`
			The operations might not function properly (error message is not displayed).

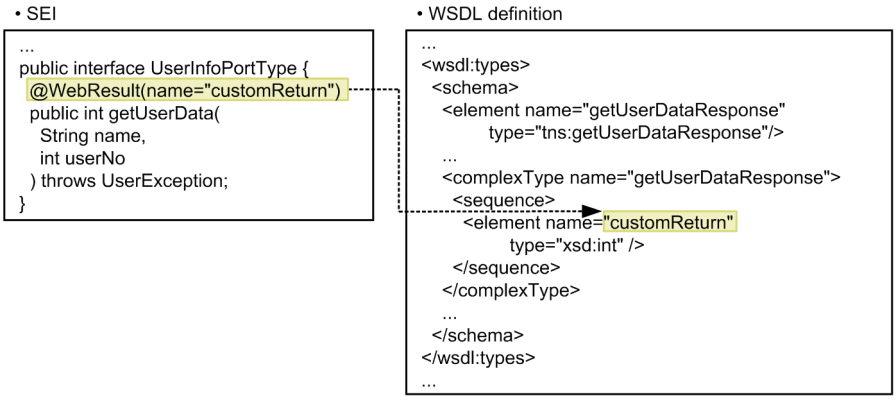
2	Strings other than Java reserved words	`http://hitachi.com/abstract`	The operations might not function properly.
3	Strings that do not begin with numeric characters	`http://1hitachi.com`	

16.2.8 javax.jws.WebResult annotation

Use the `javax.jws.WebResult` annotation to customize the mapping of the return values.

The following figure shows an example of customization with the `javax.jws.WebResult` annotation:

Figure 16-16: Example of customization with the `javax.jws.WebResult` annotation



(1) header element (javax.jws.WebResult)

To map a return value as a `header` parameter, specify `true` in the element value of the `header` element.

You can specify the `header` element in the non-wrapper style. In the wrapper style, if `true` is specified in the element value of the `header` element, an error message is output to the standard error output and logs (KDJW61038-E).

(2) name element (javax.jws.WebResult)

The `name` element is used to customize the child element name of the wrapper element that is mapped from the return value in the wrapper style. For the non-wrapper style, the `name` element is used to customize the local name of the

global element that is mapped from the argument. If the partName element is not specified in the non-wrapper style, you can also customize the part name by specifying the element value of the name element.

The precautions for specifying the name element are as follows:

- Specify the name element using one-byte alphanumeric characters and underscore (_). If other characters are specified, the operations might not function properly (error message is not output).
- For the wrapper style, specify a value complying with the Java identifier naming rules provided in the Java language specifications. If the value does not comply with the Java identifier naming rules, a compilation error might occur during the execution of the cjwsimport command in the Web Service client development.
- In a wrapper style method with 254 arguments, if the name element is customized to a value other than "return", the number of arguments becomes 255 during the execution of the cjwsimport command and a compilation error might occur.

(3) partName element (javax.jws.WebResult)

You specify the partName element to customize the mapping of the part names.

The precautions for specifying the partName element are as follows:

- The partName element is enabled in the non-wrapper style. If the partName element is specified in the wrapper style, the element is ignored.
- If the partName element and name element are specified at the same time, the enabled elements are as follows:
  - In the wrapper style: The value of the name element is enabled.
  - In the non-wrapper style: The value of the partName element is enabled.
- Specify the partName element using one-byte alphanumeric characters and underscore (_). If other characters are specified, the operations might not function properly (error message is not output).
- For the non-wrapper style, specify a value complying with the Java identifier naming rules provided in the Java language specifications. If the value does not comply with the Java identifier naming rules, a compilation error occurs during the execution of the cjwsimport command in the Web Service client development.
- In a wrapper style method with 254 arguments, if the name element is customized to a value other than "return", the number of arguments becomes 255 during the execution of the cjwsimport command and a compilation error might occur.

(4) targetNamespace element (javax.jws.WebResult)

Use the targetNamespace element to customize the name space of the global element that is mapped from the return value.

In the targetNamespace element, the http:// or urn: protocol is specified as a name space. The following name space formats and strings can be specified:

- **Protocol**

  Code the name space protocol using the http:// or urn: protocol. If a protocol other than http:// or urn: (such as https://, file://) is coded, an error message is output to the standard error output and logs (KDJW61090-E).

  Also, if you code the name space using the relative path, an error message is output to the standard error output and logs (KDJW61091-E).
- **Name space coding format**

  The following formats cannot be coded in the name space. If the following formats are used for coding, an error message is output to the standard error output and logs (KDJW61092-E):
  - Query string (example) http://example.com/?a=b
  - Anchor (example) http://example.com/index.html#anchor
  - Port number (example) http://example.com:8080/
  - User name/ password (example) http://user:password@example.com
- **Strings that can be coded**

In the segments separated using the separation characters, forward slash (/) or period (.), you can code a string that fulfills all the conditions described in the following table:

Table 16-15: Conditions for the strings that can be coded in the name space (javax.jws.WebResult)

No.	Condition	Examples of invalid string	Operation when an invalid string is specified

1	Strings using only one-byte alphanumeric characters (0 to 9, A to Z, a to z)	`http://hitachi.com`
		`http://133.145.224.19/`
		`http://[1080:2C14;D30:BA04:275:806:270C:418A]/`
			The operations might not function properly (error message is not displayed).

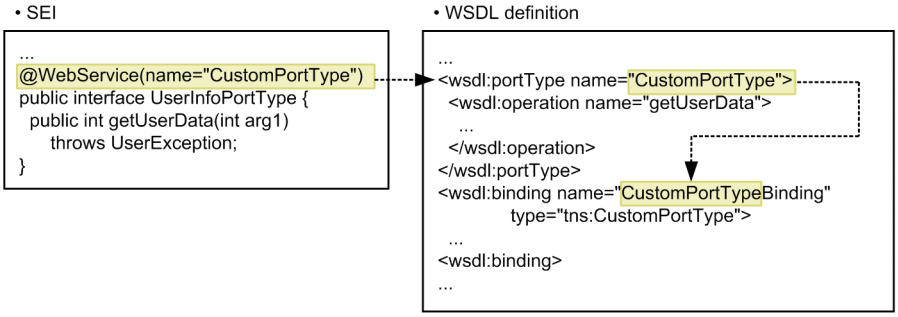
2	Strings other than Java reserved words	`http://hitachi.com/abstract`	The operations might not function properly.
3	Strings that do not begin with numeric characters	`http://1hitachi.com`	

16.2.9 javax.jws.WebService annotation

The `javax.jws.WebService` annotation is mandatory in the SEI and Web Service Implementation Class.

The following figure shows an example of customization with the `javax.jws.WebService` annotation:

Figure 16-17: Example of customization with the javax.jws.WebService annotation



Note that you can specify only one annotation between the `javax.xml.ws.WebServiceProvider` and `javax.jws.WebService` annotations. If you specify the `javax.xml.ws.WebServiceProvider` and `javax.jws.WebService annotation` annotations together, an error message will output to the standard error output and logs (KDJW61098-E).

(1) targetNamespace element (javax.jws.WebService)

If the `targetNamespace` element is specified in SEI, the specified name space is enabled for the `types` element, `message` element, and `portType` element.

If this element is specified in the Web Service Implementation Class, the specified name space is enabled for the `binding` element and `service` element. If the `endpointInterface` element of the `javax.jws.WebService` annotation is not used in the Web Service Implementation Class, the same name space is assumed to be specified in both implicit SEI and the Web Service Implementation Class.

In the `targetNamespace` element, the `http://` or `urn:` protocol is specified as a name space. The following name space formats and strings can be specified:

- **Protocol**

  Code the name space protocol using the `http://` or `urn:` protocol. If a protocol other than `http://` or `urn:` (such as `https://`, `file://`) is coded, an error message is output to the standard error output and logs (KDJW61005-E).

Also, if you code the name space using the relative path, an error message is output to the standard error output and logs (KDJW61006-E).

- **Name space coding format**

  The following format cannot be coded in the name space. If the following formats are used for coding, an error message is output to the standard error output and logs (KDJW61007-E):

  - Query string (example) `http://example.com/?a=b`
  - Anchor (example) `http://example.com/index.html#anchor`
  - Port number (example) `http://example.com:8080/`
  - User name/ password (example) `http://user:password@example.com`

- **Strings that can be coded**

  In the segments separated using the separation characters, forward slash (/) or period (.), you can code a string that fulfills all the conditions described in the following table:

  Table 16-16: Conditions for the strings that can be coded in the name space (javax.jws.WebService)

No.	Conditions	Examples of invalid strings	Operation when an invalid string is specified

1	Strings using only one-byte alphanumeric characters (0 to 9, A to Z, a to z)	`http://hitachi.com`
		`http://133.145.224.19/`
		`http://[1080:2C14;D30:BA04:275:806:270C:418A]/`
			The operations might not function properly (error message is not displayed).

2	Strings other than Java reserved words	`http://hitachi.com/abstract`	The operations might not function properly.
3	Strings that do not begin with numeric characters	`http://1hitachi.com`	

(2) endpointInterface element (javax.jws.WebService)

The precautions for specifying the `endpointInterface` element are as follows:

- The `endpointInterface` element specifies SEI with the `javax.jws.WebService` annotation using the `javax.jws.WebService` annotation of the Web Service Implementation Class. If the class is specified, an error message is output to the standard error output and logs (KDJW61009-E). If the specified SEI is not found, an error message is output to the standard error output and logs (KDJW61028-E).
- If the `endpointInterface` element is specified in the `javax.jws.WebService` annotation of the Web Service Implementation Class, the `name` element of the `javax.jws.WebService` annotation cannot be specified at the same time. If specified at the same time, an error message is output to the standard error output and logs (KDJW61019-E).
- In SEI, the `endpointInterface` element of the `javax.jws.WebService` annotation cannot be specified. If specified, an error message is output to the standard error output and logs (KDJW61024-E).
- If the `endpointInterface` element is specified in the `javax.jws.WebService` annotation of the Web Service Implementation Class, the following annotations cannot be specified in the Web Service Implementation Class. If specified, an error message is output to the standard error output and logs (KDJW61075-E).
  - `javax.jws.WebMethod`
  - `javax.jws.WebParam`
  - `javax.jws.Oneway`
  - `javax.jws.WebResult`
  - `javax.jws.SOAPBinding`

(3) name element (javax.jws.WebService)

The `name` element is specified to customize the mapping of the port type names. If the element value is specified in the `name` element, you can also customize all the elements that have the port type name in the default mapping value.

The precautions for specifying the `name` element are as follows:

- When the `name` element is used in the `javax.jws.WebService` annotation of SEI, specify the `name` element using one-byte alphanumeric characters and underscore (_). If other characters are specified, the operations might not function properly (error message is not output).
- If the `endpointInterface` element is specified in the `javax.jws.WebService` annotation of the Web Service Implementation Class, the `name` element of the `javax.jws.WebService` annotation cannot be specified at the same time in the Web Service Implementation Class. If specified at the same time, an error message is output to the standard error output and logs (KDJW61019-E).

(4) serviceName element (javax.jws.WebService)

You specify the `serviceName` element to customize the mapping of the service names.

The precautions for specifying the `serviceName` element are as follows:

- The `serviceName` element of the `javax.jws.WebService` annotation cannot be specified in SEI. If specified, an error message is output to the standard error output and logs (KDJW61023-E).
- If the `serviceName` element is used in the `javax.jws.WebService` annotation of the Web Service Implementation Class, specify the `serviceName` element using one-byte alphanumeric characters and underscore (_). If other characters are specified, the operations might not function properly.
- Specify a value complying with the Java identifier naming rules provided in the Java language specifications. If the value does not comply with the Java identifier naming rules, a compilation error occurs during the execution of the `cjwsimport` command in the Web Service client development.

(5) portName element (javax.jws.WebService)

You specify the `portName` element to customize the mapping of the port names.

- You cannot specify the `portName` element of the `javax.jws.WebService` annotation in SEI. If specified, an error message is output to the standard error output and logs (KDJW61022-E).
- If the `portName` element is used in the `javax.jws.WebService` annotation of the Web Service Implementation Class, specify the `portName` element using the one-byte alphanumeric characters and underscore (_).If other characters are specified, the operations might not function properly.

(6) wsdlLocation element (javax.jws.WebService)

The `wsdlLocation` element is the value referenced when the Web Service is invoked, so the element is not interpreted when the `apt` command or the `cjwsgen` command is executed.

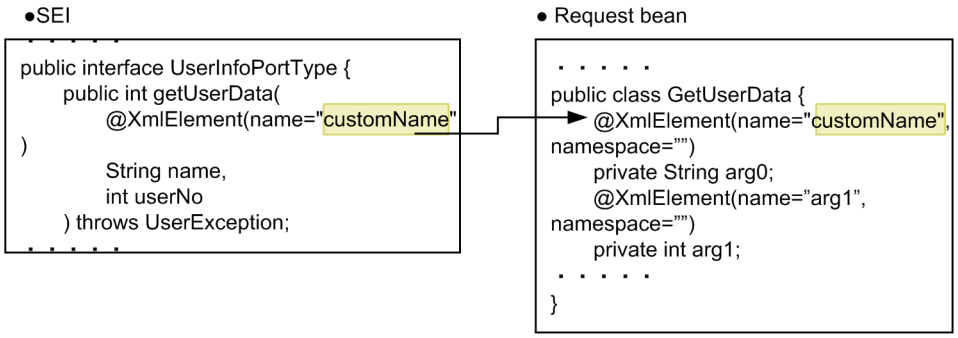
16.2.10 javax.xml.bind.annotation.XmlElement annotation

Specify the `javax.xml.bind.annotation.XmlElement` annotation in an argument or a return value of an SEI service method. When specified, the annotation is mapped to the `javax.xml.bind.annotation.XmlElement` annotation of the property of the generated request bean or response bean.

Specify the `javax.xml.bind.annotation.XmlElement` annotation in a wrapper style. The operation is not guaranteed if you specify the annotation in a non-wrapper style.

The following is an example of the mapping where the `javax.xml.bind.annotation.XmlElement` annotation is used.

Figure 16-18: Example of the mapping where javax.xml.bind.annotation.XmlElement annotation is used



(1) name element (elementjavax.xml.bind.annotation.XmlElement)

The `name` element is mapped to the `name` element of the `javax.xml.bind.annotation.XmlElement` annotation that annotates the properties of the generated request bean or response bean.

The following are the points to be noted if the `javax.xml.bind.annotation.XmlElement` annotation co-exists with the `javax.jws.WebParam` annotation or the `javax.jws.WebResult` annotation:

- If the `javax.xml.bind.annotation.XmlElement` annotation co-exists with the `javax.jws.WebParam` annotation, you must specify the same value in the `name` element of both the annotations. If the value is not the same, an error message (KDJW61103-E) is output to the standard error output and log. However, if you specify *##default* in the `name` element or a blank character ("") in the `name` element of the `javax.jws.WebParam` annotation, a comparison is not done to check whether the value is the same and hence no message is output.
- If the `javax.xml.bind.annotation.XmlElement` annotation co-exists with the `javax.jws.WebResult` annotation, you must specify the same value in the `name` element of both the annotations. If the value is not the same, an error message (KDJW61104-E) is output to the standard error output and log. However, if you specify *##default* in the `name` element or a blank character ("") in the `name` element of the `javax.jws.WebResult` annotation, a comparison is not done to check whether the value is the same and hence no message is output.

(2) namespace element (javax.xml.bind.annotation.XmlElement)

The `namespace` element is mapped to the `namespace` element of the `javax.xml.bind.annotation.XmlElement` annotation that annotates the properties of the generated request bean or response bean.

The following are the points to be noted if the `javax.xml.bind.annotation.XmlElement` annotation co-exists with the `javax.jws.WebParam` annotation or the `javax.jws.WebResult` annotation:

- If the `javax.xml.bind.annotation.XmlElement` annotation co-exists with the `javax.jws.WebParam` annotation, you must specify the same value in the `targetNamespace` element of the `javax.jws.WebParam` annotation and `namespace` annotation. If the value is not the same, an error message (KDJW61105-E) is output to the standard error output and log. However, if you specify *##default* in the `namespace` element or a blank character ("") in the `targetNamespace` element of the `javax.jws.WebParam` annotation, a comparison is not done to check whether the value is the same and hence no message is output.
- If the `javax.xml.bind.annotation.XmlElement` annotation co-exists with the `javax.jws.WebResult` annotation, you must specify the same value in the `targetNamespace` element of the `javax.jws.WebResult` annotation and `namespace` annotation. If the value is not the same, an error message (KDJW61106-E) is output to the standard error output and log. However, if you specify *##default* in the `namespace` element or a blank character ("") in the `targetNamespace` element of the `javax.jws.WebResult` annotation, a comparison is not done to check whether the value is the same and hence no message is output.

(3) nillable element (javax.xml.bind.annotation.XmlElement)

The `nillable` element is mapped to the `nillable` element of the `javax.xml.bind.annotation.XmlElement` annotation that annotates the properties of the generated request bean or response bean.

You can specify `true` or `false` in the `nillable` element. If you specify `false`, the `nillable` element is not mapped to the `javax.xml.bind.annotation.XmlElement` annotation of the properties of the generated request bean or response bean.

(4) required element (javax.xml.bind.annotation.XmlElement)

The `required` element is mapped to the `required` element of the `javax.xml.bind.annotation.XmlElement` annotation that annotates the properties of the generated request bean or response bean.

You can specify `true` or `false` in the `required` element. If you specify `false`, the `required` element is not mapped to the `javax.xml.bind.annotation.XmlElement` annotation of the properties of the generated request bean or response bean.

16.2.11 javax.xml.bind.annotation.XmlMimeType annotation

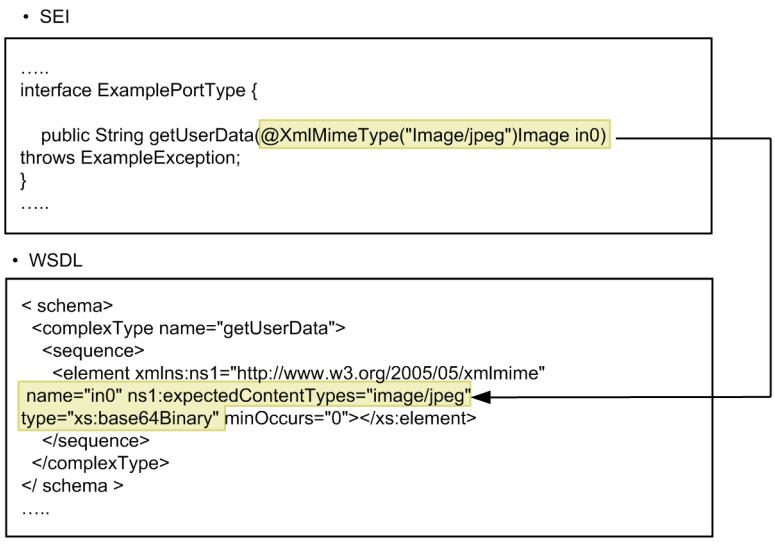
The `javax.xml.bind.annotation.XmlMimeType` annotation is a JAXB annotation that is used for associating Java type and MIME type, and this annotation is used for associating the Java type and MIME type when using the `javax.xml.ws.soap.MTOM` annotation.

You can specify the `javax.xml.bind.annotation.XmlMimeType` annotation in the service method argument, and the return value possessed by the Web Service Implementation Class containing SEI or implicit SEI, or the `getter` method of user definition type. The operation is not guaranteed if specified at any other location (fields such as service method argument, a return value, user definition exceptions possessed by the Web Service Implementation Class without implicit SEI).

When the `javax.xml.bind.annotation.XmlMimeType` annotation is annotated in SEI, the WSDL file issued by the Web Service JAX-WS engine and the WSDL file generated by the `cjwsgen` tool is assigned by the `xmime:expectedContentTypes` attribute that possesses a value specified in the `value` element of the annotation to the schema element that corresponds to the Java type with annotated `javax.xml.bind.annotation.XmlMimeType`.

The following figure shows an example using the `javax.xml.bind.annotation.XmlMimeType` annotation.

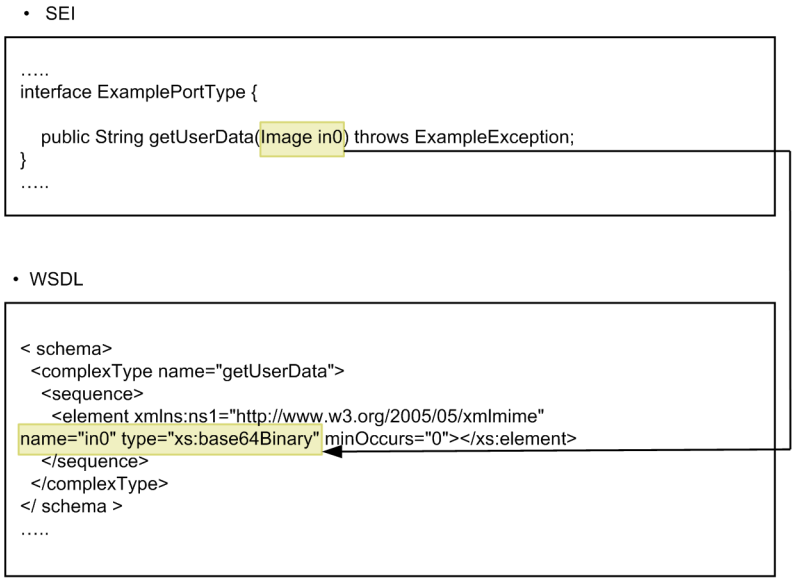
Figure 16-19: Example using the `javax.xml.bind.annotation.XmlMimeType` annotation



When Java type and MIME type are not associated, the `javax.xml.bind.annotation.XmlMimeType` annotation need not be annotated. In this case, the initial value that corresponds to the Java type is used as a value in the `Content-Type` field in the attachment part of the message that is sent in the attachment of the MTOM/XOP specification format.

The following figure shows an example where the `javax.xml.bind.annotation.XmlMimeType` annotation is not used.

Figure 16-20: Example where `javax.xml.bind.annotation.XmlMimeType` annotation is not used



The following table lists the Java type with which the MIME type can be associated and its specification location by annotating the `javax.xml.bind.annotation.XmlMimeType` annotation. Note that the operation is not guaranteed if the `javax.xml.bind.annotation.XmlMimeType` annotation is annotated with any other type.

Table 16-17: Java type where MIME type can be associated and its specification location

No	Java type	Possibility of association	Specification location			
			Method argument	Method return value	User definition type field	User definition exception fields
1	`java.awt.Image`	Y#1	Y	Y	A#2	N#3
2	`javax.xml.transform.Source`	Y	Y	Y	A#2	N#3
3	`javax.activation.DataHandler`	Y	Y	Y	A#2	N#3
4	`java.awt.Image` array type	A#4	Y	Y	A#2	N#3
5	`javax.activation.DataHandler` array type	A#4	Y	Y	A#2	N#3
6	`java.util.List<Image>`	Y	Y	Y	A#2	N#3
7	`java.util.List<DataHandler>`	Y	Y	Y	A#2	N#3
8	`javax.xml.ws.Holder<Image>`	Y	Y	N#5	N#5	N#5
9	`javax.xml.ws.Holder<Source>`	Y	Y	N#5	N#5	N#5

No	Java type	Possibility of association	Specification location			
			Method argument	Method return value	User definition type field	User definition exception fields
10	`javax.xml.ws.Holder<DataHandler>`	Y	Y	N #5	N #5	N #5

Legend:

Y: Can be specified.

A: Can be specified with conditions.

N: Cannot be specified.

#1

This complies with the JAXB specifications. The `java.awt.Image` class is an abstract class that expresses the graphical image in the Java SE specifications, where the data format is not prescribed. When the image data is instantiated by using this relationship, only the decoded information might be retained in a concrete class. Therefore, when sending an image that can be reduced while decoding such as the images in JPEG format, as an attachment, the instance at receiving side might differ from the instance at sending side or from the original data.

If you want to handle the image as is in the original format, use the MIME type (such as `application/octet-stream`) that is mapped to `javax.activation.DataHandler`.

#2

While associating the MIME type to the Java property, annotate the `javax.xml.bind.annotation.XmlMimeType` annotation in the `getter` method. If this is annotated in the field and the `setter` method, the operation is not guaranteed. The following is an example of associating MIME type in the Java property.

```
package com.sample;

import java.awt.Image;

public class UserData {

    private Image image;

    public void setImage(Image image) {
        this.image = image;
    }

    @javax.xml.bind.annotation.XmlMimeType("image/png")
    public Image getImage() {
        return image;
    }
}
```

#3

If you specify the `javax.xml.bind.annotation.XmlMimeType` annotation in a user-defined exception field, the operation is not guaranteed.

#4

You can use only a one-dimensional array. You cannot use multi-dimensional arrays. If you use multi-dimensional arrays, the operation is not guaranteed.

#5

You can specify the `Holder` type only as an argument. You cannot specify the `Holder` type in the return value.

(1) value element (javax.xml.bind.annotation.XmlMimeType)

The `value` element specifies the text expression of the MIME type that is associated with the Java type where the `javax.xml.bind.annotation.XmlMimeType` annotation is annotated.

The MIME type to be specified must be the appropriate MIME type for the Java type where the annotation is annotated. If the MIME type is inappropriate, or multiple MIME types are mentioned by separating them by using a comma, the operation is not guaranteed. Also, in the MIME type to be specified, do not mention the parameters other than the `charset` parameters of the `text/xml` and the `application/xml`. If the parameters other than the

charset parameters of the text/xml and the application/xml are mentioned, the operation is not guaranteed.

The following table describes the MIME types that can be specified in the Java type.

Table 16-18: MIME types that can be specified in the Java type

No	Java type	MIME type that can be specified in the value element
1	java.awt.Image,java.awt.Image array type, java.util.List<Image>, or javax.xml.ws.Holder<Image>	image/png#1
2		image/jpeg#1
3		image/*#2
4	javax.xml.transform.Source or javax.xml.ws.Holder<Source>	text/xml#3
5		application/xml
6	javax.activation.DataHandler, javax.activation.DataHandler array type, java.util.List<DataHandler>, or javax.xml.ws.Holder<DataHandler>	Other than above-mentioned#4

#1

Send the image type that is not mentioned in the table by using the javax.activation.DataHandler class.

#2

When image/* is specified in the MIME type, the Content-Type field value of the MIME header in the attached part of the SOAP message to be sent, is the initial value (image/png) when the java.awt.Image type is used.

#3

When text/xml is specified in the MIME type, the Content-Type field value of the MIME header in the attached part of the SOAP message to be sent, is an initial value (application/xml) when the javax.xml.transform.Source type is used.

#4

When application/* or unknown MIME type is specified in the MIME type, the Content-Type value of the MIME header in the attached part of the SOAP message to be sent, is an initial value (MIME type of the javax.activation.DataHandler object) when the javax.activation.DataHandler type is used.

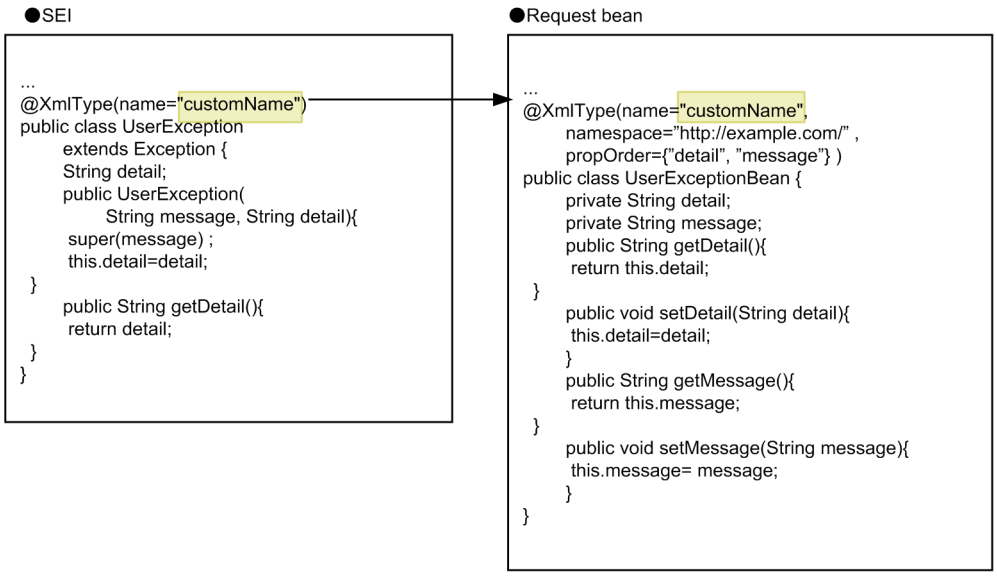
16.2.12 javax.xml.bind.annotation.XmlType annotation

Specify the javax.xml.bind.annotation.XmlType annotation in the exception class. When specified, the annotation is mapped to the javax.xml.bind.annotation.XmlType annotation of the created fault bean.

If you specify the javax.xml.bind.annotation.XmlType annotation in any location other than the exception class, the operation is not guaranteed.

The following is an example where mapping is done using the javax.xml.bind.annotation.XmlType annotation.

Figure 16-21: Example where mapping is done using the `javax.xml.bind.annotation.XmlType` annotation



(1) name element (javax.xml.bind.annotation.XmlType)

The `name` element is mapped to the `name` element of the `javax.xml.bind.annotation.XmlType` annotation of the generated fault bean.

(2) namespace element (javax.xml.bind.annotation.XmlType)

The `namespace` element is mapped to the `namespace` element of the `javax.xml.bind.annotation.XmlType` annotation of the generated fault bean.

(3) propOrder element (javax.xml.bind.annotation.XmlType)

The `propOrder` element is mapped to the `propOrder` element of the `javax.xml.bind.annotation.XmlType` annotation of the fault bean and the property order of the generated fault bean.

In the `propOrder` element, you can specify only a String type array of the property name of the property in an exception class.

Note the following points when using the `propOrder` element:

- If the number of properties is less than 2, the `propOrder` element of the `javax.xml.bind.annotation.XmlType` annotation of the generated fault bean is not mapped.
- If you specify a property name of a non-existing property, an error message (KDJW61107-E) is output to the standard error output and log.
- If you specify a blank character ("") as the property name, the operation is not guaranteed.

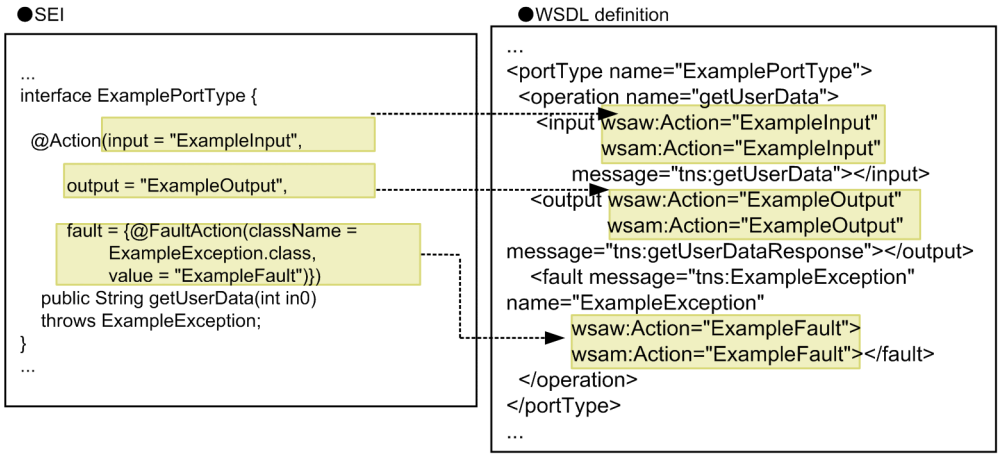
16.2.13 javax.xml.ws.Action annotation

With the `javax.xml.ws.Action` annotation, you can specify a value of the `wsa:Action` element of the addressing header that is used by the Web Service, in each of the `input`, `output`, and `fault` messages of an operation.

You can specify the `javax.xml.ws.Action` annotation only in SEI. If you specify this annotation in a Service Implementation Class, a warning message will output to the standard error output and logs, and the processing will continue (KDJW61095-W).

The following figure shows an example of customization using the `javax.xml.ws.Action` annotation:

Figure 16-22: Example of customization using the javax.xml.ws.Action annotation



(1) fault element (javax.xml.ws.Action)

In the `fault` element, you specify the value of the `wsa:Action` element (`javax.xml.ws.FaultAction` annotation) of the addressing header that is used by the Web Service, when sending a fault message. If you specify `null` or `{null}`, the operation is not guaranteed.

The `fault` element is referenced only when starting the Web Services. This is not interpreted when executing the `apt` command or the `cjwsgen` command.

(2) input element (javax.xml.ws.Action)

In the `input` element, you specify the value of the `wsa:Action` element of the addressing header that is used by the Web Services, when receiving a request message. In the `input` element, specify the characters that satisfy `xsd:anyURI` stipulated in RFC 2396. If you specify any other character, the operation is not guaranteed.

If you specify a blank space in the `input` element, the blank space will become the value of the `wsa:Action` element of the addressing header. If you specify a blank character, the specification is ignored, and it is assumed that you have not coded the `input` element.

The `input` element is referenced only when starting the Web Services. This is not interpreted when executing the `apt` command or the `cjwsgen` command.

(3) output element (javax.xml.ws.Action)

In the `output` element, you specify a value of the `wsa:Action` element of the addressing header that is used by the Web Service when sending a response message. In the `output` element, specify the characters that satisfy `xsd:anyURI` stipulated in RFC 2396. If you specify any other character, the operation is not guaranteed.

If you specify a blank space in the `output` element, the blank space will become the value of the `wsa:Action` element of the addressing header. If you specify a blank character, the specification is ignored, and it is assumed that you have not coded the `output` element.

The `output` element is referenced only when starting the Web Services. This is not interpreted when you execute the `apt` command or the `cjwsgen` command.

16.2.14 javax.xml.ws.BindingType annotation

The `javax.xml.ws.BindingType` annotation is specified in the Web Service Implementation Class. If specified in SEI, this annotation is ignored. When this annotation is specified in SEI, a warning message is output to the standard error output and log and the processing continues (KDJW61079-W).

(1) value element (javax.xml.ws.BindingType)

In the `value` element, you can specify the following field values of the `javax.xml.ws.soap.SOAPBinding` interface.

- "SOAP11HTTP_BINDING" (SOAP1.1 over HTTP)
- "SOAP12HTTP_BINDING" (SOAP1.2 over HTTP)
- "SOAP11HTTP_MTOM_BINDING" (SOAP1.1 over HTTP where attachments of MTOM/XOP specifications format are enabled by default)
- "SOAP12HTTP_MTOM_BINDING" (SOAP1.2 over HTTP where attachments of MTOM/XOP specifications format are enabled by default)

If another field value is specified, an error message is output to the standard error output and logs. The following table describes the error messages output, when you specify an invalid value in the `value` element:

Table 16-19: Error messages output when specifying an invalid value in the value element (javax.xml.ws.BindingType)

No.	Value of the value element	Error message ID	
		When you execute the apt command	When you execute the cjwsgen command
1	`javax.xml.ws.http.HTTPBinding.HTTP_BINDING`	KDJW61072-E	KDJW71005-E
2	An invalid value other than the binding identifier	KDJW61072-E	KDJW71005-E

The following table describes the mapping from the `javax.xml.ws.BindingType` annotation to the `soap:binding` element (child element of the `wsdl:binding` element of WSDL) or the `transport` attribute value of the `soap12:binding` element.

Table 16-20: Mapping the `BindingType` annotation to the `transport` attribute value

No	SOAP version	`BindingType` annotation value	`transport` attribute value

1	SOAP 1.1	`http://schemas.xmlsoap.org/soap/http`
		or
		`@SOAPBinding.SOAP11HTTP_BINDING`
			`http://schemas.xmlsoap.org/soap/http`

2		`http://schemas.xmlsoap.org/wsdl/soap/http?mtom=true`
		or
		`@SOAPBinding.SOAP11HTTP_MTOM_BINDING`	

3	SOAP 1.2	`http://www.w3.org/2003/05/soap/bindings/HTTP/`
		or
		`@SOAPBinding.SOAP12HTTP_BINDING`
			The `transport` attribute value of WSDL differs depending upon the specified value of the `-soap12binding` option or the `com.cosminexus.jaxws.publish_wsdl.soap12binding` property.

4		`http://www.w3.org/2003/05/soap/bindings/HTTP/?mtom=true`
		or
		`@SOAPBinding.SOAP12HTTP_MTOM_BINDING`	

#

For the relationship between the specified value of the `-soap12binding` option, and the `transport` attribute value of a WSDL, see *14.3 cjwsgen command*.

The following table describes the relationship between the specified value of the `com.cosminexus.jaxws.publish_wsdl.soap12binding` property and the `transport` attribute value of WSDL.

Table 16-21: Relationship between specified value of the property and the `transport` attribute value of WSDL

No	Property specification	Specified property value	`transport` attribute value
1	Not specified	None	`http://www.w3.org/2003/05/soap/bindings/HTTP/`
2	Specified	DEFAULT	
3		WSI_BP20_TRANSPORT	`http://schemas.xmlsoap.org/soap/http#`

#

As the transport attribute value is not clear in the standard specifications, you can use this URL in JAX-WS.

16.2.15 javax.xml.ws.FaultAction annotation

With the `javax.xml.ws.FaultAction` annotation, specify a value of the `wsa:Action` element of the addressing header used by the Web Service, when sending a fault message.

You can specify the `javax.xml.ws.FaultAction` annotation only in the `fault` element of the `javax.xml.ws.Action` annotation. This annotation is disabled even when specified in the SEI and a Service Implementation Class.

(1) className element (javax.xml.ws.FaultAction)

The `className` element is a required element of the `javax.xml.ws.FaultAction` annotation. Specify the class name of the exception class sent by the Web Service. If you do not specify the `className` element, an exception occurs in the JAX-WS engine at the Web Service machine, and the Web Services cannot be started (KDJW40013-E).

Note that the operation is not guaranteed if any class other than the exception class that is sent by the Web Services is specified in the `className` element.

The `className` element is referenced only when starting the Web Services. This is not interpreted when executing the `apt` command or the `cjwsgen` command.

(2) value element (javax.xml.ws.FaultAction)

In the `value` element, specify the value of the `wsa:Action` element of the addressing header that is used by the Web Service, when sending the fault message of the exception class specified in the `className` element. In the `value` element, specify the characters that satisfy `xsd:anyURI` stipulated in RFC 2396. If you specify any other character, the operation is not guaranteed.

If you specify a blank space in the `output` element, the blank space will become the value of the `wsa:Action` element of the addressing header. If you specify a blank character, the specification is ignored, and it will be assumed that you have not coded the `value` element.

The `value` element is referenced when starting the Web Services. This is not interpreted when executing the `apt` command or the `cjwsgen` command.

16.2.16 javax.xml.ws.RequestWrapper annotation

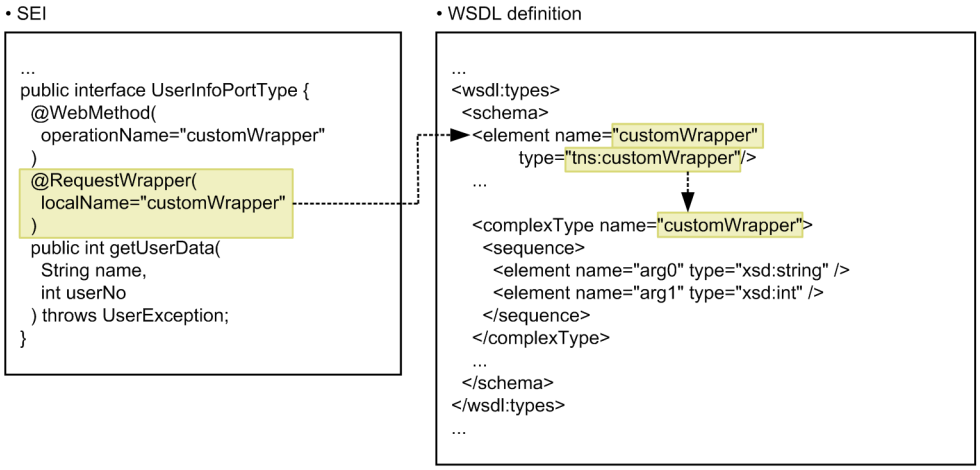
The `javax.xml.ws.RequestWrapper` annotation can be specified in the `wrapper` style. If specified in the `non-wrapper` style, a warning message is output to the standard error output and log and the processing continues (KDJW61061-W).

The `javax.xml.ws.RequestWrapper` annotation is specified in SEI. If specified in the Web Service Implementation Class, this annotation is ignored. When this annotation is specified in the Web Service

Implementation Class, a warning message is output to the standard error output and log and the processing continues (KDJW61077-W).

The following figure shows an example of customization with the `javax.xml.ws.RequestWrapper` annotation:

Figure 16–23: Example of customization with the javax.xml.ws.RequestWrapper annotation



(1) localName element (javax.xml.ws.RequestWrapper)

You specify the `localName` element to customize the mapping of the local names of the request `wrapper` element. If the element value is specified in the `localName` element, you can also customize the type name of the `wrapper` element.

The precautions for specifying the `localName` element are as follows:

- Specify the `localName` element using one-byte alphanumeric characters and underscore (_). If other characters are specified, the operations might not function properly (error message is not output).
- Specify the same name for the `localName` element and the operation name. If the names vary, an error message is output to the standard error output and logs (KDJW61047-E). However, if the `javax.xml.ws.RequestWrapper` annotation does not exist, or if you specify a blank character ("") in the `localName` element of the `javax.xml.ws.RequestWrapper` annotation, a comparison is not done to check whether the names are the same and hence a message is not output.

(2) targetNamespace element (javax.xml.ws.RequestWrapper)

You specify the `targetNamespace` element to customize the mapping of the name spaces of the request `wrapper` element.

In the `targetNamespace` element, the `http://` or `urn:` protocol is specified as a name space. The following name space formats and strings can be specified:

- **Protocol**

  Code the name space protocol using the `http://` or `urn:` protocol. If a protocol other than `http://` or `urn:` (such as `https://`, `file://`) is coded, an error message is output to the standard error output and log (KDJW61042-E).

  Also, if you code the name space using the relative path, an error message is output to the standard error output and logs (KDJW61043-E).

- **Name space coding format**

  The following formats cannot be coded in the name space. If the following formats are used for coding, an error message is output to the standard error output and logs (KDJW61044-E):

  - Query string (example) `http://example.com/?a=b`
  - Anchor (example) `http://example.com/index.html#anchor`
  - Port number (example) `http://example.com:8080/`

- User name/ password (example) `http://user:password@example.com`

- **Strings that can be coded**

  In the segments separated using the separation characters, forward slash (/) or period (.), you can code a string that fulfills all the conditions described in the following table. However, when customizing with the binding declaration, you can code the strings that can be used as the `xsd:NCName` type of the XML Schema specification.

  Table 16-22: Conditions for the strings that can be coded in the name space (javax.xml.ws.RequestWrapper)

No.	Condition	Example of invalid string	Operation when an invalid string is specified

1	Strings using only one-byte alphanumeric characters (0 to 9, A to Z, a to z)	`http://Hitachi.com`
		`http://133.145.224.19/`
		`http://[1080:2C14;D30:BA04:275:806:270C:418A]/`
			The operations might not function properly (error message is not displayed).

2	Strings other than Java reserved words	`http://hitachi.com/abstract`	The operations might not function properly.
3	Strings that do not begin with numeric characters	`http://1hitachi.com`	

(3) className element (javax.xml.ws.RequestWrapper)

The `className` element specifies the class name of the generated request bean using a fully qualified name.

The precautions for specifying the `className` element are as follows:

- Specify the `className` element using one-byte alphanumeric characters, underscore (_), and dollar mark ($). If other characters are specified, the operations might not function properly (error message is not output).
- Specify a value complying with the Java identifier naming rules provided in the Java language specifications. If the value does not comply with the Java identifier naming rules, an error message is output to the standard error output and logs (KDJW61040-E).

(4) partName element (javax.xml.ws.RequestWrapper)

Specify the `partName` element when customizing the mapping of the part name of the input message that references the request `wrapper` element.

Specify the `partName` with one-byte alphanumeric characters and underscore (_). The operation is not guaranteed if any other character is specified (no error message is output).

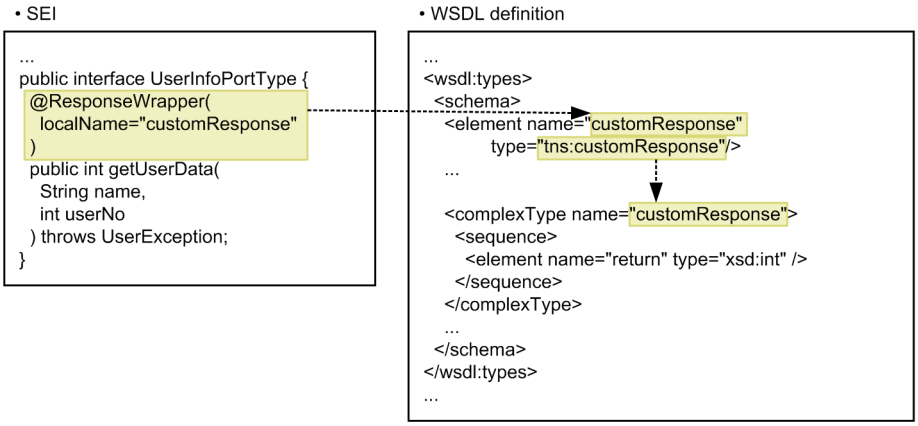
16.2.17 javax.xml.ws.ResponseWrapper annotation

The `javax.xml.ws.ResponseWrapper` annotation can be specified in the `wrapper` style. If specified in the `non-wrapper` style, a warning message is output to the standard error output and log and the processing continues (KDJW61062-W).

The `javax.xml.ws.ResponseWrapper` annotation is specified in SEI. If specified in the Web Service Implementation Class, this annotation is ignored. When this annotation is specified in the Web Service Implementation Class, a warning message is output to the standard error output and log and the processing continues (KDJW61078-W).

The following figure shows an example of customization with the `javax.xml.ws.ResponseWrapper` annotation:

Figure 16-24: Example of customization with the javax.xml.ws.ResponseWrapper annotation



(1) localName element (javax.xml.ws.ResponseWrapper)

You specify the `localName` element to customize the mapping of the local names of the response `wrapper` element. If the element value is specified in the `localName` element, you can also customize the type name of the `wrapper` element.

Specify the `localName` element using one-byte alphanumeric characters and underscore (_). If other characters are specified, the operations might not function properly (error message is not output).

(2) targetNamespace element (javax.xml.ws.ResponseWrapper)

You specify the `targetNamespace` element to customize the mapping of the name spaces of the response `wrapper` element.

In the `targetNamespace` element, the `http://` or `urn:` protocol is specified as a name space. The following name space formats and strings can be specified:

- **Protocol**

  Code the name space protocol using the `http://` or `urn:` protocol. If a protocol other than `http://` or `urn:` (such as `https://`, `file://`) is coded, an error message is output to the standard error output and logs (KDJW61049-E).

  Also, if you code a name space using the relative path, an error message is output to the standard error output and logs (KDJW61050-E).

- **Name space coding format**

  The following formats cannot be coded in the name space. If the following formats are used for coding, an error message is output to the standard error output and log (KDJW61051-E):

  - Query string (example) `http://example.com/?a=b`
  - Anchor (example) `http://example.com/index.html#anchor`
  - Port number (example) `http://example.com:8080/`
  - User name/ password (example) `http://user:password@example.com`

- **Strings that can be coded**

  In the segments separated using the separation characters, forward slash (/) or period (.), you can code a string that fulfills all the conditions described in the following table:

Table 16-23: Conditions for the strings that can be coded in the name space (javax.xml.ws.ResponseWrapper)

No.	Condition	Example of invalid string	Operation when an invalid string is specified

1	Strings using only one-byte alphanumeric characters (0 to 9, A to Z, a to z)	`http://Hitachi.com`
		`http://133.145.224.19/`
		`http://[1080:2C14;D30:BA04:275:806:270C:418A]/`
			The operations might not function properly (error message is not displayed).

2	Strings other than Java reserved words	`http://hitachi.com/abstract`	The operations might not function properly.
3	Strings that do not begin with numeric characters	`http://1hitachi.com`	

(3) className element (javax.xml.ws.ResponseWrapper)

The `className` element specifies the class name of the generated response bean using a fully qualified name.

The precautions for specifying the `className` element are as follows:

- Specify the `className` element using a period (.) that is the delimiter of the package, one-byte alphanumeric characters, underscores (_), and dollar mark ($). If other characters are specified, the operations might not function properly (error message is not output).
- Specify a value complying with the Java identifier naming rules provided in the Java language specifications. If the value does not comply with the Java identifier naming rules, an error message is output to the standard error output and logs (KDJW61041-E).

(4) partName element (javax.xml.ws.ResponseWrapper)

Specify the `partName` element when customizing the part name of the output message that references the request `wrapper` element.

Specify the `partName` with one-byte alphanumeric characters and underscore (_). If you specify `partName` with any other character, the operation is not guaranteed. (no error message is output).

16.2.18 javax.xml.ws.ServiceMode annotation

In the `javax.xml.ws.ServiceMode` annotation, specify whether the target to be accessed by the provider is only the payload (SOAP Body) of the protocol message or is the entire protocol message (SOAP Envelop).

The `javax.xml.ws.ServiceMode` annotation is referenced only when starting the Web Services. This is not interpreted when executing the `apt` command or the `cjwsgen` command.

(1) value element (javax.xml.ws.ServiceMode)

With the `value` element, specify either `javax.xml.ws.Service.Mode.MESSAGE` or `javax.xml.ws.Service.Mode.PAYLOAD`. The default value is `javax.xml.ws.Service.Mode.PAYLOAD`.

If you specify `javax.xml.ws.Service.Mode.MESSAGE`, the entire protocol message is passed to the provider instance, and if you specify `javax.xml.ws.Service.Mode.PAYLOAD`, only the payload of the protocol message is passed to the provider instance.

16.2.19 javax.xml.ws.soap.Addressing annotation

You must have the `javax.xml.ws.soap.Addressing` annotation for using the addressing functionality.

The description of the `javax.xml.ws.soap.Addressing` annotation is as follows:

Web Services

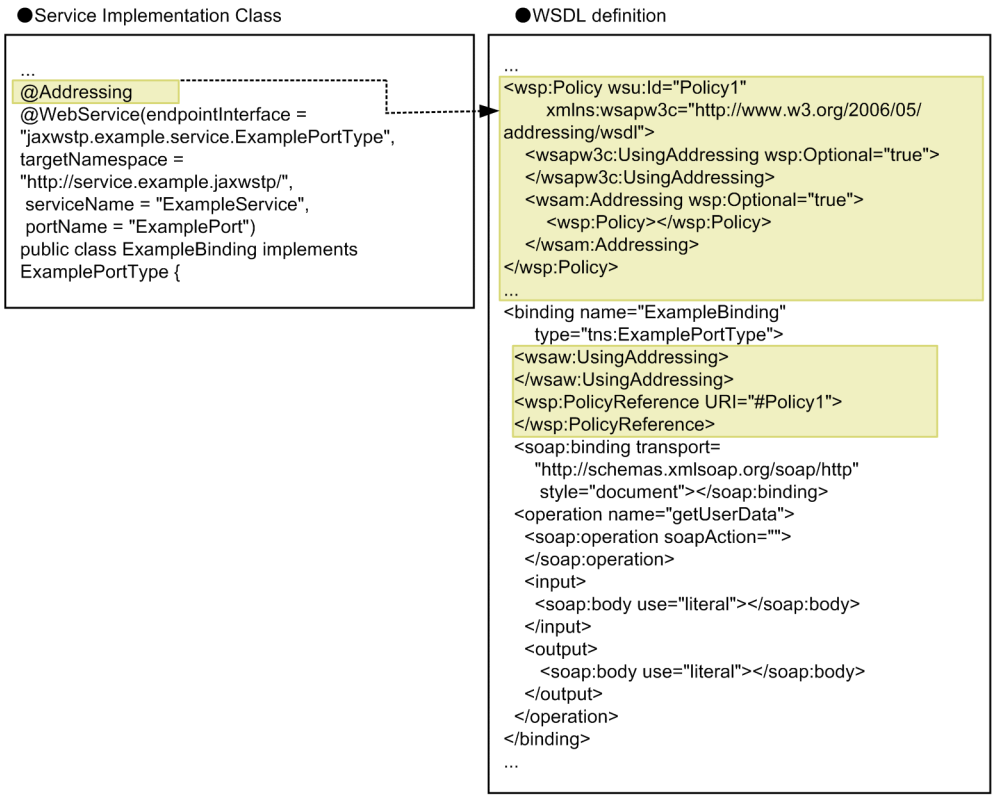
You can specify the `javax.xml.ws.soap.Addressing` annotation only in a Service Implementation Class. If you specify the `javax.xml.ws.soap.Addressing` annotation in SEI, a warning message will output to the standard error output and logs, and the processing will continue (KDJW61094-W).

Web Services client

You can specify this annotation in the `setter` method or a field that injects the port. For details, see *10.21.1(4) Enabling features*. The annotation will be ignored if you specify the annotation in any other fields or methods.

The following figure shows an example of the customization using the `javax.xml.ws.soap.Addressing` annotation:

Figure 16-25: Example of customization using the javax.xml.ws.soap.Addressing annotation (Web Service)



(1) enabled element (javax.xml.ws.soap.Addressing)

In the `enabled` element, specify whether to enable the addressing functionality in the Web Service. If you specify `true`, the addressing functionality is enabled, and if you specify `false`, the addressing functionality is disabled. The default value is `true`.

The `enabled` element is referenced only when starting the Web Services. This is not interpreted when executing the `apt` command or the `cjwsgen` command.

(2) required element (javax.xml.ws.soap.Addressing)

In the `required` element, specify whether the addressing header is required in the request message, when you invoke Web Services. If you specify `true`, the addressing header will be required, and if you specify `false`, the addressing header becomes optional. The default value is `false`.

The `required` element is referenced only when starting the Web Services. This is not interpreted when executing the `apt` command or the `cjwsgen` command.

(3) responses element (javax.xml.ws.soap.Addressing)

In the `responses` element, specify the type of the response end point requested by the end point when WS-Addressing is enabled. The following table describes the values that can be specified in the `responses` element.

Table 16-24: Values that can be specified in the `responses` element

No.	Values of responses element	Explanation
1	`javax.xml.ws.soap.AddressingFeature.Responses.ALL`	The default value. You can specify all the URIs.
2	`javax.xml.ws.soap.AddressingFeature.Responses.ANONYMOUS`	You can specify only an anonymous URI.
3	`javax.xml.ws.soap.AddressingFeature.Responses.NON_ANONYMOUS`	You can specify only a non-anonymous URI.

If you specify `javax.xml.ws.soap.AddressingFeatures.Responses.ANONYMOUS` in the element value of the `responses` element, you must specify an anonymous URI in the `wsa:ReplyTo` element and `wsa:FaultTo` element of the addressing header of the message to be sent to the end point. If you specify a non-anonymous URI, `javax.xml.ws.WebServiceException` is returned.

If you specify `javax.xml.ws.soap.AddressingFeatures.Responses.NON_ANONYMOUS` in the element value of the `responses` element, you must specify a non-anonymous URI. If an anonymous URI is specified, `javax.xml.ws.WebServiceException` is returned.

By default, you can specify all the same URIs as that in the case when `javax.xml.ws.soap.AddressingFeatures.Responses.ALL` is specified.

The `responses` element is referenced only when starting the Web Services and is not interpreted when executing the `apt` or `cjwsgen` command.

16.2.20 javax.xml.ws.soap.MTOM annotation

Specify the `javax.xml.ws.soap.MTOM` annotation in the Web Service that uses the attachments of MTOM/XOP specification format.

The description of the `javax.xml.ws.soap.MTOM` annotation is as follows.

Web Services

You can specify the `javax.xml.ws.soap.MTOM` annotation only in the Web Service Implementation Class. The annotation is ignored if specified in SEI. Also, the operation is not guaranteed if the annotation is specified in the Provider Implementation Class (class that implements the `javax.xml.ws.provider` interface).

Web Services client

You can specify this annotation in the `setter` method or a field that injects the port. For details, see *10.21.1(4) Enabling features*. The annotation will be ignored if you specify the annotation in any other fields or methods.

When creating the Web Service that uses attachments of MTOM/XOP specification format with the help of the skeleton class of the Web Service Implementation Class generated by the cjwsimport command, the `javax.xml.ws.soap.MTOM` annotation is not mapped in the skeleton class of the Web Service Implementation Class. Therefore, you must specify the `javax.xml.ws.soap.MTOM` annotation. Note that even if the `javax.xml.ws.soap.MTOM` annotation is specified in the Web Service Implementation Class, an element or attribute indicating the usage of attachments of MTOM/XOP specification format does not appear in the WSDL file issued by the Web Service JAX-WS engine or WSDL file generated by the cjwsgen command.

The `javax.xml.ws.soap.MTOM` annotation is referenced only when starting the Web Services. Therefore, this annotation is not interpreted when executing the apt command and the cjwsgen command.

The following is an example using the `javax.xml.ws.soap.MTOM` annotation in Web Service:

```
.......
@MTOM
@WebService(endpointInterface = "jaxwstp.example.service.ExamplePortType",
```

```
targetNamespace = "http://service.example.jaxwstp/", serviceName = "ExampleService",
portName = "ExamplePort")
public class ExampleBinding implements ExamplePortType {
```

(1) enabled element (javax.xml.ws.soap.MTOM)

The `enabled` element specifies whether to use the attachments of MTOM/XOP specifications format in the Web Service. When `true` is specified, attachments of MTOM/XOP specifications format can be used, and when `false` is specified, attachments of MTOM/XOP specification format cannot be used. The default value is `true`.

(2) threshold element (javax.xml.ws.soap.MTOM)

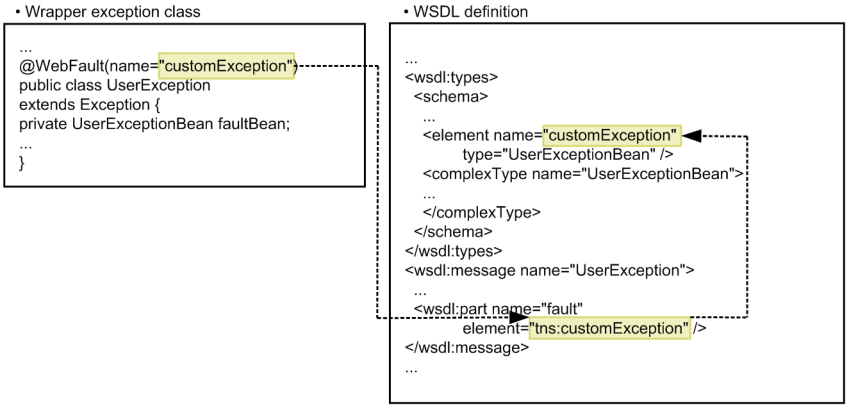
The `threshold` element is the threshold for sending the binary data as attachments of MTOM / XOP specifications format when you can use attachments of MTOM / XOP specifications format in the Web Service. In classes other than the `javax.activation.DataHandler` class, for the binary data exceeding the specified value (threshold element value ≦ size of the data to be sent), binary data is sent as attachments of MTOM/XOP specifications format. The default value is `0`.

16.2.21 javax.xml.ws.WebFault annotation

You can use the `javax.xml.ws.WebFault` annotation to customize the mapping of the return values.

The following figure shows an example of customization with the `javax.xml.ws.WebFault` annotation:

Figure 16-26: Example of customization with the javax.xml.ws.WebFault annotation



(1) name element (javax.xml.ws.WebFault)

You use the `name` element to customize the local name of the global element mapped from the fault bean.

Specify the `name` element using one-byte alphanumeric characters and underscore (_). If other characters are specified, the operations might not function properly (error message is not output).

(2) targetNamespace element (javax.xml.ws.WebFault)

You specify the `targetNamespace` element to customize the name space of the global element mapped from the fault bean.

In the `targetNamespace` element, the `http://` or `urn:` protocol is specified as a name space. The following name space formats and strings can be specified:

- **Protocol**

  Code the name space protocol using the `http://` or `urn:` protocol. If a protocol other than `http://` or `urn:` (such as `https://`, `file://`) is coded, an error message is output to the standard error output and logs (KDJW61067-E).

Also, if you code the name space using the relative path, an error message is output to the standard error output and logs (KDJW61068-E).

- **Name space coding format**

  The following formats cannot be coded in the name space. If the following formats are used for coding, an error message is output to the standard error output and log (KDJW61069-E):

  - Query string (example) `http://example.com/?a=b`
  - Anchor (example) `http://example.com/index.html#anchor`
  - Port number (example) `http://example.com:8080/`
  - User name/ password (example) `http://user:password@example.com`

- **Strings that can be coded**

  In the segments separated using the separation characters, forward slash (/) or period (.), you can code a string that fulfills all the conditions described in the following table. However, when you perform the customization using the binding declaration, you can code character strings that can be used as the `xsd:NCName` type of XML Schema specifications.

  Table 16-25: Conditions for the strings that can be coded in the name space (javax.xml.ws.WebFault)

No.	Conditions	Examples of invalid strings	Operation when an invalid string is specified

1	Strings using only one-byte alphanumeric characters (0 to 9, A to Z, a to z)	`http://Hitachi.com`
		`http://133.145.224.19/`
		`http://[1080:2C14;D30:BA04:275:806:270C:418A]/`
			The operations might not function properly (error message is not displayed).

2	Strings other than Java reserved words	`http://hitachi.com/abstract`	The operations might not function properly.
3	Strings that do not begin with numeric characters	`http://1hitachi.com`	

(3) faultBean element (javax.xml.ws.WebFault)

The `faultBean` element specifies the class name of the generated fault bean using a fully qualified name. If the wrapper exception class has the `getFaultInfo` method that returns the `javax.xml.ws.WebFault` annotation and the fault bean, the fault bean is not generated even if the `faultBean` element is specified.

The precautions for specifying the `faultBean` element are as follows:

- Specify the `faultBean` element using one-byte alphanumeric characters and underscore (_). If other characters are specified, the operations might not function properly (error message is not output).
- Specify a value complying with the Java identifier naming rules provided in the Java language specifications. If the value does not comply with the Java identifier naming rules, an error message is output to the standard error output and logs (KDJW61039-E).

(4) messageName element (javax.xml.ws.WebFault)

Specify the `messageName` element when customizing the fault message name (the `name` attribute of the `wsdl:message` element referenced from the `wsdl:fault` element) corresponding to the wrapper exception class.

Specify the `messageName` element with one- byte alphanumeric characters and underscore (_). If you specify the `messageName` element with any other character, the operation is not guaranteed (no error message is output).

16.2.22 javax.xml.ws.WebServiceProvider annotation

Specify the `javax.xml.ws.WebServiceProvider` annotation in the class that implements the `javax.xml.ws.provider` interface, and this annotation declares that the class that satisfies the requirements of the provider defines the endpoint of the Web Service.

You can specify only one annotation from the `javax.xml.ws.WebServiceProvider` and `javax.jws.WebService` annotations. If you specify the `javax.xml.ws.WebServiceProvider` and `javax.jws.WebService` annotations together, an error message will output to the standard error output and logs (KDJW61098-E).

The `javax.xml.ws.WebServiceProvider` annotation is referenced only when starting the Web Services. This is not interpreted when executing the `apt` command or the `cjwsgen` command.

(1) targetNamespace element (javax.xml.ws.WebServiceProvider)

In the `targetNamespace` element, specify the `http://` protocol or the `urn:` protocol as a name space. You can specify the following formats and character strings for the name space:

- **Protocol**

  Code the name space protocol using the `http://` protocol or the `urn:` protocol. If you code a protocol other than `http://` or `urn:` (such as `https://` and `file://`), an error message will output to the standard error output and logs (KDJW61099-E).

  Also, if you code the name space using the relative path, an error message will output to the standard error output and logs (KDJW61100-E).

- **Name space coding format**

  You cannot code the following formats in the name space. If you use the following formats for coding, an error message will output to the standard error output and logs (KDJW61101-E):

  - Query string (example) `http://example.com/?a=b`
  - Anchor (example) `http://example.com/index.html#anchor`
  - Port number (example) `http://example.com:8080/`
  - User name/ password (example) `http://user:password@example.com`

- **Character strings that you can code**

  In a segment demarcated with delimiters such as a forward slash (/) or period (.), you can code a character string that satisfies all the conditions described in the following table:

  Table 16-26: Conditions for character strings that you can code in a name space (javax.xml.ws.WebServiceProvider)

No.	Condition	Examples of invalid character strings	Operation when an invalid character string is specified

1	Character strings using only single-byte alphanumeric characters (0 to 9, A to Z, and a to z)	`http://Hitachi.com/`
		`http://133.145.224.19/`
		`http://[1080:2C14;D30:BA04:275:806:270C:418A]/`
			The operation is not guaranteed (no error message is displayed).

2	Character strings containing anything than reserved terminology of Java	`http://hitachi.com/abstract`	The operation is not guaranteed.
3	Character strings that do not begin with a numeric character	`http://1hitachi.com`	

(2) serviceName element (javax.xml.ws.WebServiceProvider)

Specify the `serviceName` element with single-byte alphanumeric characters and underscores (_). If you specify any other character, the operation is not guaranteed (no error message is displayed).

(3) portName element (javax.xml.ws.WebServiceProvider)

Specify the `portName` element with single-byte alphanumeric characters and underscores (_). If you specify any other character, the operation is not guaranteed (no error message is displayed).

(4) wsdlLocation element (javax.xml.ws.WebServiceProvider)

For details about the `wsdlLocation` element, see the section *10.6 Issuing the Meta data*.

17 Web Resources and Providers

This chapter describes the support range of the resource classes and providers of RESTful Web Services (Web resources).

17.1 Resource classes

This section describes the difference between the resource methods, sub-resource methods, and the sub-resource locators of a resource class. This chapter also describes the difference between the root resource and sub-resource classes.

The resource methods, sub-resource methods, and the sub-resource locators of a resource class are defined depending on the availability of `Path` annotations and request method identifiers. The following table describes the respective definitions.

Table 17-1: Definition of resource and sub-resource methods, and sub-resource locator

No.	Method or locator	Path annotation	Request method designator
1	Resource method	N	Y
2	Sub-resource method	Y	Y
3	Sub-resource locator	Y	N

Legends:

Y: Indicates availability

N: Indicates non-availability

Use a JAX-RS engine for generating the instances of root resource classes. When generating an instance, an injection is performed to constructor parameters, fields, and bean properties according to the JAX-RS specifications.

On the other hand, the JAX-RS engine is not used for generating an instance of a sub-resource class. A sub-resource class must be instantiated by a corresponding sub-resource locator. Therefore, you are required to use a sub-resource locator or a sub-resource class to initialize constructor parameters, fields, and bean properties.

17.1.1 Root resource classes

A *root resource class* is a `public` class of Java that is set up as an annotation using the `Path` annotation at a class level, and contains at least one of the resource methods, sub-resource methods, or sub-resource locators.

The following example describes a root resource class containing a resource method and a sub-resource locator.

```
package com.sample.resources;

import javax.ws.rs.GET;
import javax.ws.rs.Path;

//root resource class
@Path("/root")
public class Resource2 {

  //subresource locator corresponding to the request to context root+ "/root/sub1"
  @Path("/sub1")
  public SubResource1 subResourceLocator1() {
    //returns an instance of the subresource class to be processed
    return new SubResource1();
  }

  //subresource locator corresponding to the request to context root+ "/root/sub2"
  @Path("/sub2")
  public SubResource2 subResourceLocator2() {
    //returns an instance of the subresource class to be processed

    return new SubResource2();
}
  //resource method
  @GET
  public String getValue() {
    String returnValue = "";
    //sets a return value
    return returnValue;
  }

}
```

`com.sample.resources.Resource2` is a root resource class. Note that the annotation is performed with the `Path` annotation at a class level. This root resource class contains two sub-resource locators; `subResourceLocator1()` and `subResourceLocator2()`, and the resource method `getValue()` for processing `HTTP GET` requests. Note that the `Path` annotation is used for the annotation of a sub-resource locator and the request method designator is used for the annotation of a resource method. Both `SubResource1` and `SubResource2` are sub-resource classes. For details, see the following subsections:

- *17.1.1(4) Resource method*
- *17.1.1(5) Sub-resource method*
- *17.1.1(6) Sub-resource locator*
- *17.1.7 Sub-resource classes*

```
package com.sample.resources;

import javax.ws.rs.GET;
import javax.ws.rs.Path;

//root resource class
@Path("/root")
public class Resource3 {

  //sub-resource method corresponding to the request to context root+ "root/sub1"
  @Path("/sub1")
  @GET
  public String subResourceMethod1() {
    String value = "";
    // Execute the process and then return the result.
    return value;
  }

  //sub-resource method corresponding to the request to context root+ "root/sub2"
  @Path("/sub2")
  @GET
  public String subResourceMethod2() {
    String value = "";
    // Execute the process and then return the result.
    return value;
  }
}
```

`com.sample.resources.Resource3` is a root resource class. This root resource class contains two sub-resource methods; `subResourceMethod1()` and `subResourceMethod2()`. Both the `Path` annotation and the request method are used for the annotation of a sub-resource method.

If an asterisk (*) is specified in the URL of the `Path` annotation, you can invoke only a resource method. If you invoke a sub-resource method or a sub-resource locator, the system throws `java.lang.StringIndexOutOfBoundsException` exception, which can be handled by the exception mapping provider.

(1) Life cycle

The JAX-RS engine instantiates a root resource class for each request corresponding to a Web resource. The life cycle of a root resource class is as follows:

1. A constructor is invoked
2. The required injection is performed
3. An appropriate method is invoked
4. Considered as a target for a Garbage Collection (GC)

(2) Constructor

A root resource class must contain at least one public constructor including the default public constructor (constructors that are not explicitly declared).

The following example describes a public constructor containing parameters.

```
package com.sample.resources;

import javax.ws.rs.DefaultValue;
import javax.ws.rs.Encoded;
import javax.ws.rs.QueryParam;
import javax.ws.rs.Path;
import javax.ws.rs.GET;

//root resource class
@Path("/root")
public class Resource1 {
  private String query1;

  //public constructor containing parameters
  public Resource1(@Encoded @DefaultValue("abc") @QueryParam("query") String query){
    this.query1 = query;
  }

  //resource method
  @GET
  public String getValue() {
    return "Your requested query parameter \"query\" is: " + this.query1;
  }
}
```

In this example, the root resource class `com.sample.resources.Resource1` is instantiated depending on the public constructor `Resource1()` containing the parameter `query` that is annotated using the `QueryParam` annotation.

The `Encoded` annotation allows you to disable the automatic URL decoding of the parameter `query`. Furthermore, the `DefaultValue` annotation is also included and used to specify the default value if the value to be injected into the parameter `query` does not exist in the request sent by a client.

```
package com.sample.resources;

import javax.ws.rs.GET;
import javax.ws.rs.Path;

//root resource class
@Path("/root")
public class Resource2 {
  //public constructor without parameters
  public Resource2() {
    // Execute the process
  }

 //resource method
  @GET
  public String getValue(){
    return "Your request was accepted.";
  }
}
```

In this example, the public constructor `Resource2()` that does not contain parameters instantiates the root resource class `com.sample.resources.Resource2`.

The following example describes multiple public constructors with parameters.

```
package com.sample.resources;

import javax.ws.rs.Encoded;
import javax.ws.rs.MatrixParam;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.QueryParam;

//root resource class
@Path("/root/")
public class Resource3 {
  private String matrix1;
  private String query1;

  //public constructor without parameters
  public Resource3() {
    // Execute the process
  }
```

```
  //public constructor with parameters(1st)
  @Encoded
  public Resource3(@MatrixParam("matrix1") String matrix1) {
    this.matrix1 = matrix1;
  }

  //public constructor with parameters(2nd)
  @Encoded
  public Resource3(@FormParam("form1") String form1,
      @QueryParam("query1") String query1) {
    this.form1 = form1;
    this.query1 = query1;
  }

  //resource method
  @GET
  public String getValue() {
    return "Your requested matrix parameter \"matrix1\" is: " + this.matrix1 +  "\n" +
      "Your requested query parameter \"query1\" is: " + this.query1;
  }
}
```

In this example, the second public constructor that contains the parameters `matrix1` and `query1`, annotated respectively by the `MatrixParam` and `QueryParam` annotations, instantiates the root resource class `com.sample.resources.Resource3`.

The `Encoded` annotation used at the constructor level allows you to disable the automatic URL decoding of the parameters `matrix1` and `query1`.

The following example describes the default constructor.

```
package com.sample.resources;

import javax.ws.rs.FormParam;
import javax.ws.rs.POST;
import javax.ws.rs.Path;

//root resource class
@Path("/root")
public class Resource4 {

  //resource method
  @POST
  public String getValue(@FormParam("form1") String form1) {
    return "Your requested form parameter \"form1\" is: " + form1;
  }
}
```

In this example, the implicitly declared default constructor `Resource4()` instantiates the root resource class `com.sample.resources.Resource4`.

If a default constructor throws the `java.oi.IOException` exception, an error (KDJJ10039-E) occurs, and the `route resource` class is not instantiated. `500` is returned as the HTTP status code.

If no public constructor is declared, an error (KDJJ10006-E) occurs, and the root resource class is not instantiated. The system returns `500` as the HTTP status code.

Note that the following constructors are not `public` constructors:

- `private` constructors
- `protected` constructors
- Constructors without access identifiers

The following table describes the combinations of optional and injection annotations that you can use as constructor parameters.

Table 17-2: Annotations that can be used as constructor parameters

No.	Annotation for injection	Option annotation	
		Encoded	DefaultValue
1	`MatrixParam`	Y	Y
2	`QueryParam`	Y	Y
3	`PathParam`	Y	N
4	`CookieParam`	N	Y
5	`HeaderParam`	N	Y
6	`FormParam`	N	Y
7	`Context`	N	N

Legends:

Y: Indicates the annotations that you can use in combination with injection annotations.

N: Indicates the annotations that you cannot use with injection annotations.

The optional `Encoded` annotation allows you to disable the automatic URL decoding of the parameters of the constructors to be injected.

The optional `DefaultValue` annotation allows you to specify an assumed default value, if a value is not available for injecting into the parameters of a constructor targeted for injection.

If a root resource class contains more than one public constructor with parameters, the JAX-RS engine instantiates a root resource class using the constructor with the maximum parameters. If a root resource class contains two or more constructors with the maximum parameters, the JAX-RS engine instantiates a root resource class using the constructor defined first. At this time, the warning message (KDJJ20010-W) is output to a log.

For all the conditions described below, the HTTP request will not be processed. The system returns `500` as the HTTP status code. Note that when confirming logs, you must confirm the J2EE server log file instead of the JAX-RS functionality log file.

- From among the parameters of the `root resource` class constructor, if a parameter that uses the `Injection` and `DefaultValue` annotations throws a runtime exception while the JAX-RS engine executes the injection operation
- If a default constructor of a root resource class throws a checked exception (excluding runtime exceptions) other than the `java.lang.InstantiationException`, `java.lang.IllegalAccessException`, `java.lang.reflect.InvocationTargetException`, or the `java.io.IOException` exception

(3) Fields and Bean properties

The following table describes the combinations of injection and option annotations that you can use in the fields and bean properties of a root resource class. When instantiating a root resource class, the JAX-RS engine injects values in the annotated fields and bean properties based on annotations. The bean properties are not necessarily read-only.

Table 17-3: Annotations that can be used in fields and bean properties

No.	Annotation for injection	Option annotation	
		Encoded	DefaultValue
1	`MatrixParam`	Y	Y
2	`QueryParam`	Y	Y
3	`PathParam`	Y	N
4	`CookieParam`	N	Y
5	`HeaderParam`	N	Y

No.	Annotation for injection	Option annotation	
		Encoded	DefaultValue
6	FormParam	N	Y
7	Context	N	N

Legends:

Y: Indicates the annotations that you can use in combination with injection annotations.

N: Indicates the annotations that you cannot use with injection annotations.

The optional Encoded annotation allows you to disable the automatic URL decoding of the fields or bean properties to be injected.

The optional DefaultValue annotation allows you to specify an assumed default value, if a value is not available for injecting into the bean properties or the fields targeted for injection.

The following example describes the usage of the DefaultValue annotation in the root resource class fields.

```
private @DefaultValue("value1") @QueryParam("id") String id;
```

In this example, if the query parameter id is not specified in the URL, the field id is "value1".

The following example describes the usage of the DefaultValue annotation in the bean properties of a root resource class.

```
private String property1;

@DefaultValue("10") @QueryParam("prop1")
public void setProperty1(String property1) {
  this.property1 = property1;
}
```

In this example, if the query parameter prop1 is not specified in the URI, the value of the bean property property1 is "10".

The following example describes the usage of the Encoded annotation in the fields of a root resource class:

```
 private @Encoded @QueryParam("id") String id;
```

In this example, the field id is not automatically URL decoded.

The following example describes the usage of the Encoded annotation in the bean property of a root resource class.

```
private String property1;

@Encoded @QueryParam("prop1")
public void setProperty1(String property1) {
  this.property1 = property1;
}
```

In this example, the value of the bean property property1 is not automatically URL decoded.

From among the bean properties or the fields of the root resource class, if a parameter that uses the Injection and DefaultValue annotation throws a runtime exception while the JAX-RS engine executes the injection operation, the HTTP request is not processed. The system returns 500 as the HTTP status code. Note that when confirming logs, you must confirm the J2EE server log file instead of the JAX-RS functionality log file.

(4) Resource method

The resource method is a method of a root resource class, annotated by one of the request method designators defined in the JAX-RS specifications. A root resource class can contain one or more resource methods.

The request method designators defined in JAX-RS specifications are as follows:

- GET annotation

- `POST` annotation
- `PUT` annotation
- `DELETE` annotation
- `HEAD` annotation
- `OPTIONS` annotation

An error (KDJJ10006-E) occurs when you use two or more request method designators for one resource method, and consequently the root resource class is not instantiated. `500` is returned as the HTTP status code.

An error (KDJJ10006-E) occurs when you use the same request method designator for two or more resource methods, and consequently the root resource class is not instantiated. The system returns. `500` as the HTTP status code.

If there is no resource method to dispatch a HTTP request, system specifies `405` as the HTTP status code, and the exception mapping provider throws the exception `javax.ws.rs.WebApplicationException`.

The following example describes how a request method designator is used with a resource method.

```
@GET
@Encoded
@Produces("text/plain")
public String echo(@QueryParam("id") String id){
  return "ID is: " + id;
}
```

In this example, the `echo()` method is annotated by the `GET` request method designator. Furthermore, the `echo()` method is annotated by the `Produces` annotation having "`text/plain`" in the value to return the HTTP response where the content type is "`text/plain`". Note that the parameter id is annotated by the `QueryParam` annotation that receives the query parameter id, and is additionally annotated by the `Encoded` annotation to disable the automatic URL decoding of query parameters.

(a) Visibility

The resource method must be a `public` method to which a request method designator is applied. Although the request method designators annotate the following methods, these methods are not the resource methods:

- `Private` methods
- `Protected` methods
- Methods without access identifiers

When you apply the request method designator to any of the above-mentioned methods, the warning message or the error message (KDJJ20003-W or KDJJ10006-E) is output to the log.

For details on KDJJ20003-W and KDJJ10006-E, see *13.7.1 Checking the syntax when initializing Web resources (KDJJ20003-W and KDJJ10006-E)*.

(b) Parameter annotations

The following table describes the combinations of the injection and optional annotations that can be used in the resource method parameters.

Table 17−4: Annotations that can be used in the resource method parameters

No.	Annotation for injection	Option annotation	
		Encoded	DefaultValue
1	`MatrixParam`	Y	Y
2	`QueryParam`	Y	Y
3	`PathParam`	Y	N
4	`CookieParam`	N	Y
5	`HeaderParam`	N	Y

No.	Annotation for injection	Option annotation	
		Encoded	DefaultValue
6	FormParam	N	Y
7	Context	N	N

Legends:

Y: Indicates the annotations that you can use in combination with injection annotations.

N: Indicates the annotations that you cannot use with injection annotations.

The optional Encoded annotation allows you to disable the automatic URL decoding of the parameters of the resource method to be injected.

The optional DefaultValue annotation allows you to specify the default value of the annotations, which are annotated in the parameters of the resource method to be injected.

The following example describes how to use an annotation in the parameter of a resource method.

```
@GET
@Produces("text/plain")
public String echo(@Encoded @DefaultValue("10") @QueryParam("id") String id, @Encoded
@MatrixParam("matrix1") String matrix1){
  return "ID is: " + id + "\nMatrix1 is: " + matrix1;
}
```

In this example, the resource method echo() contains the parameter id annotated by the QueryParam annotation and the parameter matrix1 annotated by the MatrixParam annotation. The parameter id is additionally annotated by the Encoded and DefaultValue annotations to disable the automatic URL decoding and to specify the default value respectively. The matrix parameter is additionally annotated by the Encoded annotation to disable the automatic URL decoding.

From among the parameters of a resource method or sub-resource method, if a parameter that uses DefaultValue and the injection annotation throws a runtime exception while the JAX-RS engine executes the injection operation, an error (KDJJ10009-E or KDJJ10006-E) occurs, and consequently the HTTP request is not processed. The system returns 500 as the HTTP status code.

(c) Entity parameters

For details on entity parameters, see *17.1.2 Entity parameter*.

(d) Return values

For details on return values, see *17.1.3 Return values*.

(5) Sub-resource method

The resource methods annotated by the Path annotation are specifically referred to as *sub-resource* methods. The only difference between a sub-resource method and resource method is whether the Path annotation is used.

An example of a sub-resource method is as follows.

```
package com.sample.resources;

import javax.ws.rs.POST;
import javax.ws.rs.Path;

//root resource class
@Path("/root/")
public class Resource1 {

  //sub-resource method
  @Path("sub1")
  @POST
  public String doSomething(String entityBody) {
    return "By Sub-Resource Method.";
  }
```

```
    //resource method
    @POST
    public String doOthers(String entityBody){
      return "By Resource Method.";
    }
  }
```

In this example, the `doSomething()` method is a sub-resource method. Consider the context root of the Web application (WAR file) containing the root resource class `com.sample.resources.Resource1` to be "*example*" and that the Web application is considered to be published on a host named "*sample.com*". In this case, the HTTP POST request corresponding to the URL "`http://sample.com/example/root/sub1`" is dispatched to the sub-resource method `doSomething()`.

On the other hand, the HTTP POST request corresponding to the URL "`http://sample.com/example/root`" is dispatched to the resource method `doOthers()`.

(6) Sub-resource locators

A method of the root resource class that is annotated only by the `Path` annotation and where the request method designators are not applied is called a *sub-resource locator*.

The sub-resource locators return sub-resource classes that execute the remaining processing of the HTTP request. For details on sub-resource classes, see *17.1.7 Sub-resource classes*.

An example of a sub-resource locator of a root resource class is as follows.

```
package com.sample.resources;

import javax.ws.rs.Encoded;
import javax.ws.rs.QueryParam;
import javax.ws.rs.PathParam;
import javax.ws.rs.GET;
import javax.ws.rs.Path;

//root resource class
@Path("/root/")
public class Resource {

  //subresource locator corresponding to the request to context root+ "/root/sub1"
  @Path("sub1")
  public SubResource getRequestHandler(@PathParam("id") String id,
      @Encoded @QueryParam("query1") String query1) {
    return new SubResource(id, query1);
  }
}
```

An example of the corresponding sub-resource class is as follows.

```
//subresource class
public class SubResource {
  private String id;
  private String query;

  public SubResource(String id, String query){
    this.id = id;
    this.query = query;
  }
  @GET
  public String getRequestParameter(){
    return "ID is: " + this.id + "\nQuery is: " + this.query;
  }
}
```

In this example, the root resource class `com.sample.resources.Resource` does not process the HTTP request directly. `com.sample.resources.SubResource` processes the sub-resource class returned by the sub-resource locator `getRequestHandler`.

You consider the context root of the Web application (WAR file) containing the resource class `com.sample.resources.Resource` to be "*example*" and that the Web application is assumed to be published on a host named "*sample.com*". In this case, the HTTP GET request corresponding to the URL "`http://`

sample.com/example/root/sub1?query1=10" is dispatched to the method getRequestHandler() of the root resource class com.sample.resources.Resource.

The resource method getRequestParameter() of the sub-resource class com.sample.resources.SubResource executes the remaining processing of the HTTP GET request.

If you use an entity parameter as the parameter of a sub-resource locator, the (KDJJ10006-E) error occurs, and consequently the HTTP request sent by the client is not processed. The system returns 500 as the HTTP status code.

The operation when a sub-resource locator is not public, the sub-resource class performs the same operations as the operations of resource method.

If the type of the return value of a sub-resource locator is void, the (KDJJ10006-E) error occurs, and consequently the HTTP request sent by the client is not processed. The system returns 500 as the HTTP status code.

From among the parameters of sub-resource locators, if a parameter that uses the Injection and DefaultValue annotation throws a runtime exception when the JAX-RS engine executes the injection operation, the HTTP request is not processed. The system returns 500 as the HTTP status code. Note that when confirming the log, you must confirm the J2EE server log file instead of the JAX-RS functionality log file.

17.1.2 Entity parameters

From among the parameters of a resource method, a parameter that is not annotated by an annotation is called an *entity parameter*. The value of the entity parameter is an *entity body*.

The following example describes an entity parameter used in the resource method.

```
@POST
public String getRequestParameters(@MatrixParam("matrix") String matrix,
    String entity) {
  return "Matrix is:" + matrix + "\mEntity Body is: " + entity;
}
```

In this example, the resource method getRequestParameters() contains the parameter matrix annotated by the MatrixParam annotation and the un-annotated parameter entity (entity parameter). When an HTTP request with the entity body content of Entity Content is received, the value of the entity parameter entity is Entity Content.

(1) Combination of Java types and MIME media types available for entity parameters

The following table describes the combinations of the Java types and the MIME media types that can be used as entity parameters. Do not use the annotations of the JAXB specifications for a POJO. If you use the annotations, the actual operations might differ from the described operations.

Table 17-5: Combinations of Java types and MIME media types that can be used as entity parameters

No.	Java type	Charset[#1]	MIME media type
1	byte[]	N	Any(*/*)
2	java.lang.String	Y	Any(*/*)
3	java.io.InputStream	N	Any (*/*)
4	java.io.Reader	Y	Any (*/*)
5	java.io.File[#2]	Y	Any (*/*)
6	javax.activation.DataSource	Y	Any (*/*)
7	javax.xml.transform.Source[#3]	N	text/xml, application/xml, application/*+xml
8	javax.xml.bind.JAXBElement<String>[#4]	N	text/xml, application/xml,

No.	Java type	Charset[#1]	MIME media type
8	javax.xml.bind.JAXBElement<String>[#4]	N	application/*+xml
9	The JAXB class annotated by the XmlRootElement annotation and/or the XmlType annotation[#4]	N	text/xml, application/xml, application/*+xml
10	javax.ws.rs.core.MultivaluedMap<String,String>	Y	application/x-www-form-urlencoded
11	org.w3c.dom.Document	N	text/xml, application/xml, application/*+xml
12	java.util.List<T>[#5]	N	text/xml, application/xml, application/*+xml
13	java.awt.image.RenderedImage	N	application/octet-stream, image/jpeg
14	com.cosminexus.jersey.core.provider.EntityHolder<T>[#6]	D	The same MIME media type as the type specified in T.
15	POJO[#7]	Y	application/json

Legend:

Any (*/*): Indicates that all MIME media types are supported

#1

Indicates whether the content of the charset parameter included in the `HTTP Content-Type` header is considered when the parameter is injected into an entity parameter.

Y: Considered. UTF-8 is considered as the charset if the charset parameter is not included in the `HTTP Content-Type` header.

D: Depends on the type specified in T

N: Not considered

Note that the charset parameter is ignored, if included in the value of the Consumes annotation.

#2

The JAX-RS engine creates a temp directory in the J2EE server environment to save a temporary file. Use the cjsetup command to configure the J2EE server environment. For details on the cjsetup command, see *cjsetup (Set up and unset up of a J2EE server)* in the *Cosminexus Application Server Command Reference Guide*.

#3

You can use the following implementation classes:

- javax.xml.transform.stream.StreamSource
- javax.xml.transform.sax.SAXSource
- javax.xml.transform.dom.DOMSource

#4

When the MIME media type is `application/fastinfoset` or `application/json` , the operation completes successfully without occurrence of an error.

#5

With T, you can specify the JAXB class annotated by the XmlRootElement annotation and XmlType annotation.

#6

You can specify the types from No. 2 through No.13, and No. 15 of this table in T.

#7

Enable the JSON POJO mapping. The operation when the JSON POJO mapping is disabled is the same as the operation when an unsupported Java type is specified in an entity parameter. For details on how to enable the JSON POJO mapping, see *18. Mapping JSON and POJO*.

(2) Notes on entity parameters

Further notes on the `entity` parameters are as follows:

Operation if an exception is thrown when converting to an entity parameter

If an exception is thrown when converting to an `entity` parameter, an error occurs. For details on how to handle the exceptions, see *17.1.8 Exception handling*.

Operation when a Java type does not support an entity parameter type or when the entity body of an HTTP request cannot use the MIME media type

The (KDJJ10024-E) error occurs when the following Java type entity parameters contain a MIME media type that cannot be used by the entity body of an HTTP request, and the system, throws javax.ws.rs.WebApplicationException, which can be handled by the exception mapping provider and has 415 as the HTTP status code.

1. javax.xml.bind.JAXBElement<String>
2. The `JAXB` class annotated by the XmlRootElement annotation and/or the XmlType annotation
3. javax.ws.rs.core.MultivaluedMap<String,String>
4. java.util.List<T>
5. POJO

The operation, however, ends successfully without any errors when the MIME media type of the entity body of an HTTP request is `application/fastinfoset` or `application/json` in the step No.1 or No.2 mentioned above.

In the entity parameter of java.awt.image.RenderedImage, if the MIME media type of the entity body of an HTTP request is image/*, the system throws java.io.IOException, which can be handled by the exception mapping provider.

- If the HTTP request contains an entity body in the entity parameter of com.cosminexus.jersey.core.provider.EntityHolder<T>, in either of the following cases, the error (KDJJ10003-E) occurs and the system throws javax.ws.rs.WebApplicationException, which can be handled by the exception mapping provider and has 500 as the HTTP status code.

  - If the type does not support T (No. 1 and No. 14 of the table)

  - If the type supports T (From No. 2 to No. 13 of the table) but the MIME media type does not support the entity body of the HTTP request

- If you use a Java type that does not support the entity parameter type (excluding the cases where the MIME media types of the entity body of the HTTP request is application/xml, text/xml, or application/*+xml and java.lang.Object is being used), the (KDJJ10024-E) error occurs and the system throws an exception javax.ws.rs.WebApplicationException, which can be handled by the exception mapping provider and has one of the following values as the HTTP status code.

  400: When the entity parameter type is java.lang.Object and the MIME media type of the entity body of the HTTP request is application/xml, text/xml or application/*+xml

  415: When the `entity` parameter type is other than java.lang.Object, or when the `entity` parameter type is java.lang.Object and the MIME media type of the entity body of an HTTP request is other than `application/xml`, `text/xml`, or `application/*+xml`

  The process, however, completes successfully without occurrence of an error when the entity parameter type is javax.mail.internet.MimeMultipart and the MIME media type of the entity body of the HTTP request is multipart/*.

The number of entity parameters available for a resource method

You can use only one entity parameter in a resource method. If a resource method contains multiple entity parameters, the warning message (KDJJ20012-W) is output to the log. The entity body of the HTTP request is injected only into the initial entity parameter and the state of the second parameter and that of those thereafter is not guaranteed.

Resource methods that contain a GET request method identifier

When a resource method with a GET request method designator contains an entity parameter, a warning message or an error message (KDJJ20003-W or KDJJ10006-E) is output to the log. For details on KDJJ20003-W and KDJJ10006-E, see *13.7.1 Checking the syntax when initializing a Web resource (KDJJ20003-W and KDJJ10006-E)*.

Important note

The Encoded annotation or the DefaultValue annotation is ignored, if annotated in the entity parameter.

Messages that are output when the type parameters of the entity parameters cannot be resolved

When the type parameter of an entity parameter cannot be resolved, a warning message or an error message (KDJJ10006-E or KDJJ20003-W) is output to the log. For details on KDJJ20003-W and KDJJ10006-E, see *13.7.1 Checking the syntax when initializing a Web resource (KDJJ20003-W and KDJJ10006-E)*.

Notes when a specific type is specified in an entity parameter

When the `entity` parameter type is `javax.ws.rs.core.MultivaluedMap`<*String,String*> or `com.cosminexus.jersey.core.provider.EntityHolder`<*javax.ws.rs.core.MultivaluedMap*<*String,String*>>, note the following points:

- If an entity body is accessed by a component other than a servlet or a servlet filter of the JAX-RS functionality, a warning message (KDJJ20007) is output to the log.

  Here, the `entity` parameters are in an undetermined state when being accessed. Reference the form parameters included in the entity body from the parameters annotated by the `FormParam` annotation.

- An entity body can contain a maximum of `10,000 form` parameters by default.

  If the number of parameters of a request exceed the specified number, an error (KDJJ10042-E) occurs and the system throws the `javax.ws.rs.WebApplicationException` exception, for which `413` is set in the HTTP status code and which can be processed by the exception mapping provider. Change the `webserver.connector.limit.max_parameter_count` property of the user property file (`usrconf.properties`) for the J2EE server, as and when required.

  For details on the user property file for the J2EE server, see *2.4 usrconf.properties (User property file for the J2EE server* in the *uCosminexus Application Server Definition Reference Guide*.

When the entity parameter type is a POJO, note the following points:

- If the JSON POJO mapping is disabled, an error (KDJJ10024-E) occurs and the system throws the `javax.ws.rs.WebApplicationException` exception, for which `415` is set as the status code and which can be processed by the exception mapping provider. For details on how to enable the JSON POJO mapping, see *18. Mapping JSON and POJO*.

17.1.3 Return values

The following table describes the combinations of the Java types and MIME media types, which can be used in the return value of a resource method. The return value is converted to the entity body of an HTTP response. Do not use the annotations in the JAXB specifications for POJO. If you use the annotation, the actual operations might differ from the described operations.

Table 17–6: Combinations of Java types and MIME media types, which can be used in the return value of a resource method

No	Java type	Charset#1	MIME media type
1	byte[]	N	Any (*/*)
2	java.lang.String	Y	Any (*/*)
3	java.io.InputStream	N	Any (*/*)
4	java.io.Reader	Y	Any (*/*)
5	java.io.File	N	Any (*/*)
6	javax.activation.DataSource	N	Any (*/*)
7	javax.xml.transform.Source#2	N	text/xml, application/xml, application/*+xml
8	javax.xml.bind.JAXBElement<String>#3	Y	text/xml,

No	Java type	Charset#1	MIME media type
8	javax.xml.bind.JAXBElement<String>[#3]	Y	application/xml, application/*+xml
9	The JAXB class annotated by the XmlRootElement annotation[#3]	Y	text/xml, application/xml, application/*+xml
10	javax.ws.rs.core.MultivaluedMap<String, String>	Y	application/x-www-form-urlencoded
11	javax.ws.rs.core.StreamingOutput	N	Any (*/*)
12	org.w3c.dom.Document	N	text/xml, application/xml, application/*+xml
13	java.util.List<T>[#4]	Y	text/xml, application/xml, application/*+xml
14	java.awt.image.RenderedImage	N	image/jpeg
15	Void	--	Any(*/*)
16	javax.ws.rs.core.Response	Y	Any (*/*)
17	javax.ws.rs.core.GenericEntity<T>[#5]	D	Same MIME media type as specified in T.
18	POJO[#6]	N[#7]	application/json

Legend:

--: Indicates a non-applicable item

Any (*/*): Indicates that all MIME media types are supported

#1

When a charset parameter is to be included in the Produces annotation or return value, indicate whether that information will be reflected in the charset parameter of the Content-type HTTP header for converting to an HTTP response.

Y: Reflected. UTF-8 is considered as the charset if the charset parameter is not included in the Produces annotation and the return value

D: Depends on the Java type specified in T

N: Not reflected

#2

The following implementation classes can be used:

- javax.xml.transform.stream.StreamSource

- javax.xml.transform.sax.SAXSource

- javax.xml.transform.dom.DOMSource

#3

When the MIME media type is `application/fastinfoset` or `application/json`, the operation completes successfully without occurrence of an error.

#4

You can specify the JAXB class annotated by the XmlRootElement annotation in T.

#5

You can specify all the types except No.17 of the table in T.

#6

Enable the JSON POJO mapping. The operations when the JSON POJO mapping is disabled are the same as the operations when you specify an unsupported Java type in an entity parameter. For details on how to enable the JSON POJO mapping, see *18. Mapping JSON and POJO*.

#7

Do not add the charset parameter in the `HTTP Content-Type` header.

The following table describes the mapping of the return value and the HTTP response.

Table 17-7: Mapping of return value and HTTP response

No.	Return value		HTTP response	
	Type	Value	HTTP status code	Entity body
1	void	--	204	Void entity body
2	Response	A non-null instance	200	Response entity properties
3		Null	204	Void entity body
4	String	A non-null instance	200	String value
5		Null	204	Void entity body
6	Supported Java types except void, Response, String	A non-null instance	200	A converted entity body based on the return value class
7		Null	204	Void entity body

Legend:

--: Indicates that the return value is unavailable

For the Java types where the return value types are not supported, the error (KDJJ10026-E) occurs, and consequently the system throws the exception javax.ws.rs.WebApplicationException, which can be handled by the exception mapping provider and has 500 as the HTTP status code. The process, however, successfully completes without occurrence of an error when the return value type is `javax.mail.internet.MimeMultipart` and the MIME media type is `multipart/*`.

If an exception is thrown when converting to entity body of an HTTP response, an error occurs. For details on how to handle the exceptions, see *17.1.8 Exception handling*.

If the type of a return value is in the following Java type and the MIME media type that cannot be used by the entity body of the HTTP response, an error (KDJJ10026-E) occurs, and consequently the system throws javax.ws.rs.WebApplicationException, which can be handled by the exception mapping provider and has 500 as the HTTP status code.

1. `javax.xml.bind.JAXBElement<String>`
2. The JAXB class annotated by the `XmlRootElement` annotation
3. `javax.ws.rs.core.MultivaluedMap<String, String>`
4. `java.util.List<T>`
5. `java.awt`.image.RenderedImage

The operation, however, successfully completes without occurrence of an error when the MIME media type of the entity body of the HTTP request is `application/fastinfoset` or `application/json` in No.1 or No.2.

For the following return values, a warning message or an error message (KDJJ20003-W or KDJJ10006-E) is output to the log. For details on KDJJ20003-W and KDJJ10006-E, see *13.7.1 Checking the syntax when initializing a Web resource (KDJJ20003-W and KDJJ10006-E)*.

- When the type of the return value of a resource method containing the GET request method designator is void
- When the type parameter of the return value cannot be resolved

17.1.4 Parameter types

The following table lists and describes the parameter types for which you can specify the `Injection` annotation. The table also describes whether you can use the types in combination with the `DefaultValue` annotation.

Table 17-8: Type of parameters supporting each annotation

No.	Data type		annotation						
			Path Param	Query Param	Matrix Param	Cookie Param	HeaderParam	FormParam	Context
1	Primitive	int	C	Y#1	Y#1	Y#2	Y#1	Y#1	N
2		short	C	Y#1	Y#1	Y#2	Y#1	Y#1	N
3		long	C	Y#1	Y#1	Y#2	Y#1	Y#1	N
4		float	C	Y#1	Y#1	Y#2	Y#1	Y#1	N
5		double	C	Y#1	Y#1	Y#2	Y#1	Y#1	N
6		char	C	Y#1	Y#1	Y#2	Y#1	Y#1	N
7		byte	C	Y#1	Y#1	Y#2	Y#1	Y#1	N
8		boolean	C	Y#1	Y#1	Y#2	Y#1	Y#1	N
9	Wrapper class	Integer	C	Y#1	Y#1	Y#2	Y#1	Y#1	N
10		Short	C	Y#1	Y#1	Y#2	Y#1	Y#1	N
11		Long	C	Y#1	Y#1	Y#2	Y#1	Y#1	N
12		Float	C	Y#1	Y#1	Y#2	Y#1	Y#1	N
13		Double	C	Y#1	Y#1	Y#2	Y#1	Y#1	N
14		Character	N	N	N	N	N	N	N
15		Byte	C	Y#1	Y#1	Y#2	Y#1	Y#1	N
16		Boolean	C	Y#1	Y#1	Y#2	Y#1	Y#1	N
17	Type that contains a constructor with one String type argument		C#7	Y#1,#7	Y#1,#7	Y#2,#7	Y#1,#7	Y#1,#7	N
18	Type that contains one String type argument and a static `valueOf` method that returns an instance of that type		C#7	Y#1,#7	Y#1,#7	Y#2,#7	Y#1,#7	Y#1,#7	N
19	Type that contains one String type argument and a static `fromString` method that returns an instance of that type		C#7	Y#1,#7	Y#1,#7	Y#2,#7	Y#1,#7	Y#1,#7	N
20	`enum` type that contains one String type argument and a static `fromString` method that returns an instance of that type		C#3,#7	Y#1,#3,#7	Y#1,#3,#7	Y#2,#3,#7	Y#1,#3,#7	Y#1,#3,#7	N
21	Type other than the enum type that contains one String type argument and both the static `valueOf` and `fromString` methods that returns an instance of that type		C#4,#7	Y#1,#4,#7	Y#1,#4,#7	Y#2,#4,#7	Y#1,#4,#7	Y#1,#4,#7	N
22	Other than the aforementioned		N	N	N	N	N	N	N

No.	Data type		annotation						
			Path Param	Query Param	Matrix Param	Cookie Param	HeaderParam	FormParam	Context
23	List<T>	When T is `Integer`	C	Y	Y	N	Y	Y	N
24		When T is `Short`	C	Y	Y	N	Y	Y	N
25		WhenC T is `Long`	C	Y	Y	N	Y	Y	N
26		When T is `Float`	C	Y	Y	N	Y	Y	N
27		When T is `Double`	C	Y	Y	N	Y	Y	N
28		When T is `Character`	N	N	N	N	N	N	N
29		When T is `Byte`	C	Y	Y	N	Y	Y	N
30		When T is `Boolean`	C	Y	Y	N	Y	Y	N
31		When T is a type that contains a constructor having one String type argument	C[#7]	Y[#7]	Y[#7]	N	Y[#7]	Y[#7]	N
32		When T is a type that contains one String type argument and a static `valueOf` method that returns an instance of that type	C[#7]	Y[#7]	Y[#7]	N	Y[#7]	Y[#7]	N
33		When T contains one String type argument and a static `fromstring` method that returns an instance of that type	C[#7]	Y[#7]	Y[#7]	N	Y[#7]	Y[#7]	N
34		When T is an enum type that contains one String type argument and a static `fromString` method that returns an instance of that type	C[#3,#7]	Y[#3,#7]	Y[#3,#7]	N	Y[#3,#7]	Y[#3,#7]	N
35	List<T>	When T is a type other than the enum type that contains one String type argument and both the static `valueOf` and `fromString` methods that return an instance of that type	C[#4,#7]	Y[#4,#7]	Y[#4,#7]	N	Y[#4,#7]	Y[#4,#7]	N
36		Other than the aforementioned	N	N	N	N	N	N	N
37	Set<T>[#5]	When T is `Integer`	C	Y	Y	N	Y	Y	N
38		When T is `Short`	C	Y	Y	N	Y	Y	N
39		When T is `Long`	C	Y	Y	N	Y	Y	N
40		When T is `Float`	C	Y	Y	N	Y	Y	N
41		When T is `Double`	C	Y	Y	N	Y	Y	N
42		When T is `Character`	N	N	N	N	N	N	N

No.	Data type		annotation						
			Path Param	Query Param	Matrix Param	Cookie Param	HeaderParam	FormParam	Context
43	Set<T>[#5]	When T is `Byte`	C	Y	Y	N	Y	Y	N
44		When T is `Boolean`	C	Y	Y	N	Y	Y	N
45		When T is a type that contains a constructor having one String type argument	C[#7]	Y[#7]	Y[#7]	N	Y[#7]	Y[#7]	N
46		When T contains one String type argument and a static `valueOf` method that returns an instance of that type	C[#7]	Y[#7]	Y[#7]	N	Y[#7]	Y[#7]	N
47		When T contains one String type argument and a static `fromString` method that returns an instance of that String type	C[#7]	Y[#7]	Y[#7]	N	Y[#7]	Y[#7]	N
48	Set<T>[#3]	When T is an enum type that contains one String type argument and both the static `valueOf` and `fromString` methods that return an instance of that type	C[#3,#7]	Y[#3,#7]	Y[#3,#7]	N	Y[#3,#7]	Y[#3,#7]	N
49		When T is a type other than the enum type that contains one String type argument and both the `valueOf` and `formString` methods that return an instance of that type	C[#4,#7]	Y[#4,#7]	Y[#4,#7]	N	Y[#4,#7]	Y[#4,#7]	N
50		When T is other than the beforementioned types	N	N	N	N	N	N	N
51	Sorted Set<T>[#5, #6]	When T is `Integer`	C	Y	Y	N	Y	Y	N
52		When T is `Short`	C	Y	Y	N	Y	Y	N
53		When T is `Long`	C	Y	Y	N	Y	Y	N
54		When T is `Float`	C	Y	Y	N	Y	Y	N
55		When T is `Double`	C	Y	Y	N	Y	Y	N
56		When T is `Character`	N	N	N	N	N	N	N
57		When T is `Byte`	C	Y	Y	N	Y	Y	N
58		When T is `Boolean`	C	Y	Y	N	Y	Y	N
59		When T is a type that contains a constructor having one String type argument	C[#7]	Y[#7]	Y[#7]	N	Y[#7]	Y[#7]	N
60		When T contains one String type argument and a static `valueOf` method that	C[#7]	Y[#7]	Y[#7]	N	Y[#7]	Y[#7]	N

No.	Data type		annotation						
			Path Param	Query Param	Matrix Param	Cookie Param	HeaderParam	FormParam	Context
60	Sorted Set<T>[#5, #6]	returns an instance of that type	C[#7]	Y[#7]	Y[#7]	N	Y[#7]	Y[#7]	N
61	Sorted Set<T>[#3, #4]	When T contains one String type argument and a static `fromString` method that returns an instance of that type	C[#7]	Y[#7]	Y[#7]	N	Y[#7]	Y[#7]	N
62		When T is `enum` type that contains one String type argument and both the static `valueOf` and `formString` methods returning instance of that type	C[#3,#7]	Y[#3,#7]	Y[#3,#7]	N	Y[#3,#7]	Y[#3,#7]	N
63		When T is a type other than the enum type that contains one string argument and both the static `valueOf` and `fromString` methods that return an instance of that type	C[#4,#7]	Y[#4,#7]	Y[#4,#7]	N	Y[#4,#7]	Y[#4,#7]	N
64		When T is other than the before-mentioned types	N	N	N	N	N	N	N
65	PathSegment		C	N	N	N	N	N	N
66	Context type	UriInfo	N	N	N	N	N	N	C
67		HttpHeaders	N	N	N	N	N	N	C
68		Request	N	N	N	N	N	N	C
69		SecurityContext	N	N	N	N	N	N	C
70		Providers	N	N	N	N	N	N	C
71		ServletConfig	N	N	N	N	N	N	C
72		ServletContext	N	N	N	N	N	N	C
73		HttpServletRequest	N	N	N	N	N	N	C
74		HttpServletResponse	N	N	N	N	N	N	C

Legends:

Y: Indicates that you can use the injection annotation

N: Indicates that you cannot use the injection annotation

C : Indicates that you can use the injection annotation, but not in combination with the `DefaultValue` annotation.

#1

On receiving multiple parameters with the same name, the JAX-RS engine uses the value of only the first parameter.

#2

On receiving multiple parameters with the same name, the JAX-RS engine injects the value of only the last parameter.

#3

When a type contains one String type argument and both the static `valueOf` and static `fromString` methods returning instance of that type, the JAX-RS engine uses the value `fromString` method.

#4

When a type contains one String type argument and both the static `valueOf` and static `fromString` methods returning instance of that type, the JAX-RS engine uses the static `valueOf` method.

#5

For a type other than the enum type, the `equals()` method and `hashCode()` method of `java.lang.Object` must be appropriately implemented in the class or in the respective parent class to be specified in `T` in accordance with the rules of the Java language.

#6

For the type other than `enum`, the `java.lang.Comparable` interface must be implemented in the class or the respective parent class to be specified in `T`.

#7

In some cases, the `valueOf` method, `fromString` method, and constructors are invoked multiple times when initializing for validating the injection.

The error (KDJJ10006-E) occurs when you use any of the annotations for the types described in No. 22, No. 36, No. 50, No. 64, and No. 65 of the aforementioned table. The system returns t HTTP response with the HTTP status code `500` in the root resource class and throws `java.lang.RuntimeException`, which can be handled by the exception mapping provider.

However, if you use any of the annotations for the types described in No. 14, No. 28, No. 42, and No. 56 of the table, the system throws `java.lang.RuntimeException`, which can be handled by the exception mapping provider.

17.1.5 Exception mapping

If system throws an exception while executing injections into the parameters of the resource method of a root resource class, the parameters of constructor, fields, and bean properties as well as the parameters of the resource method of a sub-resource class, the JAX-RS engine handles those exceptions as described in the following table. For details on the supported Java types and annotations, see *17.1.4 Parameter types*.

Table 17-9: Support provided by the JAX-RS engine for the exceptions thrown at injection

No.	Annotation	Exceptions thrown at injection.	
		WebApplicationException	Other
1	`MatrixParam`	Handles WebApplicationException as explained in *17.1.8 Exception handling*	Wraps the thrown exception with a non-entity `WebApplicationException` having the HTTP status code 404. Furthermore, the handling is done as explained in *17.1.8 Exception handling*
2	`QueryParam`		
3	`PathParam`		
4	`CookieParam`		Wraps the thrown exception with a non-entity `WebApplicationException` having the HTTP status code 400 . Furthermore, the handling is done as explained in *17.1. 8 Exception handling*
5	`FormParam`		
6	`HeaderParam`		

! Important note

System throws `WebApplicationException` only for No. 17 through 21, No. 31 through 36, No. 45 through 50, and No. 59 through 64 from among the parameter types listed in the table describing the types of parameters supporting each annotation in *17.1.4 Parameter types*.

17.1.6 URI template

Use the `Path` annotation to specify for which URL the root resource class, the sub-resource method, or the sub-resource locator will execute the HTTP request. The value of the `Path` annotation is called *URI template*.

When the URI template is to be specified in a root resource at the class level, describe a relative URI for the context root of the Web application (WAR file) containing the Web resources. However, for sub-resource methods or sub-resource locators, describe a relative URI corresponding to the URI template of a root resource class.

The value of annotation is automatically encoded. For example, the following annotations have the same meaning:

- `@Path ("widget list/{id}")`
- `@Path ("widget%20list/{id}")`

If the `Path` annotation of two or more root resource classes contain the same URI template or contain a URI template that is resolved by the same regular expression, the error (KDJJ10006-E) occurs, and consequently the root resource class is not instantiated. The system returns `500` as the HTTP status code.

If the path annotations of two or more sub-resource methods contain the same URI template or contain a URI template that is resolved by the same regular expression, and if other information such as the media type declaration and the request method designator match, the (KDJJ10006-E) error occurs. In the root resource class, the system returns an HTTP response with the HTTP status code `500`. In the sub-resource class, the system throws `java.lang.RuntimeException`, which can be handled by the exception mapping provider.

When the path annotation of two or more sub-resource locators contain the same URI template or contain a URI template that is resolved by the same regular expression, the (KDJJ10006-E) error occurs. The system returns an HTTP response with the HTTP status code of `500` in the root resource class, and throws `java.lang.RuntimeException`, which can be handled by the exception mapping provider, in the sub-resource class.

If the `Path` annotation is annotated in an interface or in an abstract class, the (KDJJ10006-E) error occurs and the request sent by the client is not processed. The system returns `500` as the HTTP status code.

(1) Template parameters

A URI template can contain zero or more embedded parameters called *template parameters*. Start coding the template parameters with an opening curly bracket ({), and continue coding one or more alphanumeric characters and symbols other than a forward slash (/) and code a closing bracket (}) at the end. You can acquire the actual values of the template parameters by injecting to the parameters, fields, or bean properties annotated by the `PathParam` annotation. For details on how to code the template parameters, check the standard specifications.

An example of a template parameter is as follows.

```
package com.someshop;

import javax.ws.rs.PathParam;
import javax.ws.rs.GET;
import javax.ws.rs.Path;

//root resource class
@Path("/customers")
public class CustomerResource {
  //subresource method
  @GET
  @Path("{id}")
  public String getCustomer(@PathParam("id") int id) {
    //execute to return the assigned value
  }
}
```

In this example, the expression {id} included in the `Path` annotation is the template parameter. Consider the context root of the Web application containing the root resource class `com.someshop.CustomerResource` to be "*resource*" and that the Web application is published on a host named "*someshop.com*". In this case, the HTTP GET request corresponding to the URL "`http://someshop.com/resource/customers/333`" is dispatched to the sub-resource method `getCustomer()` and the actual value of the template parameter id is injected to the parameter id annotated by the `PathParam` annotation.

However, the HTTP GET request corresponding to the URL "`http://someshop.com/resource/customers/333/444`" is not dispatched by any method. 404 is returned as the HTTP status code.

The template parameters can be embedded anywhere in the value of the `Path` annotation (URI template). The following example describes the usage of multiple template parameters.

```
package com.someshop;

import javax.ws.rs.PathParam;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
```

```
//root resource class
@Path("/")
public class CustomerResource {
  //subresource class
  @GET
  @Path("customers/{firstname}-{lastname}")
  public String getCustomer(@PathParam("firstname") String firstname,
      @PathParam("lastname") String lastname) {
    //execute to return the assigned value
  }
}
```

In this example, the expressions {*firstname*} and {*lastname*} to be included in the `Path` annotation are the two template parameters separated by a hyphen.

The HTTP GET request corresponding to the URL "`http://someshop.com/resource/customers/John-Smith`" is dispatched to the sub-resource method `getCustomer()` and the actual values of the template parameters *firstname* and *lastname* are respectively injected to the parameters *firstname* and *lastname* annotated by the `PathParam` annotation.

(a) Regular expressions

The regular expressions other than wild cards can be used in the `Path` annotation. The following example describes the usage of regular expressions in template parameters.

```
package com.someshop;

import javax.ws.rs.PathParam;
import javax.ws.rs.GET;
import javax.ws.rs.Path;

//root resource class
@Path("/customers")
public class CustomerResource {
  //subresource method1
  @GET
  @Path("{id : \\d+}")
  public String getCustomer(@PathParam("id") int id) {
    // Implementation to return appropriate value
  }
//subresource method2
@GET
  @Path("{path : .+}")
  public String getCustomerIdAndName(@PathParam("path") String path) {
    //execution to return the assigned value
  }
}
```

In this example, the expression {id : \\d+} to be included in the `Path` annotation is a template parameter that uses a regular expression. The identifier id and the regular expression "\\d+" are used together. A colon (:) separates the identifier and the regular expression.

The regular expression "\\d+" matches with one or more digits. The HTTP request corresponding to the URL "`http://someshop.com/resource/customers/333`" is dispatched to the sub-resource method `getCustomer()`.

The regular expression " +" matches with any character. The HTTP GET request corresponding to the URL "`http://someshop.com/resource/customers/33/John/Smith`" is dispatched to the sub-resource method `getCustomerIdAndName()`.

(b) Notes when using template parameters

The system returns an error (KDJJ10006-E) in the following cases:

- If invalid characters are used in template parameters
- If invalid syntactical regular expressions are written in template parameters

The system returns an HTTP response with the HTTP status code `500` in the root resource class and throws `java.lang.RuntimeException`, which can be handled by the exception mapping provider, in the sub-resource class (KDJJ10006-E).

17.1.7 Sub-resource class

A *sub-resource class* is a Java class that contains any one of the resource methods, sub-resource methods, or sub-resource locators, and is not annotated by the `Path` annotation at the class level.

An example of a sub-resource class is as follows.

```
package com.sample.resources;

import javax.ws.rs.GET;
import javax.ws.rs.Path;

public class Resource {
  @Path("/subresourcemethod1")
  @GET
  public String subResourceMethod1() {
    return "from sub resource method1";
  }
  @GET
  public String resourceMethod() {
    return "from resource method";
  }
}
```

The JAX-RS engine does not generate an instance of a sub-resource class. The sub-resource class must be instantiated with a corresponding sub-resource locator.

(1) Mechanism

A sub-resource class is generated in the following way. The generated sub-resource class processes an HTTP request as follows:

1. The HTTP request is dispatched to the sub-resource locator.
2. The sub-resource locator generates a sub-resource class and delegates the processing of the HTTP request to the generated sub-resource class.
3. The sub-resource class directly processes the HTTP request, or the request is further delegated to a sub-resource class in the same way.

For details on sub-resource locators, see *17.1.1(6) Sub-resource locators*.

The JAX-RS engine treats the instance returned by the sub-resource locator at runtime as a sub-resource class and not the return value type declared in the method signature of the sub-resource locator.

For example, assume that there are three sub-resource classes - M, N and O. N inherits M and O inherits N. In the same way, assume a sub-resource locator named R that contains the return value M. When a sub-resource locator returns an instance of M, the sub-resource class M executes the HTTP request. Similarly, when the sub-resource locator returns an instance of N, the sub-resource class N executes the HTTP request. When the system returns an instance of O, the sub-resource class O executes the HTTP request.

(2) Life cycle

A JAX-RS engine does not generate an instance of a sub-resource class. The sub-resource class must be instantiated with the corresponding sub-resource locator. Accordingly, the bean properties, fields, and parameters of the constructor must be initialized with a sub-resource locator or a sub-resource class.

(3) Constructor

Do not use annotations of the JAX-RS specifications in the parameters of the constructor of a sub-resource class. All such annotations are ignored, if used.

(4) Fields and bean properties

Do not use annotations of the JAX-RS specifications in the fields and the bean properties of a sub-resource class. All such annotations are ignored, if used.

(5) Resource methods, sub-resource methods, and sub-resource locators

A resource method of a sub-resource class, a sub-resource method, and a sub-resource locator match with the resource methods of the root resource class except for the differences explained hereafter. For details on root resource classes, see the following sub-sections:

- *17.1.1(4) Resource method*
- *17.1.1(5) Sub-resource method*
- *17.1.1(6) Sub-resource locator*

If the return value type of a sub-resource locator is `void`, an error occurs and the HTTP request sent by the client is not processed. 500 is returned as the HTTP status code. Note that you must confirm the J2EE server log file instead of the JAX-RS functionality log file.

When the following conditions hold true, the (KDJJ10006-E) error occurs, and consequently the system throws `java.lang.RuntimeException`, which can be handled by the exception mapping provider.

- When two or more request method designators are used for one resource method
- When the same request method designator is used for two or more resource methods
- When a sub-resource locator contains an entity parameter

17.1.8 Exception handling

The JAX-RS engine handles the exceptions thrown from the following locations as explained in this sub-section:

- Resource method
- Sub-resource method
- Sub-resource locator
- Sub-resource class and constructor of the root resource class

(1) WebApplicationException (When no exception mapping provider exists)

If the system throws `WebApplicationException` and no exception mapping provider corresponding to `WebApplicationException` or the respective parent exists, the JAX-RS engine handles `WebApplicationException` as described in the following table.

Table 17-10: Handling WebApplicationException (When no exception mapping provider exists)

No.	Condition		Handling result	
	Response property	HTTP status code of the response property	HTTP status code of the HTTP response	Message ID
1	Set	• 499 or less • The values of the enumerated type `Response.Status`	The value held by the `response` property of `WebApplicationException` is used.	`KDJJ30021-I`
2	Set	• 499 or less • Values that do not exist in the enumerated type `Response.Status`	The value held by the `response` property of `WebApplicationException` is used.	`KDJJ30022-I`
3	Set	• 500 or more • The values of the enumerated type `Response.Status`	The value held by the `response` property of `WebApplicationException` is used.	`KDJJ10018-E`
4	Set	• 500 or more	The value held by the `response` property of	`KDJJ10019-E`

No.	Condition		Handling result	
	Response property	HTTP status code of the response property	HTTP status code of the HTTP response	Message ID
4	Set	• Values that do not existing in the enumerated type `Response.Status`	`WebApplicationException` is used.	KDJJ10019-E
5	Not set	--	500	KDJJ10018-E

Legends:
--: Not applicable

An example of generating `WebApplicationException` and setting the response property is as follows.

```
package com.sample.resources;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.WebApplicationException;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.Response.ResponseBuilder;

//root resource class
@Path("/root")
public class Resource {

  //subresource method
  @Path("/subresourcemethod")
  @GET
  public String subResourceMethod() {
    //Use the ResponseBuilder to generate the Response instance
    ResponseBuilder rb = Response.status(208).
    entity("entity for WebApplicationException");

    //set the Response instance to the WebApplicationException
    throw new WebApplicationException(rb.build());
}
}
```

You consider the context root of the Web application (WAR file) containing the root resource class `com.sample.resources.Resource` to be "*resource*" and that the Web application is published on a host called "*example.com*". In this case, the HTTP GET request corresponding to the URL "`http://example.com/resource/root/subresourcemethod`" is dispatched to the method `subResourceMethod()`. The HTTP response is converted from the response property of `WebApplicationException`.

(2) Other exceptions (When no exception mapping provider exists)

If system throws an exception other than `WebApplicationException` and if the exception mapping provider corresponding to the `WebApplicationException` exception or the respective parent does not exist, the JAX-RS engine handles `WebApplicationException` as described in the following table:

Table 17-11: Other exceptions (When no exception mapping provider exists)

No.	Conditions	Handling result		
	Exception type	Operation by the JAX-RS engine	HTTP status code of the HTTP response	Message ID
1	Runtime exception	Throws a runtime exception again.	500	KDJJ10010-E, KDJJ10039-E
2	Other than the aforementioned	Wraps the exception in RuntimeException and then throws the exception.	500	KDJJ10017-E, KDJJ10039-E

(3) If the exception mapping provider exists

If the exception mapping provider corresponding to the thrown exception or respective parent exists, the exception handling depends on the operations of the exception mapping provider. Note that if multiple exception mapping providers corresponding to the thrown exception or respective parent exist, the exception mapping provider that can handle the exception closest to the exceptions (including the thrown exceptions) handles the exception.

17.1.9 Media type declaration

You can respectively use the `Consumes` annotation and `Produces` annotation to specify the MIME media types supported by the Web resource. When these annotations are not used, all the media types are considered to be supported.

The `Consumes` annotation and `Produces` annotation can be used in:

- A root resource class (class level)
- A sub-resource class (class level)
- A resource method (method level)
- A sub-resource method (method level)

The annotations used at the method level take precedence over the annotations used at the class level.

When two or more resource methods or sub-resource methods can process the same MIME media type, and if the request method designators, paths, or other information matches, the error (KDJJ10006-E) occurs. The system returns an HTTP response with `500` as the HTTP status code, in the root resource class and throws `java.lang.RuntimeException`, which can be handled by the exception mapping provider in the sub-resource class.

When the Content-Type header of an HTTP request does not match with any of the `Consumes` annotations, the (KDJJ10040-E) error occurs and the system throws `javax.ws.rs.WebApplicationException`, which has 415 as the HTTP status code and can be handled by the exception mapping provider.

If the `HTTP Accept` header of an HTTP response does not match with any of the `Produces` annotations, the (KDJJ10041-E) error occurs and the system throws `javax.ws.rs.WebApplicationException`, which has `406` as the HTTP status code and can be handled by the exception mapping provider.

An example of a media type declaration is as follows.

```
package com.sample.resources;

java.awt.image.RenderedImage
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.Consumes;

@Path("sample")
@Produces("image/jpeg")
public class ImageBasedResource {

@GET
  public RenderedImage getAsImage() {
  //implementation
  }

  @GET
  @Produces("text/html")
  public String getAsHtml() {
    //implementation
  }

  @POST
  @Consumes("image/jpeg")
  public void addWidget(RenderedImage image) {
    //implementation
  }
}
```

In this example, the resource method `getAsImage()` is called to process the HTTP GET request that requests the HTTP response of the MIME media type image/jpeg.

Furthermore, the resource method `getAsHtml` is called to process the HTTP GET request that requests the HTTP response of the MIME media type text/html.

Additionally, the resource method `addImage` is called to process the HTTPPOST request containing the entity body of the MIME media type image/jpeg.

17.1.10 Disabling URL decoding

- By default, the JAX-RS engine automatically decodes the URL encoded values when injecting the values to the parameters, fields, and bean properties annotated by the following annotations. Use the `Encoded` annotation in combination with each of the following annotations if you want to use the original non-decoded values.
- `MatrixParam` annotation
- `QueryParam` annotation
- `PathParam` annotation

17.1.11 Inheriting annotations

The child classes or implementation classes inherit the annotations of the JAX-RS specifications used in the methods of an interface or a parent class.

The conditions for inheriting the annotations are as follows:

- When the annotations of the JAX-RS specifications are not used in the methods of child class and respective parameters
- When the annotations of the JAX-RS specifications are not used in methods of the implementation class and respective parameters

When a parent class is inherited and an interface has been implemented, if both of the above conditions for inheriting annotations are fulfilled, the annotations of the parent class take precedence.

When multiple parent classes are inherited and multiple interfaces have been implemented, the annotation of the parent class that was inherited first or that of the interface that was implemented first should get precedence.

An example of inheriting annotations is as follows.

```
package com.sample.resources;

import javax.ws.rs.GET;
import javax.ws.rs.QueryParam;

//interface
public interface A {

  //method that uses the annotation of JAX-RS specifications
  @GET
  public String getValue(@QueryParam("query") String query);

}

package com.sample.resources;

import javax.ws.rs.Path;

//root resource class that implements the interface
@Path("/root/")
public class Resource implements A {

  //implementation of method
  public String getValue(String query) {
    //implementation

  }
}

package com.sample.resources;
```

```
import javax.ws.rs.Path;
import javax.ws.rs.Produces;

//root resource class that implements the interface
@Path("/root1/")
public class Resource1 implements A {

  //implementation of method
  @Produces("text/xml")
  public String getValue(String query) {
    //implementation
  }
}
```

In this example, consider the context root of the Web application (WAR file) containing the interface `com.sample.resources.A` and the root resource classes `com.sample.resources.Resource` and `com.sample.resources.Resource1` to be "*resource*" and that the Web application is published on a host named "*example.com*".

The HTTP GET request corresponding to the URL "`http://example.com/resource/root?query=10`" is dispatched to the resource method `getValue()`. This is because the resource method `getValue()` inherits the GET annotation of the interface.

However, the HTTP GET request corresponding to the URL "`http://example.com/resource/root1?query=10`" is not dispatched to the resource method `getValue()` and the system returns `500` as the HTTP status code. This is because the resource method `getValue()` does not inherit the GET annotation of the interface.

17.2 Provider

Provider is a class that implements the interface defined as per the standard specifications and is annotated by a Provider annotation.

Provider is instantiated for each Web application (WAR file). The life cycle of a provider is as follows:

1. The constructor is called
2. The required injection is performed
3. An appropriate method is called
4. Subjected to GC (Garbage collection)

17.2.1 Entity provider

Built-in entity providers are provided in the JAX-RS functionality.

17.2.2 Exception mapping provider

Implement an exception mapping provider when customizing mapping between the exceptions that can be processed by the exception mapping provider and the HTTP responses.

Implements the `ExceptionMapper<T>` interface for the exception mapping provider and annotate the interface with the `Provider` annotation.

An example of the exception mapping provider is as follows.

```
package com.sample.providers;

import javax.ws.rs.core.Response;
import javax.ws.rs.ext.ExceptionMapper;
import javax.ws.rs.ext.Provider;

//exception mapping provider to customize the mapping of the RuntimeException to the
HTTP response
@Provider
public class RuntimeExceptionMapper implements
    ExceptionMapper<RuntimeException> {

  public Response toResponse(RuntimeException runtimeException) {
    int httpStatus = 0;
    String entity = "";

    //Use the ResponseBuilder class to create the HTTP response
    return Response.status(httpStatus).entity(entity).build();
  }
}
```

In this example, the provider `com.sample.providers.RuntimeExceptionMapper` is the exception mapping provider. The `ExceptionMapper` interface with `RuntimeException` specified in the type parameter is implemented. Here, the HTTP response is created by calling the `toResponse` method of the `ResponseBuilder` class.

Note that only one exception mapping provider can be created for one exception. If two or more exception mapping providers are created for one exception, the (KDJJ10028-E, KDJJ10039-E) errors occur and the exception mapping provider is not instantiated. The system returns `500` as the HTTP status code.

(1) Constructor

The exception mapping provider must contain at least one public constructor including the default public constructor (a constructor that is not explicitly declared).

If no public constructor is declared, the (KDJJ10002-E, KDJJ10006-E) errors occur and the exception mapping provider is not instantiated. The system returns `500` as the HTTP status code.

The following constructors are not public constructors:

- `private` constructor
- `protected` constructor
- A constructor without an access identifier

For constructor parameters, you can use the `Context` annotation as an injection annotation. If an injection annotation other than the `Context` annotation is used, the (KDJJ10006-E) error occurs and the exception mapping provider is not instantiated. The system returns `500` as the HTTP status code. The following injection annotations cannot be used:

- `MatrixParam`
- `QueryParam`
- `PathParam`
- `CookieParam`
- `HeaderParam`
- `FormParam`

If you specify an annotation other than an injection annotation for these parameters, the (KDJJ10006-E) error message is output and the system returns `500` as the HTTP status code.

If the exception mapping provider contains one or more public constructors with parameters, the JAX-RS engine uses the constructor with maximum parameters to instantiate the exception mapping provider. When there are two or more constructors with maximum parameters, the JAX-RS engine uses the initially defined constructor to instantiate the exception mapping provider. In such cases, the warning message (KDJJ20011-W) is output to the log.

If a system throws a runtime exception when the exception mapping provider is instantiated, the (KDJJ10028-E, KDJJ10039-E) errors occur and 500 is returned as the HTTP status code.

(2) Fields and bean properties

You can use `Context` annotation as the injection annotation for the fields and bean properties of the exception mapping provider.

If you use an injection annotation other than the `Context` annotation, that annotation is ignored. You cannot use the following annotations.

- `MatrixParam`
- `QueryParam`
- `PathParam`
- `CookieParam`
- `HeaderParam`
- `FormParam`

18 Mapping JSON and POJO

This chapter describes how the built-in entity providers, which the JAX-RS functionality provides, map the JSON and POJO.

Do not use the annotation of the JAXB specifications for a POJO. If you use the annotation, the actual operations might differ from the following description.

18.1 Settings for mapping JSON and POJO

The JSON and POJO mapping enables data conversion from POJO to JSON and vice versa.

This section describes how to map the JSON and POJO on a server and client.

18.1.1 Mapping on a server

To enable or to disable the JSON and POJO mapping:

- Common definition file (`cjrconf.properties`)

  Specify `true` or `false` for the `com.sun.jersey.api.json.POJOMappingFeature` feature. This value is not case-sensitive.
- Initialization parameter of a `web.xml` servlet

  You can set the following values for the servlet initialization parameter (`com.sun.jersey.api.json.POJOMappingFeature`). This value is not case-sensitive.

  Table 18-1: Values available for the servlet initialization parameter (`com.sun.jersey.api.json.POJOMappingFeature`)

Value	Explanation
`true`	Enables the JSON and POJO mapping.
`false`	Disables the JSON and POJO mapping.
Values other than `true` and `false`	The value specified as the initialization parameter of a `web.xml` servlet is ignored and the value specified in the common definition file is used.

If a value is specified in both the common definition file and as the servlet initialization parameter, the value specified in the servlet initialization parameter is preferred.

18.1.2 Mapping on a client

Specify `true` or `false` for the `com.sun.jersey.api.json.POJOMappingFeature` feature of the common definition file (`cjrconf.properties`) or add the `com.sun.jersey.api.json.POJOMappingFeature` feature to the property map of the `client` object.

The following example shows how to add the `com.sun.jersey.api.json.POJOMappingFeature` feature:

```
// Generate a ClientConfig object
ClientConfig cc = new DefaultClientConfig();
// Add features to the ClientConfig object
// to enable the JSON POJO mapping
cc.getFeatures().put(JSONConfiguration.FEATURE_POJO_MAPPING, true);
// Specify the generated ClientConfig object
// to generate a Client object
Client client = Client.create(cc);
```

In the above example, a `Client` object is generated by specifying `ClientConfig` to which `JSONConfiguration.FEATURE_POJO_MAPPING` is already added.

If a value is specified in the common definition file and the `Client` object, the value specified in the `Client` object is given preference.

18.2 POJO to JSON mapping

For details on the JSON format, see *RFC 4627*. This section describes the requirements for the POJO to JSON mapping and the available data types.

18.2.1 Requirements for mapping

Requirements for the POJO to JSON mapping:

(1) POJO

- Define the POJO class as a public or package scope. You can also specify the `final` modifier.
- You can include any constructors.
- If you map multiple fields or properties to the same JSON element, the `JsonMappingException` exception is thrown. For details on exception handling, see *18.3.3 Exception handling*.
- If a field name is same as that of a Bean property name, the Bean property name is preferred.

(2) Field

- Define the field with a public scope. Otherwise, mapping will not proceed.
- Do not specify a `static` or `transient` modifier in the field. Otherwise, mapping will not proceed.
- You can also specify the `final` modifier.

(3) Bean property

- Define the `getter` method of `Bean property` as public scope. Otherwise, mapping will not proceed.
- If you define multiple `getter` methods with the same name in upper and lower cases for the same property, the `JsonMappingException` exception is thrown. For details on exception handling, see *18.3.3 Exception handling*.
- Do not specify `static` in `Bean property`. Otherwise, mapping will not proceed.
- You can also specify the `final` modifier.
- The `Bean property` does not need to be writable. For details on the Bean property, see the *JavaBeans specifications*.

18.2.2 Available data types

You can specify the following data types for the POJO fields and Bean properties.

Table 18-2: Available data types for the POJO fields and Bean properties (POJO to JSON mapping)

No.	Data	
1	Primitive	`int`
2		`short`
3		`long`
4		`float`
5		`double`
6		`char`
7		`byte`

No.	Data	
8	Primitive	`boolean`
9	Wrapper class	`Integer`
10		`Short`
11		`Long`
12		`Float`
13		`Double`
14		`Character`
15		`Byte`
16		`Boolean`
17	`java.lang.String`	
18	`java.math.BigInteger`	
19	`java.math.BigDecimal`	
20	`java.util.Date`	
21	`java.util.Calendar`	
22	`java.lang.Enum`	
23	`POJO`#1	
24	`java.util.List<`*T*`>`#2	
25	`java.util.Set<`*T*`>`#2	
26	`java.util.Map<`*T,T*`>`#2	
27	An Array containing any item from No.1 through 26#2	

#1

You can recursively have a POJO as a field or aBean property. For conditions of a supported POJO, see *18.2 POJO to JSON mapping*.

#2

The type of T is any one item listed in No. 1 through 26 in the above table.

Notes:

- If a field or a Bean property is not initialized, the default value of the respective data types (the respective default value for the primitive types, `null` for object type) is mapped to the generated JSON.
- If the value of the types in No. 9 through 27 in the above table is `null`, `null` is mapped to the corresponding value of the generated JSON.
- If the value of the types in No.24 through 27 in the above table includes `null`, `null` is mapped to the corresponding value of the generated JSON.
- If the value of the types in No.20 or 21 in the table is other than `null`, the corresponding value of the generated JSON is mapped to a value same as the value expressed in milliseconds. For example, if you are using `Date` as a data type, the value acquired on calling the `getTime()` method of the `Date` class is mapped. If you are using the `Calendar` type, the value acquired on calling the `getTime()` method of the `Date` class is mapped to the `Date` object acquired on calling the `getTime()` method of the `Calendar` class.
- If the value of the type in No.26 in the above table is `null`, the `JsonMappingException` exception is thrown. For details on exception handling, see *18.3.3 Exception handling*.

- If No. 27 of the table is the `char` array or `byte` array, the corresponding value of the generated JSON is mapped with the next respective value, and not with the array.
  - The `char` array: character string generated from an array
  - The `byte` array: Base64 encoded character string generated from an array

  The following are the examples:
  - The `Bean` property of `char[]` type having value {*a*,*b*} and name "`bean`" is mapped to {"`bean`":["`ab`"]} and not {"`bean`":["`a`","`b`"]}.
  - The Bean property of `byte[]` type having value {*1*,*2*} and name "`bean`" is mapped to {"`bean`":["`AQ==`"]} and not {"`bean`":[*1*,*2*]}.

18.2.3 Exception handling

The POJO to JSON mapping uses a mechanism of the entity provider of the JAX-RS specifications. The exceptions thrown during the POJO to JSON mapping, therefore, is handled in the same way as the exceptions thrown during any other mapping. For details on how exceptions are handled during mapping, see *17.1.3 Return value* and for mapping on servers and for mapping on clients, see *25.15.1 Combination of Java types and MIME media types available for an HTTP request entity*.

18.3 JSON to POJO mapping

For details on the JSON format, see *RFC 4627*. This section describes the requirements for the JSON to POJO mapping and the available data types.

18.3.1 Requirements for mapping

(1) POJO

- Define the `POJO` class as a public or package scope. You can also specify the `final` modifier.
- The `POJO` class must have a default constructor. Declaring a default constructor explicitly is optional. You can also use any of the public, private, protected, or package scopes. If the `POJO` class does not have a default constructor, the `JsonMappingException` exception is thrown. As per the Java language specifications, you must declare the default constructor explicitly if you declare a constructor with a parameter. For details on exception handling, see *18.3.3 Exception handling*.
- The `POJO` class might have a Bean property or a field of the types declared as an inner class. Do not use an inner class as an interface or a non-static class. For a Bean property or a field of a type declared as an inner class that is an interface or a non-static class, the `JsonMappingException` exception is thrown. For details on exception handling, see *18.3.3 Exception handling*.
- If a value that cannot be mapped to a POJO is in the JSON format, the `JsonMappingException` exception is thrown. For exception handling, see *18.3.3 Exception handling*.
- If a field name is same as that of a Bean property name, the Bean property is preferred.
- Only when the JSON format is blank, error does not occur even if the JSON format has no fields or Bean properties to be mapped to the POJO.

(2) Field

- Define the field as public scope. Otherwise the `JsonMappingException` exception is thrown. For details on exception handling, see *18.3.3 Exception handling*.
- Do not specify a `static`, `transient`, or `final` modifier in the field. These modifiers will not be mapped.
- The fields that do not have a value corresponding to the JSON format will not be initialized (exception will not be thrown).

(3) Bean property

- We recommend that you define the `setter` method of the Bean property as a public scope. You can also use private, protected, or package scope to declare the `setter` method of the Bean property.
- Do not declare multiple `setter` methods with the same name in upper and lower case. If you declare, the `JsonMappingException` exception is thrown. For details on exception handling, see *18.3.3 Exception handling*.
- Do not specify `static` for the Bean property. `static` will not be mapped.
- Specifying the `final` modifier is optional.
- The Bean property need not be readable. For details on the Bean property, see the *JavaBeans specifications*.
- The `setter` method of a Bean property that does not have a value corresponding to the JSON format will not be called (exception will not be thrown).

18.3.2 Available data types

You can specify the following data types for the POJO fields and the Bean property.

Table 18-3: Available data types for the POJO fields and the Bean property (JSON to POJO mapping)

No.	Data	
1	Primitive	`int`
2		`short`
3		`long`
4		`float`
5		`double`
6		`char`
7		`byte`
8		`boolean`
9	Wrapper class	`Integer`
10		`Short`
11		`Long`
12		`Float`
13		`Double`
14		`Character`
15		`Byte`
16		`Boolean`
17	`java.lang.String`	
18	`java.math.BigInteger`	
19	`java.math.BigDecimal`	
20	`java.util.Date`	
21	`java.util.Calendar`	
22	`java.lang.Enum`	
23	`POJO`[#1]	
24	`java.util.List<`*T*`>`[#2]	
25	`java.util.Set<`*T*`>`[#2]	
26	`java.util.Map<`*K*`,`*V*`>`[#2]	
27	Any array in 1 through 26[#2,#3]	

#1

You can recursively have the POJO as a field or a Bean property. For conditions of a supported POJO, see *18.3 JSON to POJO mapping*.

#2

The type of T, K, and V can be any item listed in No.1 through 26 in the above table.

#3

The value of the corresponding JSON format must conform to the array structure.

Points to note when setting elements:

- If the JSON format has any of the following errors, the `JsonParseException` exception is thrown. For details on exception handling, see *18.3.3 Exception handling*.

- If the JSON format contains a value other than a numeric value, character string, array, object, `true`, `false`, or `null`
- If two objects are not separated with a comma
- If a character that is not allowed is used among tokens (spaces, linefeeds, carriage returns, and horizontal tabs are allowed)
- If the format of a numeric value is incorrect (such as 12, 1.2eE8, 0X3F7A)

- If the data type is not a numerical field (1-5, 7, 9-13, 15 of the table) or the JSON format value corresponding to the Bean property is not a numerical value, the `JsonMappingException` exception is thrown. For details on exception handling, see *18.3.3 Exception handling*.

(1) int type(primitive)

- If the value of the corresponding JSON format is a blank character string, the `JsonMappingException` exception is thrown. For details on exception handling, see *18.3.3 Exception handling*.
- If the value of the corresponding JSON format is `null`, the field or the Bean property is initialized with the default value of the `int` type.
- The value of the corresponding JSON format must be a number within the range of the `int` type. If you specify a value outside the range, the operation is not guaranteed.

(2) short type (primitive)

- If the corresponding value of the JSON format is `null` or a blank character string, the field or the Bean property is initialized with the default value of the `short` type.
- The value of the corresponding JSON format must be a number within the range of the `short` type. If you specify a value outside the range, the operation is not guaranteed.

(3) long type (primitive)

- If the value of the corresponding JSON format is a blank character string, the `JsonMappingException` exception is thrown. For details on exception handling, see *18.3.3 Exception handling*.
- If the value of the corresponding JSON format is `null`, the field or the Bean property is initialized with the default value of the `long` type.
- The value of the corresponding JSON format must be a number within the range of the `long` type. If you specify a value outside the range, the operation is not guaranteed.

(4) float type (primitive)

- If the value of the corresponding JSON format is a blank character string, the `JsonMappingException` exception is thrown. For details on exception handling, see *18.3.3 Exception handling*.
- If the value of the corresponding JSON format is `null`, the field or the Bean property is initialized with the default value of the `float` type.
- The value of the corresponding JSON format must be a number within the range of the `float` type. If you specify a value outside the range, the operation is not guaranteed.

(5) double type (primitive)

- If the value of the corresponding JSON format is a blank character string, the `JsonMappingException` exception is thrown. For details on exception handling, see *18.3.3 Exception handling*.
- If the value of the corresponding JSON format is `null`, the field or the Bean property is initialized with the default value of the `double` type.
- The value of the corresponding JSON format must be a number within the range of the `double` type. If you specify a value outside the range, the operation is not guaranteed.

(6) char type (primitive)

- If the value of the corresponding JSON format is a blank character string, the `JsonMappingException` exception is thrown. For details on exception handling, see *18.3.3 Exception handling*.
- If the value of the corresponding JSON format is `null`, the field or the Bean property is initialized with the default value of the `char` type.
- The value of the corresponding JSON format must be a number within the range of the `char` type. If you specify a value outside the range, the operation is not guaranteed.

(7) byte type (primitive)

- If the value of the corresponding JSON format is `null` or a blank string expression, the field or the Bean property is initialized with the default value of the `byte` type.
- The value of the corresponding JSON format must be a number within the range of the `byte` type. If you specify a value outside the range, the operation is not guaranteed.

(8) boolean type (primitive)

- If the value of the corresponding JSON format is `null` or a blank character string expression, the field or the Bean property is initialized with the default value of the `boolean` type.
- If a value other than `true` or `false` (case-sensitive) is set to the corresponding JSON format, operations of the JAX-RS engine will not be defined.

(9) Integer type (wrapper class)

- If the value of the corresponding JSON format is `null` or a blank character string expression, `null` is mapped.
- The value of the corresponding JSON format must be a number within the range of the `Integer` type. If you specify a value outside the range, the operation is not guaranteed.

(10) Short type (wrapper class)

- If the value of the corresponding JSON format is a blank character string expression, the field or the Bean property is initialized with the default value of the `Short` type.
- If the value of the corresponding JSON format is `null`, `null` is mapped.
- The value of the corresponding JSON format must be a number within the range of the `Short` type. If you specify a value outside the range, the operation is not guaranteed.

(11) Long type (wrapper class)

- If the value of the corresponding JSON format is `null` or a blank character string expression, `null` is mapped.
- The value of the corresponding JSON format must be a number within the range of the `Long` type. If you specify a value outside the range, the operation is not guaranteed.

(12) Float type (wrapper class)

- If the value of the corresponding JSON format is `null` or a blank character string expression, `null` is mapped.
- The value of the corresponding JSON format must be a number within the range of the `Float` type. If you specify a value outside the range, the operation is not guaranteed.

(13) Double type (wrapper class)

- If the value of the corresponding JSON format is `null` or a blank character string expression, `null` is mapped.

- The value of the corresponding JSON format must be a number within the range of the `Double` type. If you specify a value outside the range, the operation is not guaranteed.

(14) Character type (wrapper class)

- If the value of the corresponding JSON format is a blank character string, the `JsonMappingException` exception is thrown. For details on exception handling, see *18.3.3 Exception handling*.
- If the value of the corresponding JSON format is `null`, the field or the Bean property is initialized with the default value of the `Char` type.
- The value of the corresponding JSON format must be a number within the range of the `Char` type. If you specify a value outside the range, the operation is not guaranteed.

(15) Byte type (wrapper class)

- If the value of the corresponding JSON format is a blank character string expression, the field or the Bean property is initialized with the default value of the `Short` type.
- If the value of the corresponding JSON format is `null`, `null` is mapped.
- The value of the corresponding JSON format must be a number within the range of the `Short` type. If you specify a value outside the range, the operation is not guaranteed.

(16) Boolean type (wrapper class)

- If the value of the corresponding JSON format is a blank character string expression, the field or the Bean property is initialized with the default value of the `Boolean` type.
- If the value of the corresponding JSON format is `null`, `null` is mapped.
- If a value other than `true`, `false` (case-sensitive) is set to the corresponding JSON format, operations of the JAX-RS engine will not be defined.

(17) java.lang.String type

- If the value of the corresponding JSON format is a blank character string expression, a blank character string expression is mapped.
- If the value of the corresponding JSON format is `null`, `null` is mapped.
- The program must be designed in such a way that quotation marks ("), if any, in the value of the corresponding JSON format must escape. If the quotation marks do not escape, the `JsonParseException` exception is thrown. For details on exception handling, see *18.3.3 Exception handling*.
- The program must be designed in such a way that control characters such as `\n`,`\r`,`\t`,and `\b`, if any, in the value of the corresponding JSON format must escape. If the control characters do not escape, the `JsonParseException` exception is thrown. For details on exception handling, see *18.3.3 Exception handling*.
- The program must be designed in such a way that the control character \u in the value of the corresponding JSON format, if existing, must escape. Furthermore, a character string or a hexadecimal value must follow this \u. If the above conditions are not met, the `JsonParseException` exception is thrown. For details on exception handling, see *18.3.3 Exception handling*.

(18) java.math.BigInteger type, java.math.BigDecimal type

- If the value of the corresponding JSON format is `null` or a blank character string expression, `null` is mapped.
- If the value of the corresponding JSON format is that other than a number, the `JsonMappingException` exception is thrown. For details on exception handling, see *18.3.3 Exception handling*.

(19) java.util.Datetype, java.util.Calendar type

- If the value of the corresponding JSON format is `null` or a blank character string expression, `null` is mapped.

- The standard supported formats are as follows:
  - `yyyy-MM-dd'T'HH:mm:ss.SSSZ`
  - `yyyy-MM-dd'T'HH:mm:ss.SSS'Z'`
  - `EEE, dd MMM yyyy HH:mm:ss zzz`
  - `yyyy-MM-dd`

  The following table lists pattern characters, describes their meaning, and gives an example wherever applicable:

Element	Meaning	Example
`yyyy`	Represents years with a four-digit number.	2013
`MM`	Represents months with a two-digit number.	04
`MMM`	Represents months with a string of three characters.	Apr
`dd`	Represents dates of month with a two-digit number.	30
`EEE`	Represents days of a week with a string of three characters.	Sun
`'T'`	Is a fixed character.	--
`HH`	Represents hours of a day with a two-digit number.	23
`mm`	Represents minutes with a two-digit number.	30
`ss`	Represents seconds with a two-digit number.	10
`SSS`	Represents milliseconds with a three-digit number.	978
`'Z'`	Is a fixed character.	--
`Z`	Represents a time zone defined in RFC 822.	-0530
`zzz`	Represents standard time.	IST

Legend:
--: No example is available for fixed characters.

For details on patterns of the `Date` or `Calendar` formats, especially `E` and `M` depending on the locale, see the information related to the `SimpleDateFormat` class in the *Java SE specifications*.

- If the value of the corresponding JSON format has an unsupported format, the `JsonMappingException` exception is thrown. For details on exception handling, see *18.3.3 Exception handling*.
- If the value of the corresponding JSON format is a number within the range of the `long` type, i.e. if the value corresponding to the `Date` or `Calendar` type is expressed in milliseconds, the objects of the `Date` or `Calendar` type initialized with their respective values in milliseconds are mapped. If the value is outside the range of the `long` type, the operation is not guaranteed.
- For example, if the value of the corresponding JSON format is `1346850421185`, the result (`Date` object) of calling the `Date`(long date) constructor of the `Date` class by specifying `1346850421185` in the `date` parameter is mapped to a field or a Bean property of the `Date` type. Similarly, if you create a `Date` object, specify that `Date` object in `date` parameter and call the `setTime(Date` *`date`*`)` method of the `Calendar` class, the result (`Calendar` object) is mapped to the field or the Bean property of the Calendar type.

(20) java.lang.Enum type

- If the value of the corresponding JSON format is a blank character string, the `JsonMappingException` exception is thrown. For details on exception handling, see *18.3.3 Exception handling*.
- If the value of the corresponding JSON format is `null`, `null` is mapped.
- If the value of the corresponding JSON format is not included in the `Enum` type, the `JsonMappingException` exception is thrown. For details on exception handling, see *18.3.3 Exception handling*.

(21) java.util.List<T> type

- If the value of the corresponding JSON format is an empty list ([]), an empty list is mapped.
- If the value of the corresponding JSON format is `null`, `null` is mapped.
- If the list of the corresponding JSON format contains `null`, `null` is set to the corresponding element in the field of the `List<`*T*`>` type or the Bean property to be mapped.
- The value included in the list of the corresponding JSON format must conform to the conditions of the type set to T. For conditions, see notes for the respective types in *18.3.2 Available data types*. These notes also describe the operations carried out if the conditions are not fulfilled.

(22) java.util.Set<T> type

- If the value of the corresponding JSON format is an empty list ([]), an empty set is mapped.
- If the value of the corresponding JSON format is `null`, `null` is mapped.
- If the list of the corresponding JSON format contains `null`, `null` is set to the corresponding element in the field of the `Set<`*T*`>` type or the Bean property type to be mapped.
- The value included in the corresponding JSON format set must conform to the conditions of the type set to T. For conditions, see notes for the respective types in *18.3.2 Available data types*. These notes also describe the operations carried out if the conditions are not fulfilled.

(23) java.util.Map<K,V> type

- If the value of the corresponding JSON format is a blank map ({}), an empty map is mapped.
- If the value of the corresponding JSON format is `null`, `null` is mapped.
- If the map of the corresponding JSON format includes `null` as a key, the `JsonParseException` exception is thrown. For details on exception handling, see *18.3.3 Exception handling*.
- If the map of the corresponding JSON format includes `null` as the value, `null` is set to the corresponding values of the fields of the `Map<`*K*`,`*V*`>` type to be mapped or in the Bean property.
- For a key or a value to be included in the map of the corresponding JSON format, the key or the value must conform to the conditions of the type to be set to K, V respectively. For conditions, see notes for the respective types in *18.3.2 Available data types*. These notes describe the operations carried out if the conditions are not fulfilled.

18.3.3 Exception handling

The JSON to POJO mapping uses a mechanism of the entity provider of the JAX-RS specifications. Therefore, exceptions thrown during the JSON to POJO mapping is handled in the same way as the exceptions thrown during any other mapping. For details on how exceptions are handled during mapping, see *17.1.3 Return value*, and for mapping on servers and clients, see *25.15.2 Combination of Java types and MIME media types available for an HTTP response entity*.

18.4 Exceptions that occur during mapping

The following table lists the exceptions that occur during mapping.

Table 18-4: Exceptions that occur during the JSON POJO mapping

No.	Interface or class
`com.cosminexus.org.codehaus.jackson.map` package	
1	`JsonMappingException`
`com.cosminexus.org.codehaus.jackson` package	
2	`JsonParseException`
3	`JsonGenerationException`

19 Support Range of the JAX-WS Specifications

This chapter describes the support range of the JAX-WS Specifications that you must consider when you develop a Web Service.

19.1 Support range of the JAX-WS 2.2 specifications

This section describes the support range of the functionality in the JAX-WS 2.2 specifications, and the support for Conformance.

19.1.1 Support range of the functionality in the JAX-WS 2.2 specifications

The following table describes the support range of the functionality in the JAX-WS 2.2 specifications.

For details on the preconditions, see *1.4.2 Preconditions related to the functionality and specifications*.

Table 19-1: Support range of the functionality in the JAX-WS 2.2 specifications

Division			Support	Remarks
Major division #1	Subdivision #2			
2	Mapping Java from WSDL 1.1		Y	
2	Customized mapping with an embedded binding declaration or an external binding file		Y	For details on the support range of the binding declarations, see *15.2 Customized mapping of WSDL to Java*.
2	Assigning annotations to the generated Java codes		Y	For details on the support range of the annotations, see *16.2.1 List of annotations*.
2.1	Mapping a Namespace to a package		Y	For details on mapping, see *15.1.1 Mapping a namespace to a package name*.
2.2	Mapping a port type to an SEI		Y	For details on mapping, see *15.1.2 Mapping a port type to an SEI name*.
2.3	Mapping an operation to an SEI method		Y	For details on mapping, see *15.1.3 Mapping an operation to a method name*.
2.3	Overloading of methods		Y	For details on the overloading of methods, see *15.1.11(1) Overloading of Java methods*.
2.3	MEP: Supporting a request-response operation		Y	
2.3	MEP: Supporting a one-way operation		Y	
2.3.1.1	Supporting the Non-wrapper style operations		Y	
2.3.1.2	Supporting the Wrapper style operations		Y	
2.3.2	Order and return value types of parameters	`parameterOrder` attribute	N	
		Attributes other than the `parameterOrder` attribute	Y	
2.3.3	`javax.xml.ws.Holder<T>` class		Y	For details on the support range of the `javax.xml.ws.Holder<T>` class, see *19.2.4(12) javax.xml.ws.Holder<T> class*.
2.3.4	Asynchronous mapping		N	

Division			Support	Remarks
Major division #1	Subdivision #2			
2.4.1	`W3CEndpointReference` class		Y	
2.5	Mapping a fault to a service-specific exception		Y	For details on mapping, see *15.1.7 Mapping a fault to an exception class*.
2.6	SOAP binding	SOAP 1.1/HTTP	Y	
		SOAP 1.2/HTTP	Y	
2.6	MIME binding		N	
2.6.2.1	`soap: header` element		Y	For details on the support range of the `soap: header` element (codeable syntax), see *20.1.22 soap: header element*.
2.7	Mapping a service and port to a service class		Y	For details on mapping, see *15.1.9 Mapping a service and port to a service class*.
2.8	Mapping an XML name to a Java identifier		Y	
2.8.1	Processing to be executed for a name conflict		Y	For details on the processing to be executed for a name conflict, see *15.1.11(2) Mapping for name conflict*.
3	Mapping Java to WSDL 1.1		Y	
3	Customized mapping using annotations		Y	For details on the support range of the annotations, see *16.2.1 List of annotations*.
3.1	Mapping a Java identifier to an XML name		Y	
3.2	Mapping a package to a Namespace		Y	
3.3	Mapping an implicit SEI to a port type		Y	
3.4	Mapping an SEI to a port type		Y	For details on mapping, see *16.1.3 Mapping the SEI name to the port type*.
3.5	Mapping an SEI method to an operation		Y	For details on mapping, see *16.1.4 Mapping the SEI method name to an operation*.
3.5.1	One-way operations		Y	
3.6	Mapping the parameters and return values		Y	For details on mapping, see *16.1.5 Mapping the parameter and return value to the message part (For wrapper style)* and *16.1.6 Mapping the parameter and return value to the message part (For non-wrapper style)*.
3.6	`soap: header` element		Y	
3.6.2.1	Mapping the Document Wrapped styles		Y	
3.6.2.2	Mapping the Document Bare styles		Y	
3.6.2.3	Mapping the RPC styles		N	
3.7	Mapping a service-specific exception to a fault		Y	For details on mapping, see *16.1.7 Mapping the Java wrapper exception class to a fault*.

Division			Support	Remarks
Major division [#1]	Subdivision [#2]			
3.8	Mapping a service class to a binding		Y	For details on mapping, see *16.1.8 Mapping SEI to binding*.
3.9	Processing of generics		Y	For details on the processing of generics, see *16.1.10(1) Deleting the generics type* and *16.1.10(3) Limitation of the generics type*.
3.10	SOAP/HTTP binding	SOAP 1.1/ HTTP	Y	
		SOAP 1.2/ HTTP	Y	
3.11	Mapping a service class to a service and port		Y	For details on mapping, see *16.1.9 Mapping a Web Service implementation class to a service and port*.
4	Client APIs		Y	For details on the support range of APIs, see *19.2 Support range of APIs*.
5	Service APIs		Y	For details on the support range of APIs, see *19.2 Support range of APIs*.
6	Core APIs		Y	For details on the support range of APIs, see *19.2 Support range of APIs*.
7	Annotations		Y	For details on the support range of annotations, see *16.2.1 List of annotations*.
8	Binding declarations		Y	For details on the support range of the binding declarations, see *15.2 Customized mapping of WSDL to Java*.
9.1.1	Logical handlers		Y	For details on the logical handlers, see *36.4 Types of handlers*.
9.1.1	Protocol handlers		Y	For details on the protocol handlers, see *36.4 Types of handlers*.
9.2.1.1	Handler settings in the Web Service client	Dynamic settings using APIs	Y	For details on the handler settings, see *36.9 Setting the handler chain*.
		Other settings	N	
9.2.1.2	Sequencing the handlers		Y	For details on the execution order of the handlers, see *36.5 Execution sequence and organization of the handler chain*.
9.2.1.3	Handler settings in the Web Services	Settings using the `javax.jws.HandlerChain` annotation	Y	For details on the handler settings, see *36.9 Setting the handler chain*.
		Other settings	N	
9.2.2	Deploying a handler based on the WSEE specifications (JSR-109)		N	
9.3	Handler processing model		Y	For details on the handler processing, see *36.5 Execution sequence and organization of the handler chain*.
9.4	Message context		Y	For details on the support range of the message context properties, see

Division		Support	Remarks
Major division #1	Subdivision #2		
9.4	Message context	Y	*19.2.5(1) Support range of the message context properties*.
10.1.1	Dynamic (programmed) settings for SOAP binding	Y	
10.1.2	Static (deployment-based) settings for SOAP binding	N	
10.2	Processing model for SOAP binding	Y	
10.3	SOAP message context	Y	For details on the support range of the message context, see *19.2.5 Using a message context*.
10.4.1	SOAP 1.1/ HTTP binding	Y	
10.4.1	SOAP 1.2/ HTTP binding	Y	
10.4.1.1	MTOM specifications	Y	
10.4.1.2	one-way operations	Y	
10.4.1.3	HTTP basic authentication	Y	
10.4.1.4	Session management	Y	
10.4.1.5	WS-Addressing specifications	Y	
11	XML/ HTTP binding	N	

Legend:

Y: Supported inthe Cosminexus JAX-WS functionality.

N: Not supported in the Cosminexus JAX-WS functionality.

Blank column: Indicates that there is no supplemented content.

#1

Indicates the relevant location (chapter, section, or subsection) in the JAX-WS 2.2 specifications.

#2

Indicates the contents described at the relevant location in the JAX-WS 2.2 specifications.

19.1.2 Supporting Conformance

The following table describes whether Conformance is supported. Note that the Conformance number applies to "Appendix A Conformance Requirements" in the JAX-WS 2.2 specifications.

Table 19-2: Supporting Conformance

Division		Support	Remarks
Number#1	Title #2		
2.1	WSDL 1.1 support	Y	
2.2	Customization required	Y	For details on the support range of the binding declarations, see *15.2 Customized mapping of WSDL to Java*.
2.3	Annotations on generated classes	Y	For details on the support range of annotations, see *16.2.1 List of annotations*.
2.4	Definitions mapping	Y	For details on the values specifiable in the `targetNamespace` attribute of the

Division			Support	Remarks
Number[#1]	Title [#2]			
2.4	Definitions mapping		Y	`wsdl:definitions` element, see *15.1.1(2) Conditions for namespace*.
2.5	WSDL and XML Schema import directives		Y	For details on the `wsdl:import` element, see *26.3 Format of the wsdl:import element*.
2.6	Optional WSDL extensions		Y	Conformance is supported, but the WSDL extended elements and attributes that are not provided in the JAX-WS 2.2 specifications are not supported.
2.7	SEI naming		Y	For details on the values specifiable in the `name` attribute of the `wsdl:portType` element, see *15.1.2(2) Conditions for the port type name*.
2.8	javax.jws.WebService required		Y	
2.9	javax.xml.bind.XmlSeeAlso required		Y	
2.10	Method naming		Y	For details on the values specifiable in the `name` attribute of the `wsdl:operation` element, see *15.1.3(2) Conditions for the operation name*.
2.11	javax.jws.WebMethod required		Y	
2.12	Transmission primitive support	one-way	Y	
		request-response	Y	
2.13	Using javax.jws.OneWay		Y	
2.14	Using javax.jws.SOAPBinding		Y	
2.15	Using javax.jws.WebParam		Y	For details on the values specifiable in the `name` attribute of the `wsdl:part` element, see *15.1.5(2) Conditions for part names*.
2.16	Using javax.jws.WebResult		Y	
2.17	Generating @Action		Y	
2.18	Generating @Action input		Y	
2.19	Generating @Action output		Y	
2.20	Generating @Action fault		Y	
2.21	Use of JAXB annotations		Y	For details on the support range of annotations, see *16.2.1 List of annotations*.
2.22	Non-wrapped parameter naming		Y	For details on the values specifiable in the `name` attribute of the `wsdl:part` element, see *15.1.5(2) Conditions for part names*.
2.23	Default mapping mode		Y	
2.24	Disabling wrapper style		Y	
2.25	Wrapped parameter naming		Y	For details on the values specifiable in the `name` attribute of the `wrapper` child element, see *15.1.4(2) Conditions for the wrapper child element name*.

Division			Support	Remarks
Number[#1]	Title [#2]			
2.26	Parameter name clash		Y	
2.27	Using javax.xml.ws.RequestWrapper		Y	
2.28	Using javax.xml.ws.ResponseWrapper		Y	
2.29	Use of Holder		Y	
2.30	Asynchronous mapping required		N	
2.31	Asynchronous mapping option		N	
2.32	Asynchronous method naming		--	Asynchronous-related functionality is not supported.
2.33	Asynchronous parameter naming		--	Asynchronous-related functionality is not supported.
2.34	Failed method invocation		--	Asynchronous-related functionality is not supported.
2.35	Response bean naming		--	Asynchronous-related functionality is not supported.
2.36	Asynchronous fault reporting		--	Asynchronous-related functionality is not supported.
2.37	Asynchronous fault cause		--	Asynchronous-related functionality is not supported.
2.38	JAXB class mapping		Y	
2.39	JAXB customization use		Y	
2.40	JAXB customization clash		Y	
2.41	javax.xml.ws.wsaddressing.W3CEndpointReference		Y	
2.42	javax.xml.ws.WebFault required		Y	
2.43	Exception naming		Y	For details on the values specifiable in the `name` attributes of the `wsdl:fault` and `wsdl:message` elements, see *15.1.7(2) Conditions for the fault name*.
2.44	Fault equivalence		Y	
2.45	Fault equivalence		Y	
2.46	Required WSDL extensions	SOAP	Y	
		MIME	N	
2.47	Unbound message parts		Y	
2.48	Duplicate headers in binding		Y	
2.49	Duplicate headers in message		Y	
2.50	Use of the MIME type information		--	MIME binding is not supported.
2.51	MIME type mismatch		--	MIME binding is not supported.
2.52	MIME part identification		--	MIME binding is not supported.
2.53	Service superclass required		Y	

Division		Support	Remarks
Number[#1]	Title [#2]		
2.54	Service class naming	Y	For details on the values specifiable in the `name` attribute of the `wsdl:service` element, see *15.1.9(2) Conditions for the service name and port name*.
2.55	javax.xml.ws.WebServiceClient required	Y	
2.56	Generated service default constructor	Y	
2.57	Generated service(WebServiceFeature ...) constructor	Y	
2.58	Generated service(URL) constructor	Y	
2.59	Generated service(URL,WebServiceFeature...) constructor	Y	
2.60	Generated service(URL,QName) constructor	Y	
2.61	Generated service(URL,QName,WebServiceFeature...) constructor	Y	
2.62	Failed getPort Method	Y	
2.63	javax.xml.ws.WebEndpoint required	Y	
3.1	WSDL 1.1 support	Y	For details on the support range of the WSDL 1.1 specifications, see *20.1 Support range of the WSDL 1.1 specifications*.
3.2	Standard annotations	Y	For details on the support range of annotations, see *16.2.1 List of annotations*.
3.3	Java identifier mapping	Y	
3.4	Method name disambiguation	Y	
3.5	Package name mapping	Y	For details on the values specifiable in the Java identifiers and annotations, see *16. Mapping Java to WSDL*.
3.6	WSDL and XML Schema import directives	Y	
3.7	Class mapping	Y	
3.8	portType naming	Y	For details on the values specifiable in the Java identifiers and annotations, see *16. Mapping Java to WSDL*.
3.9	Inheritance flattening	Y	
3.10	Inherited interface mapping	Y	Conformance is supported, but the mapping such as that described for Conformance is not performed with the Cosminexus JAX-WS functionality.
3.11	Operation naming	Y	For details on the values specifiable in the Java identifiers and annotations, see *16. Mapping Java to WSDL*.
3.12	Generating wsam:Action	Y	
3.13	One-way mapping	Y	
3.14	One-way mapping errors	Y	The one-way pattern is not supported.
3.15	Use of JAXB annotations	Y	
3.16	Overriding JAXB types empty namespace	Y	

Division		Support	Remarks
Number[#1]	Title [#2]		
3.17	Parameter classification	Y	
3.18	Parameter naming	Y	For details on the values specifiable in the Java identifiers and annotations, see *16. Mapping Java to WSDL*.
3.19	Result naming	Y	For details on the values specifiable in the Java identifiers and annotations, see *16. Mapping Java to WSDL*.
3.20	Header mapping of parameters and results	Y	
3.21	Dynamically generating wrapper beans	Y	
3.22	Default wrapper bean names	Y	For details on the values specifiable in the Java identifiers and annotations, see *16. Mapping Java to WSDL*.
3.23	Default wrapper bean package	Y	For details on the values specifiable in the Java identifiers and annotations, see *16. Mapping Java to WSDL*.
3.24	Wrapper element names	Y	For details on the values specifiable in the Java identifiers and annotations, see *16. Mapping Java to WSDL*.
3.25	Wrapper bean name clash	Y	
3.26	Default Wrapper wsdl:part names	Y	
3.27	Customizing Wrapper wsdl:part names	Y	
3.28	Wrapper property	Y	
3.29	Null Values in rpc/literal	--	The rpc/literal style is not supported.
3.30	Exception naming	Y	For details on the values specifiable in the Java identifiers and annotations, see *16. Mapping Java to WSDL*.
3.31	wsdl:message naming	Y	
3.32	wsdl:message naming using WebFault	Y	
3.33	java.lang.RuntimeExceptions and java.rmi.RemoteExceptions	Y	
3.34	Fault bean's @XmlType	Y	
3.35	Fault bean name clash	Y	
3.36	Dynamically generating exception beans	Y	
3.37	Binding selection	Y	
3.38	SOAP binding support	Y	
3.39	SOAP binding style required	Y	
3.40	Service creation	Y	
3.41	Port selection	Y	For details on the values specifiable in the Java identifiers and annotations, see *16. Mapping Java to WSDL*.
3.42	Port binding	Y	
3.43	Use of Addressing	Y	

Division		Support	Remarks
Number[#1]	Title [#2]		
4.1	Service completeness	N	Contains unsupported APIs.
4.2	Service Creation Failure	Y	
4.3	Service creation using features	Y	
4.4	Use of Executor	--	Asynchronous-related functionality is not supported.
4.5	Default Executor	--	Asynchronous-related functionality is not supported.
4.6	javax.xml.ws.BindingProvider.getEndpointReference	Y	
4.7	BindingProvider's W3CEndpointReference	Y	
4.8	Message context decoupling	Y	
4.9	Required BindingProvider properties	Y	
4.10	Optional BindingProvider properties	Y	
4.11	Additional context properties	Y	
4.12	Asynchronous response context	--	Asynchronous-related functionality is not supported.
4.13	Proxy support	Y	
4.14	Implementing BindingProvider	Y	
4.15	Service.getPort failure	Y	
4.16	Proxy's Addressing use	Y	
4.17	Remote Exceptions	Y	
4.18	Exceptions During Handler Processing	Y	
4.19	Other Exceptions	Y	
4.20	Dispatch support	Y	
4.21	Failed Dispatch.invoke	Y	
4.22	Failed Dispatch.invokeAsync	--	Asynchronous-related functionality is not supported.
4.23	Failed Dispatch.invokeOneWay	Y	Asynchronous-related functionality is not supported.
4.24	Reporting asynchronous errors	--	The `javax.xml.ws.Response` interface is not supported.
4.25	Marshalling failure	Y	
4.26	Use of the Catalog	Y	
5.1	Provider support required	Y	
5.2	Provider default constructor	Y	
5.3	Provider implementation	Y	
5.4	WebServiceProvider annotation	Y	
5.5	Endpoint publish(String address, Object implementor) Method	--	The `javax.xml.ws.Endpoint` class is not supported.

Division		Support	Remarks
Number[#1]	Title [#2]		
5.6	Default Endpoint Binding	--	The `javax.xml.ws.Endpoint` class is not supported.
5.7	Other Bindings	--	The `javax.xml.ws.Endpoint` class is not supported.
5.8	Publishing over HTTP	--	The `javax.xml.ws.Endpoint` class is not supported.
5.9	WSDL Publishing	--	The `javax.xml.ws.Endpoint` class is not supported.
5.10	Checking publishEndpoint Permission	--	The `javax.xml.ws.Endpoint` class is not supported.
5.11	Required Metadata Types	--	The `javax.xml.ws.Endpoint` class is not supported.
5.12	Unknown Metadata	--	The `javax.xml.ws.Endpoint` class is not supported.
5.13	Use of Executor	--	The `javax.xml.ws.Endpoint` class is not supported.
5.14	Default Executor	--	The `javax.xml.ws.Endpoint` class is not supported.
5.15	Endpoint's W3CEndpointReference	Y	
5.16	Building W3CEndpointReference	Y	
6.1	Read--only handler chains	Y	The `setHandlerChain` method of the `javax.xml.ws.Binding` interface is not supported.
6.2	Concrete javax.xml.ws.spi.Provider required	Y	
6.3	Provider createAndPublishEndpoint Method	--	The `javax.xml.ws.Provider` interface is not supported.
6.4	Concrete javax.xml.ws.spi.ServiceDelegate required	Y	
6.5	Protocol specific fault generation	Y	
6.6	Protocol specific fault consumption	Y	
6.7	One-way operations	Y	The one-way pattern is not supported.
6.8	javax.xml.ws.WebServiceFeatures	--	The `javax.xml.ws.WebServiceFeature` class is not supported.
6.9	enabled property	--	The `javax.xml.ws.WebServiceFeature` class is not supported.
6.10	javax.xml.ws.soap.MTOMFeature	Y	
6.11	javax.xml.ws.RespectBindingFeature	--	The `javax.xml.ws.RespectBindingFeature` class is not supported.
6.12	HTTP SPI in SE platform	N	The development and operations in the stand-alone Java SE are not supported with JAX-WS.

Division		Support	Remarks
Number[#1]	Title [#2]		
7.1	Correctness of annotations	Y	For details on the support range of annotations, see *16.2.1 List of annotations*.
7.2	Handling incorrect annotations	Y	For details on the support range of annotations, see *16.2.1 List of annotations*.
7.3	Unsupported WebServiceFeatureAnnotation	Y	
7.4	WebServiceProvider and WebService	Y	
7.5	JSR-181 conformance	Y	For details on the support range of annotations, see *16.2.1 List of annotations*.
8.1	Standard binding declarations	Y	
8.2	Binding language extensibility	Y	
8.3	Multiple binding files	Y	
9.1	Handler framework support	Y	
9.2	Logical handler support	Y	
9.3	Other handler support	Y	For details on the support range of APIs, see *19.2 Support range of APIs*.
9.4	Incompatible handlers	Y	
9.5	Incompatible handlers	Y	
9.6	Handler chain snapshot	Y	
9.7	HandlerChain annotation	Y	
9.8	Handler resolver for a HandlerChain annotation	Y	
9.9	Binding handler manipulation	Y	
9.10	Handler initialization	Y	
9.11	Handler destruction	Y	
9.12	Invoking close	Y	
9.13	Order of close invocations	Y	
9.14	Message context property scope	Y	
10.1	SOAP required roles	Y	
10.2	SOAP required roles	Y	
10.3	Default role visibility	Y	
10.4	Default role persistence	Y	
10.5	None role error	Y	
10.6	Incompatible handlers	Y	
10.7	Incompatible handlers	Y	
10.8	Logical handler access	Y	
10.9	SOAP 1.1 HTTP Binding Support	Y	
10.10	SOAP 1.2 HTTP Binding Support	Y	

Division		Support	Remarks
Number[#1]	Title [#2]		
10.11	SOAP MTOM Support	Y	
10.12	Semantics of MTOM enabled	Y	
10.13	MTOM support	Y	
10.14	SOAP bindings with MTOM disabled	Y	
10.15	SOAP bindings with MTOM enabled	Y	
10.16	MTOM on Other SOAP Bindings	--	Not applicable to this Conformance because the implementation of the other `javax.xml.ws.soap.SOAPBinding` interfaces is not supported.
10.17	One-way operations	Y	The one-way pattern is not supported.
10.18	HTTP basic authentication support	Y	
10.19	Authentication properties	Y	
10.21	URL rewriting support	N	
10.22	Cookie support	Y	
10.22	SSL session support	Y	Conformance is supported, but SSL session-based status management is not supported with the Cosminexus JAX-WS functionality.
10.23	SOAP Addressing Support	Y	
11.1	XML/HTTP Binding Support	N	
11.2	Incompatible handlers	N	
11.3	Incompatible handlers	N	
11.4	Logical handler access	N	
11.5	One-way operations	N	
11.6	HTTP basic authentication support	N	
11.7	Authentication properties	N	
11.8	URL rewriting support	N	
11.9	Cookie support	N	
11.10	SSL session support	N	

Legend:

Y: Supported.

N: Not supported.

--: Not applicable.

Blank column: Indicates that there is no supplemented content.

#1

Indicates the numbers in "*Appendix A Conformance Requirements*" in the JAX-WS 2.2 specifications.

#2

Title mentioned in Conformance.

19.2 Support range of APIs

19.2.1 List of interfaces and classes (JAX-WS)

This subsection describes the types of JAX-WS API interfaces and classes. This subsection also describes the support range of interfaces and classes.

(1) Types of interfaces and classes

The JAX-WS API interfaces and classes are classified into the following APIs:

- **Client API**
  The client API is used with the dispatch-based or API-based Web Service client. The stub-based Web Service client uses the service classes and stubs generated with commands to access the Web Services, so the client API is not used.
- **Service API**
  The service API is used for coding the advanced implementation in a Web Service. You use a service API to prepare a complex functionality, such as the Web Services and handlers that use the provider implementation class.
- **Core API**
  The core API is available with both, Web Services and Web Service client. The core API contains the `Holder` class for storing the `inout` and `out` parameters or exceptions.

(2) List of interfaces and classes

The following table describes a list of JAX-WS API interfaces and classes. The operations are not guaranteed if you use interfaces and classes that are not supported by the Cosminexus JAX-WS functionality to develop a Web Service.

Table 19-3: List of JAX-WS API interfaces and classes

No.	Interface or class name	Explanation	Support
`javax.xml.ws package`			
1	`AsyncHandler<T>`	--	N
2	`Binding`	--	N
3	`BindingProvider`	Interface that provides access to the context object associated with the protocol binding.	Y
4	`Dispatch<T>`	Interface for sending XML messages.	Y
5	`LogicalMessage`	Interface that expresses the XML message that is not captured in the protocol and includes the method that provides the method of accessing the message payload.	Y
6	`Provider<T>`	Interface for receiving XML messages.	Y
7	`Response<T>`	--	N
8	`WebServiceContext`	Interface that provides an access to the information associated with the current in-processing request for Web Services Implementation Class or Provider Implementation Class. Specify the `javax.annotation.Resource` annotation and then inject and use the information associated with the current in-processing requests.	Y
9	`Endpoint`	--	N

No.	Interface or class name	Explanation	Support
10	EndpointReference	Abstract class indicating the WS-Addressing EndpointReference of the remote reference for a Web Service endpoint.	Y
11	Holder<T>	Class that stores the value of type T.	Y
12	RespectBindingFeature	--	N
13	Service	Class that expresses the Web Service to be used by the Web Service client.	Y
14	WebServiceFeature	The user does not use this interface directly.	Y
15	WebServicePermission	--	N
16	ProtocolException	Class used to report the protocol level fault information to the client.	Y
17	WebServiceException	Exception class indicating a JAX-WS API runtime exception.	Y
javax.xml.ws.handler package			
18	Handler<C extends MessageContext>	Base interface of the handler.	Y
19	HandlerResolver	Interface implemented by the implementer of the Web Service client in order to control the handler chain set by proxy.	Y
20	LogicalHandler<C extends LogicalMessageContext>	Logical handler. Implement this interface when you want to implement the logical handler. This interface does not have methods.	Y
21	LogicalMessageContext	Message context for the logical handler.	Y
22	MessageContext	Interface that provides the methods for managing the property set.	Y
23	PortInfo	Information used to query the port for which the generation of the handler chain was requested by the handler resolver.	Y
javax.xml.handler.soap package			
24	SOAPHandler<T extends SOAPMessageContext>	SOAP handler. Implement this interface when you want to implement the SOAP handler.	Y
25	SOAPMessageContext	Message context for the SOAP handler.	Y
javax.xml.ws.http package			
26	HTTPBinding	--	N
27	HTTPException	--	N
javax.xml.ws.soap package			
28	SOAPBinding	Abstract class for SOAP binding.	Y
29	AddressingFeature	Feature class indicating that WS-Addressing is used.	Y
30	MTOMFeature	Feature class indicating that an attachment in the MTOM/XOP specification format is used.	Y
31	SOAPFaultException	Class indicating the SOAP fault exception.	Y
javax.xml.ws.spi package			

No.	Interface or class name	Explanation	Support
32	`Provider`	Abstract class that creates the `ServiceDelegate` and `Endpoint` objects. The user does not use this class directly.	Y
33	`ServiceDelegate`	Abstract class used internally by a `Service` object. The user does not use this class directly.	Y
`javax.xml.ws.wsaddressing package`			
34	`W3CEndpointReference`	Implementation class of the `EndpointReference` abstract class.	Y
35	`W3CEndpointReferenceBuilder`	Builder class used to create the `W3CEndpointReference` class.	Y
`com.sun.xml.ws.developer package`			
36	`StreamingAttachmentFeature`	Feature class indicating that streaming is used.	Y
37	`StreamingDataHandler`	Abstract class indicating an attachment that uses streaming.	Y
`org.jvnet.mimepull package`			
38	`MIMEConfig`	Class used to specify settings related to the parsing and output of MIME messages.	Y

Legend:

--: Indicates that there is no description (not-supported).

Y: Supported in the Cosminexus JAX-WS functionality.

N: Not supported in the Cosminexus JAX-WS functionality.

19.2.2 Client API

This subsection describes the support range of the client APIs.

(1) javax.xml.ws.BindingProvider interface

The following table describes the support range of the `javax.xml.ws.BindingProvider` interface. For details, see the *JAX-WS 2.2 specifications*.

Table 19-4: Support range of the javax.xml.ws.BindingProvider interface

No.	Return value type	Method name/ Explanation		Support
1	`Binding`	`getBinding()`		Y
		Description	Acquires binding of this binding provider.	
2	`EndpointReference`	`getEndpointReference()`		N
3	`<T extends EndpointReference> T`	`getEndpointReference (java.lang.Class<T> clazz)`		N
4	`java.util.Map <java.lang.String, java.lang.Object>`	`getRequestContext()`		Y
		Explanation	Obtains the context used for initializing the message context of the request message.	

No.	Return value type	Method name/ Explanation		Support
5	`java.util.Map <java.lang.String, java.lang.Object>`	`getResponseContext()`		Y
		Explanation	Obtains the context acquired by the processing of the response message. If this method is invoked before the response message is processed, `null` is returned.Do not call this method for one-way operations as response messages are not available. Operation is not guerenteed if you use this method in the one-way operations.	

Legend:

Y: Supported in the Cosminexus JAX-WS functionality.

N: Not supported in the Cosminexus JAX-WS functionality.

(2) javax.xml.ws.Dispatch interface

The following table describes the support range of the `javax.xml.ws.Dispatch` interface. For details, see the *JAX-WS 2.2 specifications*.

Table 19-5: Support range of the javax.xml.ws.Dispatch interface

No.	Return value type	Method name	Support
1	`T`	`invoke(T msg)`	Y
2	`Response<T>`	`invokeAsync(T msg)`	N
3	`java.util.concurrent.Future <?>`	`invokeAsync(T msg, AsyncHandler<T> handler)`	N
4	`void`	`invokeOneWay(T msg)`	Y

Legend:

Y: Supported in the Cosminexus JAX-WS functionality.

N: Not supported in the Cosminexus JAX-WS functionality.

(3) javax.xml.ws.EndpointReference class

The following table describes the support range of the `javax.xml.ws.EndpointReference` class. For details, see the *JAX-WS 2.2 specifications*.

Table 19-6: Support range of the javax.xml.ws.EndpointReference class

No.	Return value type	Method name	Support
1	--	`EndpointReference()`	Y
2	`<T> T`	`getPort (java.lang.Class<T> serviceEndpointInterface, WebServiceFeature... features)`	N
3	`static EndpointReference`	`readFrom (javax.xml.transform.Source eprInfoset)`	Y
4	`java.lang.String`	`toString()`	N
5	`abstract void`	`writeTo(javax.xml.transform.Result result)`	N

Legend:

--: Indicates that there is no return value type.

Y: Supported in the Cosminexus JAX-WS functionality.

N: Not supported in the Cosminexus JAX-WS functionality.

(4) javax.xml.ws.Service class

The following table describes the support range of the `javax.xml.ws.Service` class. For details, see the *JAX-WS 2.2 specifications*.

Table 19-7: Support range of the javax.xml.ws.Service class

No.	Return value type	Method name/ Explanation		Support
1	--	`Service(java.net.URL wsdlDocumentLocation, javax.xml.namespace.QName serviceName)`		Y
		Explanation	Constructor that generates the `Service` instance.	
		Argument	`wsdlDocumentLocation`: This is the location of the WSDL document. If `null` is specified, the value of the `wsdlLocation` attribute of the `javax.xml.ws.WebServiceClient` annotation, specified in the class that invokes the `Service` constructor, is set up. For details on the URL format to be specified, follow the specifications for the `java.net.URL` class. If the catalog functionality is active, this method maps the value specified in this argument to the URI that points to a different WSDL location specified in the catalog file. `serviceName`: This is the name of the service.	
		Exception	`javax.xml.ws.WebServiceException`: This exception occurs in the following cases: • When a non-existent local path name is specified in `wsdlDocumentLocation` • When a non-existent HTTP URL with `?wsdl` is specified in `wsdlDocumentLocation` • When a non-existent HTTP URL without `?wsdl` is specified in `wsdlDocumentLocation` • When `null` is specified in `serviceName` • When `QName` other than WSDL service name (`name` attribute value of the `wsdl:service` element) is specified in `serviceName` • When `null` is specified in `wsdlDocumentLocation` and the `javax.xml.ws.WebServiceClient` annotation is not used in the class that invokes the `Service` constructor • When `null` is specified in `wsdlDocumentLocation` and the `wsdlLocation` attribute is not set in the `javax.jws.WebService` annotation of the class that invokes the `Service` constructor	
2	`void`	`addPort(javax.xml.namespace.QName portName, java.lang.String bindingId, java.lang.String endpointAddress)`		Y
		Note	You must specify the same `QName` for `portName` when you invoke `createDispatch()`. You cannot specify `null`. If you specify `null` in `bindingId`, the SOAP1.1/HTTP binding ID is set up.	
3	`static Service`	`create(javax.xml.namespace.QName serviceName)`		Y
4	`static Service`	`create(javax.xml.namespace.QName serviceName, javax.xml.ws.WebServiceFeature ... features)`		N

No.	Return value type	Method name/ Explanation		Support
5	static Service	create(java.net.URL wsdlDocumentLocation, javax.xml.namespace.QName serviceName)		Y
		Explanation	Generates the Service instance.	
		Argument	wsdlDocumentLocation: This is the location of the WSDL document. For details on the URL format to be specified, follow the specifications for the java.net.URL class. serviceName: This is the WSDL service name (name attribute value of the wsdl:service element).	
		Exception	javax.xml.ws.WebServiceException: This exception occurs in the following cases: • When a non-existent local path name is specified in wsdlDocumentLocation • When a non-existent HTTP URL with ?wsdl is specified in wsdlDocumentLocation • When a non-existent HTTP URL without ?wsdl is specified in wsdlDocumentLocation • When a name other than the WSDL service name (name attribute value of the wsdl:service element) is specified in serviceName (however, except when null is specified in wsdlDocumentLocation)	
6	static Service	create(java.net.URL wsdlDocumentLocation, javax.xml.namespace.QName serviceName, javax.xml.ws.WebServiceFeature ... features)		N
7	<T> Dispatch<T>	createDispatch (EndpointReference endpointReference, java.lang.Class<T> type, Service.Mode mode, WebServiceFeature... features)		N
8	Dispatch <java.lang.Object>	createDispatch (EndpointReference endpointReference, javax.xml.bind.JAXBContext context, Service.Mode mode, WebServiceFeature... features)		N
9	<T> Dispatch<T>	createDispatch (javax.xml.namespace.QName portName, java.lang.Class<T> type, Service.Mode mode)		Y
		Note	In portName, you must specify the same port name as the name specified in the arguments of addPort(). You cannot specify null.	
10	<T> Dispatch<T>	createDispatch (javax.xml.namespace.QName portName, java.lang.Class<T> type, Service.Mode mode, WebServiceFeature... features)		N
11	Dispatch <java.lang.Object>	createDispatch (javax.xml.namespace.QName portName, javax.xml.bind.JAXBContext context, Service.Mode mode)		Y
		Note	In portName, you must specify the same port name as the name specified in the arguments of addPort(). You cannot specify null.	
12	Dispatch <java.lang.Object>	createDispatch (javax.xml.namespace.QName portName, javax.xml.bind.JAXBContext context, Service.Mode mode, WebServiceFeature... features)		N

No.	Return value type	Method name/ Explanation		Support
13	`java.util.concurrent.Executor`	`getExecutor()`		N
14	`HandlerResolver`	`getHandlerResolver()`		Y
		Explanation	Returns the `HandlerResolver` instance used by this `Service` instance. If the `HandlerResolver` instance does not exist, returns `null`.	
15	`<T> T`	`getPort (java.lang.Class<T> serviceEndpointInterface)`		Y
		Explanation	Returns port (proxy for accessing the service).	
		Argument	`serviceEndpointInterface`: This is the SEI `Class` class.	
		Exception	`javax.xml.ws.WebServiceException`: This exception occurs in the following cases: • When `serviceEndpointInterface` is `null` • When invoked from the `Service` instance generated with `Service.create()` where `null` is specified in `wsdlDocumentLocation` • When SEI that does not use the `javax.jws.WebService` annotation is specified in the argument • When the `HandlerResolver` object implementing `getHandlerChain()` that returns `null` is set up with `setHandlerResolver()` before this method is invoked • When the `HandlerResolver` object implementing `getHandlerChain()` that returns a handler chain containing a handler, which is neither a logical handler nor a SOAP handler, is set up using `setHandlerResolver()` before this method is invoked	
16	`<T> T`	`getPort (java.lang.Class<T> serviceEndpointInterface, WebServiceFeature... features)`		N
17	`<T> T`	`getPort(EndpointReference endpointReference, java.lang.Class<T> serviceEndpointInterface, WebServiceFeature... features)`		Y#
		Explanation	Returns port (proxy for accessing the service).	
		Argument	`endpointReference`: This is the service endpoint invoked by the port. `serviceEndpointInterface`: This is the SEI `Class` class. `features`: This is the list of `WebServiceFeature` configured on the port.	
		Exception	`javax.xml.ws.WebServiceException`: This exception occurs in the following cases: • When an exception occurs while the port is being generated • When the WSDL required for processing this method does not exist	

No.	Return value type	Method name/ Explanation		Support
17	<T> T	Exception	• When the endpointReference Meta data does not match serviceName of the Service instance • When portName cannot be extracted from the WSDL or endpointReference Meta data • When an invalid endpointReference is specified • When an invalid serviceEndpointInterface is specified • When there is no compatibility with the port, or when an unsupported WebServiceFeature is specified	Y#
18	<T> T	getPort(javax.xml.namespace.QName portName, java.lang.Class<T> serviceEndpointInterface)		Y
		Explanation	Returns port (proxy for accessing the service).	
		Argument	portName: This is the WSDL port name (name attribute value of the wsdl:port element). serviceEndpointInterface: This is the SEI Class class.	
		Exception	javax.xml.ws.WebServiceException: This exception occurs in the following cases: • When QName other than the WSDL port name (name attribute value of the wsdl:port element) is specified in portName • When portName is null • When serviceEndpointInterface is null	
19	<T> T	getPort(javax.xml.namespace.QName portName, java.lang.Class<T> serviceEndpointInterface, WebServiceFeature... features)		N
20	java.util.Iterator <javax.xml.namespace.QName>	getPorts()		Y
		Explanation	Returns the Iterator of the QName list indicating the service end point (port included in WSDL (wsdl:port element)).	
		Exception	javax.xml.ws.WebServiceException: This exception occurs when the Service.create() method is invoked for the Service instance generated by specifying null in wsdlDocumentLocation.	
21	javax.xml.namespace.QName	getServiceName()		Y
		Explanation	Returns this service name (WSDL service name (name attribute value of the wsdl:service element)).	
22	java.net.URL	getWSDLDocumentLocation()		Y
		Explanation	Returns the location of the WSDL file for this service. Returns null when the Service.create() method is invoked for the Service instance generated by specifying null in the argument wsdlDocumentLocation.	
23	void	setExecutor (java.util.concurrent.Executor executor)		N

No.	Return value type	Method name/ Explanation		Support
24	void	setHandlerResolver (HandlerResolver handlerResolver)		Y
		Explanation	Set the HandlerResolver instance of this Service instance.	

Legend:

--: Indicates that there is no return value type.

Y: Supported in the Cosminexus JAX-WS functionality.

N: Not supported in the Cosminexus JAX-WS functionality.

#

The following table describes the operations when variable length arguments are specified in the getPort method.

Table 19-8: Specification methods and operations for the variable length arguments

No.	Specification method	Specification example	Operation
1	Argument is omitted	getPort(epr, sei)	Operation is performed considering that no variable length argument is specified.
2	One argument is specified	getPort(epr, sei, new FooFeatures())	Operation is performed considering that one variable length argument is specified.
3	Two arguments are specified	getPort(epr, sei, new FooFeatures(), new BarFeatures())	Operation is performed considering that two variable length arguments are specified.
4	null is specified in the argument	getPort(epr, sei, null)	Operation is performed considering that no variable length argument is specified.
5	null is specified in the argument and cast in the WebServiceFeature type array	getPort(epr, sei, (WebServiceFeature) null)	Operation is performed considering that a variable length argument with null is specified, and the exception NullPointerException occurs.
6	null is specified in the argument and cast in the WebServiceFeature type array	getPort(epr, sei, (WebServiceFeature[]) null)	Operation is performed considering that no variable length argument is specified.

(5) javax.xml.ws.wsaddressing.W3CEndpointReference class

The following table describes the support range of the javax.xml.ws.wsaddressing.W3CEndpointReference class. For details, see the *JAX-WS 2.2 specifications*.

Table 19-9: Support range of the javax.xml.ws.wsaddressing.W3CEndpointReference class

No.	Return value type	Method name	Support
1	--	W3CEndpointReference()	Y
2	--	W3CEndpointReference (javax.xml.transform.Source source)	Y
3	void	writeTo(javax.xml.transform.Result result)	N

Legend:

--: Indicates that there is no return value type.

Y: Supported in the JAX-WS functionality of Application Server.

N: Not supported in the JAX-WS functionality of Application Server.

.

19.2.3 Service API

This subsection describes the support range of the service APIs.

(1) javax.xml.ws.Provider interface

The following table describes the support range of the `javax.xml.ws.Provider` interface. For details, see the *JAX-WS 2.2 specifications*.

Table 19-10: Support range of the javax.xml.ws.Provider interface

No.	Return value type	Method name	Support
1	`T`	`invoke(T request)`	Y

Legend:

Y: Supported in the Cosminexus JAX-WS functionality.

(2) javax.xml.ws.WebServiceContext interface

The following table describes the support range of the `javax.xml.ws.WebServiceContext` interface. For details, see the *JAX-WS 2.2 specifications*.

Table 19-11: Support range of the javax.xml.ws.WebServiceContext interface

No.	Return value type	Method name and Description		Support
1	`<T extends EndpointReference> T`	`getEndpointReference (java.lang.Class<T> clazz, org.w3c.dom.Element... referenceParameters)`		N
2	`EndpointReference`	`getEndpointReference (org.w3c.dom.Element... referenceParameters)`		N
3	`MessageContext`	`getMessageContext()`		Y
		Description	Acquires the message context of an in-processing request when this method is called.	
		Exception	java.lang.IllegalStateException: This exception is thrown if the request is unprocessed and the method is called.	
4	`java.security.Principal`	`getUserPrincipal()`		N
5	`boolean`	`isUserInRole(java.lang.String role)`		N

Legend:

Y: Supported in the JAX-WS functionality of Application Server.

N: Not supported in the JAX-WS functionality of Application Server.

(3) javax.xml.ws.wsaddressing.W3CEndpointReferenceBuilder class

The following table describes the support range of the `javax.xml.ws.wsaddressing.W3CEndpointReferenceBuilder` class. For details, see the *JAX-WS 2.2 specifications*.

Table 19-12: Support range of the javax.xml.ws.wsaddressing.W3CEndpointReferenceBuilder class

No.	Return value type	Method name	Support
1	--	`W3CEndpointReferenceBuilder()`	Y

No.	Return value type	Method name	Support
2	W3CEndpointReferenceBuilder	address(java.lang.String address)	Y
3	W3CEndpointReferenceBuilder	attribute(javax.xml.namespace.QName name, java.lang.String value)	Y
4	W3CEndpointReference	build()	Y
5	W3CEndpointReferenceBuilder	element(org.w3c.dom.Element element)	Y
6	W3CEndpointReferenceBuilder	endpointName (javax.xml.namespace.QName endpointName)	Y
7	W3CEndpointReferenceBuilder	interfaceName(javax.xml.namespace.QName interfaceName)	Y
8	W3CEndpointReferenceBuilder	metadata (org.w3c.dom.Element metadataElement)	Y
9	W3CEndpointReferenceBuilder	referenceParameter (org.w3c.dom.Element referenceParameter)	Y
10	W3CEndpointReferenceBuilder	serviceName (javax.xml.namespace.QName serviceName)	Y
11	W3CEndpointReferenceBuilder	wsdlDocumentLocation (java.lang.String wsdlDocumentLocation)	Y#

Legend:

--: Indicates that there is no return value type.

Y: Supported in the Cosminexus JAX-WS functionality.

#

Mapping of the argument `wsdlDocumentLocation` by using the catalog file is not supported.

19.2.4 Core API

This subsection describes the support range of the core APIs.

(1) com.sun.xml.ws.developer.StreamingAttachmentFeature class

The following table describes the support range of the `com.sun.xml.ws.developer.StreamingAttachmentFeature` class.

Table 19-13: Support range of the com.sun.xml.ws.developer.StreamingAttachmentFeature class

No.	Return value type	Method name		Support
1	--	StreamingAttachmentFeature()		N
2	--	StreamingAttachmentFeature(String dir,boolean parseEagerly,long memoryThreshold)		Y
		Explanation	Creates StreamingAttachmentFeature.	
		Argument	dir: This is the path of the output destination directory for the temporary file. If a non-existent directory, a directory without access permissions, or an existing file name is specified, a message is displayed, and the MIME body included in the SOAP message containing the received attachment is expanded in the memory (KDJW10026-W).	

No.	Return value type	Method name		Support
2	--	Argument	`parseEagerly`: Specifies whether the SOAP message containing the received attachment will be parsed in detail. If `true`, the SOAP message containing an attachment is parsed in detail. `memoryThreshold`: This is the threshold value (unit: byte) for determining whether the MIME body included in the SOAP message will be expanded in the memory when a SOAP message containing an attachment is received. Specify a value greater than 16 KB (16384 L) or `-1`. If any other value is specified, the operation is not guaranteed. If you specify `-1`, the MIME body is always expanded in the memory.	Y
3	`java.lang.String`	`getID()`		Y
		Explanation	Obtains a unique identifier for `StreamingAttachmentFeature`.	
4	`org.jvnet.mimepull.MIMEConfig`	`getConfig()`		Y
		Explanation	Obtains the `MIMEConfig` instance indicating the streaming settings. If you invoke this method once, you cannot change the streaming settings by using the following `setter` methods: • `setDir(String dir)` • `setParseEagerly(boolean parseEagerly)` • `setMemoryThreshold(int memoryThreshold)`	
5	`void`	`setDir(String dir)`		Y
		Explanation	Specifies the directory to be used to output the MIME body, which is included in the SOAP message containing the received attachment, as a temporary file, when streaming is enabled. If this method is invoked multiple times, the value specified last is valid. If you invoke the `getConfig()` method, you cannot reset the values using this method.	
		Argument	`dir`: This is the path of the output destination directory for the temporary file. If a non-existent directory, a directory without access permissions, or an existing file name is specified, a message is displayed, and the MIME body included in the SOAP message containing the received attachment is expanded in the memory (KDJW10026-W).	
6	`void`	`setParseEagerly(boolean parseEagerly)`		Y
		Explanation	Specifies whether the SOAP message containing the received attachment will be parsed in detail, when streaming is enabled. If this method is invoked multiple times, the value specified last is valid. If you invoke the `getConfig()` method, you cannot reset the values using this method.	
		Argument	`parseEagerly`: Specifies whether the SOAP message containing the received attachment will be parsed in detail. If `true`, the SOAP message containing an attachment is parsed in detail.	
7	`void`	`setMemoryThreshold(int memoryThreshold)`		Y
		Explanation	Specifies the threshold value for determining whether the MIME body included in the SOAP message will be expanded in the memory when a SOAP message containing an attachment is received, when streaming is enabled. If this method is invoked	

No.	Return value type	Method name		Support
7	`void`	Explanation	multiple times, the value specified last is valid. If you invoke the `getConfig()` method, you cannot reset the values using this method.	Y
		Argument	`memoryThreshold`: This is the threshold value (unit: byte) for determining whether the MIME body included in the SOAP message will be expanded in the memory when a SOAP message containing an attachment is received. Specify a value greater than 16 KB (16384 L) or `-1`. If any other value is specified, the operation is not guaranteed. If you specify `-1`, the MIME body is always expanded in the memory.	

Legend:

--: Indicates that there is no return value type.

Y: Supported in the Cosminexus JAX-WS functionality.

N: Not supported in the Cosminexus JAX-WS functionality.

(2) com.sun.xml.ws.developer.StreamingDataHandler class

The following table describes the support range of the `com.sun.xml.ws.developer.StreamingDataHandler` class.

Table 19-14: Support range of the com.sun.xml.ws.developer.StreamingDataHandler class

No.	Return value type	Method name		Support
1	--	`StreamingDataHandler(Object o, String s)`		N
2	--	`StreamingDataHandler(URL url)`		N
3	--	`StreamingDataHandler(DataSource dataSource)`		N
4	`java.io.InputStream`	`readOnce()`		Y
		Explanation	Obtains `java.io.InputStream` of this object.	
		Exception	`IOException`: This exception occurs when `InputStream` corresponding to this object cannot be obtained.	
5	`void`	`moveTo(File file)`		Y
		Explanation	Moves the attachment indicated by this object to the specified file. • If `null` is specified in the argument `file`, `java.io.IOException` occurs (KDJW10023-E). • If the file or directory path existing in the argument `file`, or a file path containing a non-existent parent directory is specified, `IOException` occurs (KDJW10027-E).	
		Argument	`file`: This is the output destination file path.	
		Exception	`IOException`: This exception occurs in the following cases: • `null` is specified in the file.	

No.	Return value type	Method name		Support
5	`void`	Exception	• An existing file or directory path, or a file path containing a non-existent parent directory is specified in the file. • An I/O error occurs.	Y
6	`void`	`close()`		Y
		Explanation	Releases the resources of the attachment indicated by this object.	
		Exception	`IOException`: This exception is thrown when an I/O error occurs.	

Legend:

--: Indicates that there is no return value type.

Y: Supported in the Cosminexus JAX-WS functionality.

N: Not supported in the Cosminexus JAX-WS functionality.

(3) org.jvnet.mimepull.MIMEConfig class

The following table describes the support range of the `org.jvnet.mimepull.MIMEConfig` class.

Table 19-15: Support range of the org.jvnet.mimepull.MIMEConfig class

No.	Return value type	Method name		Support
1	--	`MIMEConfig()`		N
2	`void`	`setParseEagerly(boolean parseEagerly)`		Y
		Explanation	Specifies whether the SOAP message containing the received attachment will be parsed in detail, when streaming is enabled. If this method is invoked multiple times, the value specified last is valid.	
		Argument	`parseEagerly`: Specifies whether the SOAP message containing the received attachment will be parsed in detail. If `true`, the SOAP message containing an attachment is parsed in detail.	
3	`void`	`setMemoryThreshold(long memoryThreshold)`		Y
		Explanation	Specifies the threshold value for determining whether the MIME body included in the SOAP message will be expanded in the memory when a SOAP message containing an attachment is received, when streaming is enabled. If this method is invoked multiple times, the value specified last is valid.	
		Argument	`memoryThreshold`: This is the threshold value (unit: byte) for determining whether the MIME body included in the SOAP message will be expanded in the memory when a SOAP message containing an attachment is received. Specify a value greater than 16 KB (16384 L) or `-1`. If any other value is specified, the operation is not guaranteed. If you specify `-1`, the MIME body is always expanded in the memory.	
4	`void`	`setDir(String dir)`		Y
		Explanation	Specifies the directory to be used to output the MIME body, which is included in the SOAP message containing the received attachment, as a temporary file, when streaming is enabled. If this method is invoked multiple times, the value other than `null` or null character ("") and the one specified first is valid.	

No.	Return value type	Method name		Support
4	`void`	Argument	`dir`: This is the path of the output destination directory for the temporary file.	Y
5	`void`	`validate()`		Y
		Explanation	Validates whether a temporary file can be created. If the temporary file cannot be created, specify settings to expand the attachment in the memory. If a non-existent directory, a directory without access permissions, or an existing file name is specified using the `setDir(String dir)` method and not validated using this method, `org.jvnet.mimepull.MIMEParsingException` occurs when a temporary file is output by streaming.	

Legend:

--: Indicates that there is no return value type.

Y: Supported in the Cosminexus JAX-WS functionality.

N: Not supported in the Cosminexus JAX-WS functionality.

(4) javax.xml.ws.Binding interface

The following table describes the support range of the `javax.xml.ws.Binding` interface. For details, see the *JAX-WS 2.2 specifications*.

Table 19-16: Support range of the javax.xml.ws.Binding interface

No.	Return value type	Method name and Description		Support
1	`java.lang.String`	`getBindingID()`		N
2	`java.util.List<Handler>`	`getHandlerChain()`		Y
		Description	Acquires a copy of the handler chain of the protocol binding instance.	
3	`void`	`setHandlerChain (java.util.List<Handler> chain)`		Y
		Description	Specifies the handler chain of the protocol binding instance.	
		Argument	chain: List of handlers that configure the handler chain.	
		Exception	`javax.xml.ws.WebServiceException`: The exception is thrown if an error occurs when setting the handler chain or when `null` is specified in chain.	

Legend:

Y: Supported in the JAX-WS functionality of Application Server.

N: Not supported in the JAX-WS functionality of Application Server

(5) javax.xml.ws.handler.Handler<C extends MessageContext> interface

The following table describes the support range of the `javax.xml.ws.handler.Handler<C extends MessageContext>` interface. For details, see the *JAX-WS 2.2 specifications*.

Table 19-17: Support range of the javax.xml.ws.handler.Handler<C extends MessageContext> interface

No.	Return value type	Method name/ Explanation		Support
1	`MessageContext.Scope`	`close(MessageContext context)`		Y
		Explanation	This method is invoked just before the JAX-WS runtime dispatches a message, fault, or exception, when the message exchange is completed.	
2	`boolean`	`handleFault(C context)`		Y
		Explanation	This method is invoked to process a fault message.	
3	`boolean`	`handleMessage(C context)`		Y
		Explanation	This method is invoked to perform normal processing for the inbound and outbound messages.	

Legend:

Y: Supported in the Cosminexus JAX-WS functionality.

(6) javax.xml.ws.handler.HandlerResolver interface

The following table describes the support range of the `javax.xml.ws.handler.HandlerResolver` interface. For details, see the *JAX-WS 2.2 specifications*.

Table 19-18: Support range of the javax.xml.ws.handler.HandlerResolver interface

No.	Return value type	Method name/ Explanation		Support
1	`java.util.List<Handler>`	`getHandlerChain(PortInfo portInfo)`		Y
		Explanation	Obtains the handler chain of the specified port. This method must not return `null`.	
		Argument	`portInfo`: This is the information of the port you want to access.	

Legend:

Y: Supported in the Cosminexus JAX-WS functionality.

(7) javax.xml.ws.handler.LogicalMessageContext interface

The following table describes the support range of the `javax.xml.ws.handler.LogicalMessageContext` interface. For details, see the *JAX-WS 2.2 specifications*.

Table 19-19: Support range of the javax.xml.ws.handler.LogicalMessageContext interface

No.	Return value type	Method name/ Explanation		Support
1	`LogicalMessage`	`getMessage()`		Y
		Explanation	Obtains the messages from this message context.	

Legend:

Y: Supported in the Cosminexus JAX-WS functionality.

(8) javax.xml.ws.handler.MessageContext interface

The following table describes the support range of the `javax.xml.ws.handler.MessageContext` interface. For details, see the *JAX-WS 2.2 specifications*.

Table 19-20: Support range of the javax.xml.ws.handler.MessageContext interface

No.	Return value type	Method name/ Explanation		Support
1	`MessageContext.Scope`	`getScope(java.lang.String name)`		Y#1
		Explanation	Obtains the scope of the property.	
		Exception	`java.lang.IllegalArgumentException`: This exception occurs when `null`, an empty string, and a property name not associated with this message context is specified in `name`. The standard message context properties and user-defined message context properties are assumed to be associated with the message context. For details on the message context properties, see *19.2.5(1) Support range of the message context properties*.	
2	`void`	`setScope(java.lang.String name, MessageContext.Scope scope)`		Y#2
		Description	Specifies scope of the property.	
		Exception	`java.lang.IllegalArgumentException` This exception is thrown if `null`, a blank character string, or a property name not associated with the message context is specified in the name.The standard message context properties and user-defined message context properties are regarded as the property names associated with the message context. For details on the message context properties, see *19.2.5(1) Support range of the message context properties*.	

Legend:

Y: Supported in the Cosminexus JAX-WS functionality.

#1

Supported only when used in a handler.

#2

This is a service side handler for inbounding and supports the addition of the user-defined message context property only. For details on the notes when adding the user-defined message context property, see *10.21.2(2) Notes when adding a user-defined message context property*.

(9) javax.xml.ws.handler.PortInfo interface

The following table describes the support range of the `javax.xml.ws.handler.PortInfo` interface. For details, see the *JAX-WS 2.2 specifications*.

Table 19-21: Support range of the javax.xml.ws.handler.PortInfo interface

No.	Return value type	Method name/ Explanation		Support
1	`java.lang.String`	`getBindingID()`		Y
		Explanation	Obtains the URI that identifies the binding, used by the port to be accessed.	
2	`javax.xml.namespace.QName`	`getPortName()`		Y
		Explanation	Obtains `QName` from the WSDL port to be accessed.	
3	`javax.xml.namespace.QName`	`getServiceName()`		Y
		Explanation	Obtains `QName` of the WSDL service that includes the port to be accessed.	

Legend:
Y: Supported in the Cosminexus JAX-WS functionality.

(10) javax.xml.ws.handler.soap.SOAPHandler<T extends SOAPMessageContext> interface

The following table describes the support range of the `javax.xml.ws.handler.soap.SOAPHandler<T extends SOAPMessageContext>` interface. For details, see the *JAX-WS 2.2 specifications*.

Table 19-22: Support range of the javax.xml.ws.handler.soap.SOAPHandler<T extends SOAPMessageContext> interface

No.	Return value type	Method name/ Explanation		Support
1	`java.util.Set<javax.xml.namespace.QName>`	`getHeaders()`		Y
		Explanation	Obtains the headers that can be processed with this handler instance.	

Legend:
Y: Supported in the Cosminexus JAX-WS functionality.

(11) javax.xml.ws.handler.soap.SOAPMessageContext interface

The following table describes the support range of the `javax.xml.ws.handler.soap.SOAPMessageContext` interface. For details, see the *JAX-WS 2.2 specifications*.

Table 19-23: Support range of the javax.xml.ws.handler.soap.SOAPMessageContext interface

No.	Return value type	Method name/ Explanation		Support
1	`java.lang.Object[]`	`getHeaders(javax.xml.namespace.QName header, javax.xml.bind.JAXBContext context, boolean allRoles)`		Y
		Explanation	Obtains a header with a specific `QName` from the messages of this message context. If this message context does not have messages or if a header that matches with `QName` specified in `header` does not exist, this method returns an empty array.	
		Exception	`javax.xml.ws.WebServiceException`: This exception occurs in the following cases: • When `null` is specified in the `header` • When `null` is specified in the `context`	
2	`javax.xml.soap.SOAPMessage`	`getMessage()`		Y
		Explanation	Obtains the SOAP message from this message context.	
3	`java.util.Set<java.lang.String>`	`getRoles()`		Y
		Explanation	Obtains the SOAP actor and role associated with the execution of the handler chain.	
4	`void`	`setMessage(javax.xml.soap.SOAPMessage message)`		N

Legend:
Y: Supported in the Cosminexus JAX-WS functionality.
N: Not supported in the Cosminexus JAX-WS functionality.

(12) javax.xml.ws.Holder<T> class

The following table describes the support range of the `javax.xml.ws.Holder<T>` class. For details, see the *JAX-WS 2.2 specifications*.

Table 19-24: Support range of the javax.xml.ws.Holder<T> class

No.	Return value type	Method name/ Explanation		Support
1	--	`Holder()`		Y
		Explanation	Creates a new holder containing a null value.	
2	--	`Holder(T value)`		Y
		Explanation	Creates a new holder containing the specified value.	

Legend:

--: Indicates that there is no return value type.

Y: Supported in the Cosminexus JAX-WS functionality.

(13) javax.xml.ws.LogicalMessage interface

The following table describes the support range of the `javax.xml.ws.LogicalMessage` interface.

Table 19-25: Support range of the javax.xml.ws.LogicalMessage interface

No.	Return value type	Method name/ Explanation		Support
1	`javax.xml.transform.Source`	`getPayload()`		Y
		Explanation	Obtains the message payload as an XML source.	
2	`java.lang.Object`	`getPayload(javax.xml.bind.JAXBContext context)`		Y
		Explanation	Obtains the message payload as a JAXB object.	
3	`void`	`setPayload(java.lang.Object payload, javax.xml.bind.JAXBContext context)`		N
4	`void`	`setPayload(javax.xml.transform.Source payload)`		N

Legend:

Y: Supported in the Cosminexus JAX-WS functionality.

N: Not supported in the Cosminexus JAX-WS functionality.

(14) javax.xml.ws.ProtocolException class

The following table describes the support range of the `javax.xml.ws.ProtocolException` class. For details, see the *JAX-WS 2.2 specifications*.

Table 19-26: Support range of the javax.xml.ws.ProtocolException class

No.	Return value type	Method name/ Explanation		Support
1	--	`ProtocolException()`		Y
		Explanation	Sets up a new exception where the detailed message is `null`.	
2	--	`ProtocolException(java.lang.String message)`		Y
		Explanation	Sets up a new exception containing the specified detailed message.	
3	--	`ProtocolException(java.lang.String message, java.lang.Throwable cause)`		Y
		Explanation	Sets up a new exception using the specified detailed message and cause.	
4	--	`ProtocolException(java.lang.Throwable cause)`		Y

No.	Return value type	Method name/ Explanation		Support
4	--	Explanation	Sets up a new exception containing the specified cause and detailed message.	Y

Legend:

--: Indicates that there is no return value type.

Y: Supported in the Cosminexus JAX-WS functionality.

(15) javax.xml.ws.soap.AddressingFeature class

The following table describes the support range of the `javax.xml.ws.soap.AddressingFeature` class. For details, see the *JAX-WS 2.2 specifications*.

Table 19-27: Support range of the javax.xml.ws.soap.AddressingFeature class

No.	Return value type	Method name		Support
1	--	`AddressingFeature()`		Y
		Explanation	Creates `AddressingFeature`.	
2	--	`AddressingFeature(boolean enabled)`		Y
		Explanation	Creates `AddressingFeature`.	
		Argument	`enabled`: Specifies whether to enable or disable WS-Addressing.	
3	--	`AddressingFeature(boolean enabled, boolean required)`		Y
		Explanation	Creates `AddressingFeature`.	
		Argument	`enabled`: Specifies whether to enable or disable WS-Addressing. `required`: Specify this argument when you want to use WS-Addressing.	
4	--	`AddressingFeature(boolean enabled, boolean required, AddressingFeature.Responses responses)`		Y
		Explanation	Creates `AddressingFeature`.	

		Argument	`enabled`:
			Specifies whether to enable or disable WS-Addressing.
			`required`:
			Specify this argument to request the use of WS-Addressing.
			`responses`:
			Specifies the type of response endpoint to be requested. You can specify one of the following response endpoint types: #
			• All URIs
			`javax.xml.ws.soap.AddressingFeature.Responses.ALL`
			• Only anonymous URIs
			`javax.xml.ws.soap.AddressingFeature.Responses.ANONYMOUS`
			• Only non-anonymous URIs
			`javax.xml.ws.soap.AddressingFeature.Responses.NON_ANONYMOUS`	

No.	Return value type	Method name		Support
5	`java.lang.String`	`getID()`		Y
		Explanation	Obtains a unique identifier for `AddressingFeature`.	
6	`AddressingFeature.Responses`	`getResponses()`		Y
		Explanation	Obtains the type of response endpoint to be requested.	
7	`Boolean`	`isRequired()`		Y
		Explanation	Obtains information about whether the use of `AddressingFeature` will be requested.	

Legend:

--: Indicates that there is no return value type.

Y: Supported in the Cosminexus JAX-WS functionality.

#

The specification of responses does not affect the operations of the client-side JAX-WS engine.

(16) javax.xml.ws.soap.MTOMFeature class

The following table describes the support range of the `javax.xml.ws.soap.MTOMFeature` class. For details, see the *JAX-WS 2.2 specifications*.

Table 19-28: Support range of the javax.xml.ws.soap.MTOMFeature class

No.	Return value type	Method name		Support
1	--	`MTOMFeature()`		Y
		Explanation	Creates `MTOMFeature`.	
2	--	`MTOMFeature(boolean enabled)`		Y
		Explanation	Creates `MTOMFeature`.	
		Argument	`enabled`: Specifies whether to use attachments in the MTOM/XOP specification format.	
3	--	`MTOMFeature(boolean enabled, int threshold)`		Y
		Explanation	Creates `MTOMFeature`.	
		Argument	`enabled`: Specifies whether to use attachments in the MTOM/XOP specification format. `threshold`: This is the size (unit: byte) of the binary data to be sent as an attachment in the MTOM/XOP specification format. If the value specified for the threshold <= binary data size, the binary data is sent as an attachment in the MTOM/XOP specification format.	
4	--	`MTOMFeature(int threshold)`		Y
		Explanation	Creates `MTOMFeature`.	
		Argument	`threshold`: This is the size (unit: byte) of the binary data to be sent as an attachment in the MTOM/XOP specification format. If the value specified for the threshold <= binary data size, the binary data is sent as an attachment in the MTOM/XOP specification format.	

No.	Return value type	Method name		Support
5	`java.lang.String`	`getID()`		Y
		Explanation	Obtains the unique identifier of `MTOMFeature`.	
6	`int`	`getThreshold()`		Y
		Explanation	Obtains the threshold value to determine whether to send the binary data as an attachment in the MTOM/XOP specification format.	

Legend:

--: Indicates that there is no return value type.

Y: Supported in the Cosminexus JAX-WS functionality.

(17) javax.xml.ws.soap.SOAPBinding interface

The following table describes the support range of the `javax.xml.ws.soap.SOAPBinding` interface. For details, see the *JAX-WS 2.2 specifications*.

Table 19-29: Support range of the javax.xml.ws.soap.SOAPBinding interface

No.	Return value type	Method name		Support
1	`javax.xml.soap.MessageFactory`	`getMessageFactory()`		N
2	`java.util.Set<java.lang.String>`	`getRoles()`		N
3	`javax.xml.soap.SOAPFactory`	`getSOAPFactory()`		N
4	`boolean`	`isMTOMEnabled()`		Y
		Explanation	Returns `true` if attachments in the MTOM/XOP specification format are enabled.	
5	`void`	`setMTOMEnabled(boolean flag)`		Y
		Explanation	Enables or disables attachments in the MTOM/XOP specification format. If the enabling or disabling of attachments in the MTOM/XOP specification format is not set up by using `MTOMFeature` or this method, you can specify the settings with this method. If the enabling or disabling of attachments in the MTOM/XOP specification format is already set up, you cannot respecify the settings using this method.	
		Argument	`flag`: Specifies whether to enable or disable attachments in the MTOM/XOP specification format.	
6	`void`	`setRoles(java.util.Set<java.lang.String> roles)`		N

Legend:

Y: Supported in the Cosminexus JAX-WS functionality.

N: Not supported in the Cosminexus JAX-WS functionality.

(18) javax.xml.ws.soap.SOAPFaultException class

The following table describes the support range of the `javax.xml.ws.soap.SOAPFaultException` class. For details, see the *JAX-WS 2.2 specifications*.

Table 19-30: Support range of the javax.xml.ws.soap.SOAPFaultException class

No.	Return value type	Method name/ Explanation		Support
1	--	`SOAPFaultException (javax.xml.soap.SOAPFault fault)`		Y
		Explanation	Sets up a new exception of `SOAPFaultException`.	
2	`javax.xml.soap.SOAPFault`	`getFault()`		Y
		Explanation	Obtains the embedded `SOAPFault` instance.	

Legend:

--: Indicates that there is no return value type.

Y: Supported in the Cosminexus JAX-WS functionality.

(19) javax.xml.ws.WebServiceException class

The following table describes the support range of the `javax.xml.ws.WebServiceException` class. For details, see the *JAX-WS 2.2 specifications*.

Table 19-31: Support range of the javax.xml.ws.WebServiceException class

No.	Return value type	Method name/ Explanation		Support
1	--	`WebServiceException()`		Y
		Explanation	Sets up a new exception where the detailed message is `null`.	
2	--	`WebServiceException (java.lang.String message)`		Y
		Explanation	Sets up a new exception containing the specified detailed message.	
3	--	`WebServiceException(java.lang.String message, java.lang.Throwable cause)`		Y
		Explanation	Sets up a new exception containing the specified detailed message and cause.	
4	--	`WebServiceException (java.lang.Throwable cause)`		Y
		Explanation	Sets up a new exception containing the specified cause and detailed message.	

Legend:

--: Indicates that there is no return value type.

Y: Supported in the Cosminexus JAX-WSfunctionality.

19.2.5 Using a message context

Within the support range of the JAX-WS APIs, you can access the message context from the handler, Web Services client, and Web Service.The details are as follows.

In the case of a handler

The message context is passed by the parameters of the called back method (such as the `handleMessage` method).

In the case of a Web Services client

You can access a copy of the message context using the `getRequestContext` method of the `javax.xml.ws.BindingProvider` interface and the `getResponseContext` method.

For Web Services

You can access the message context by using the `getMessageContext` method of the `javax.xml.ws.WebServiceContext` interface.

For details on the JAX-WS APIs, see the *JAX-WS 2.2 specifications*. Also, for details on the support range of the JAX-WS APIs in the Cosminexus JAX-WS functionality, see *19.2.1 List of interfaces and classes (JAX-WS)*. The Cosminexus JAX-WS functionality supports the standard properties defined in the JAX-WS 2.2 specifications and the vendor-specific properties.

The standard properties mentioned in Chapter 9 of the JAX-WS 2.2 specifications can only be referenced. The operations are not supported if the standard properties are changed.

(1) Support range of the message context properties

The following table describes the support range of the message context properties.

Table 19–32: List of message context properties

No.	Property name	Description location #1	Mandatory #2	Support						
				Web Service client		Web Services #5	Handler			
							Web Service client		Web Service	
				Out#3	In#4		Out#6	In#7	In#8	Out#9
javax.xml.ws.handler.message										
1	.outbound#10	9.4.1.1	Y	N#10	N#10	N	R	R	R	R
javax.xml.ws.binding.attachments										
2	.inbound#10	9.4.1.1	Y	N#10	N#10	R	R	R	R	R
3	.outbound#10	9.4.1.1	Y	N#10	N#10	N	R	R	R	R
javax.xml.ws.reference										
4	.parameters#10	9.4.1.1	Y	N#10	N#10	R	R	R	R	R
javax.xml.ws.wsdl										
5	.description#10, #12	9.4.1.1	--	N#10	N#10	N#11	N#11	N#11	N#11	N#11
6	.service#10, #12	9.4.1.1	--	N#10	N#10	R	R	R	R	R
7	.port#10, #12	9.4.1.1	--	N#10	N#10	R	R	R	R	R
8	.interface#10, #12	9.4.1.1	--	N#10	N#10	R	R	R	R	R
9	.operation#10, #12, #13	9.4.1.1	--	N#10	R#13	R	R	R	R	R
javax.xml.ws.http.request										
10	.headers	9.4.1.1	Y	R/C#14	N#10	R	R/C#14	R#15	R	R
11	.method#10, #11	9.4.1.1	Y	N#10	N#10	R	R#15	R#15	R	R
12	.querystring#10, #11	9.4.1.1	Y	N#10	N#10	R	R#15	R#15	R	R
13	.headers	9.4.1.1	Y	N#10	N#10	R#16	R#15	R#15	R#16	R#16
javax.xml.ws.http.response										
14	.headers#10	9.4.1.1	Y	N#10	N#10	N	R#17	R	R#17	R#17
15	.code#10	9.4.1.1	Y	N#10	N#10	N	R#18	R	R	R
javax.xml.ws.servlet										
16	.context#10, #11	9.4.1.1	Y	N#10	N#10	R	R#15	R#15	R	R

No.	Property name	Description location #1	Mandatory #2	Support						
				Web Service client		Web Services #5	Handler			
							Web Service client		Web Service	
				Out#3	In#4		Out#6	In#7	In#8	Out#9
17	.request#10	9.4.1.1	Y	N#10	N#10	R	R#15	R#15	R	R
18	.response#10	9.4.1.1	Y	N#10	N#10	R	R#15	R#15	R	R
javax.xml.ws.service.endpoint										
19	.address#19	4.2.1.1	Y	R/C	R	N	R/C	R	R	R
javax.xml.ws.security.auth										
20	.username	4.2.1.1	Y	R/C	R	N	R/C	R	R	R
21	.password	4.2.1.1	Y	R/C	R	N	R/C	R	R	R
javax.xml.ws.session										
22	.maintain	4.2.1.1	Y	R/C	R	N	R/C	R	R	R
javax.xml.ws.soap.http.soapaction										
23	.use	4.2.1.1	--	N#20	N#20	N#20	N#20	N#20	N#20	N#20
24	.uri	4.2.1.1	--	N#20	N#20	N#20	N#20	N#20	N#20	N#20
com.cosminexus.jaxws										
25	.connect.timeout			R/C	R	N	R/C	R	R	R
26	.request.timeout			R/C	R	N	R/C	R	R	R
com.cosminexus.xml.ws.client.http										
27	.HostnameVerification Property			R/C#21	N	N	N	N	N	N

Legend:

Y: Indicates that the property is mandatory.

--: Indicates that the property is not mandatory.

R/C: Can be referenced and changed.

R: Can only be referenced. The operations are not guaranteed if the property is changed.

N: Cannot be referenced and changed.

Blank column: Indicates not applicable because this property is provided by the Cosminexus JAX-WS functionality.

#1

Indicates the locations defined in the JAX-WS 2.2 specifications.

#2

Indicates whether the property is mandatory in the JAX-WS 2.2 specifications.

#3

Indicates whether the property can be referenced or changed in the request context that can be obtained with `javax.xml.ws.BindingProvider#getRequestContext`.

#4

Indicates whether the property can be referenced or changed in the request context that can be obtained with `javax.xml.ws.BindingProvider#getResponseContext`.

#5

For details on injection of the Web Services context, see *10.21.2 Injecting aWeb Services context*, and for details on the message context property in Web Services, see *19.2.5(2)(l) Message context property on Web Services*.

#6

Indicates that the handler is associated with the Web Service client and whether the property can be referenced or changed for the outbound processing (when the request message is sent).

#7

Indicates that the handler is associated with the Web Service client and whether the property can be referenced or changed for the inbound processing (when the response message is received).

#8

Indicates that the handler is associated with the Web Service implementation class or provider implementation class and whether the property can be referenced or changed for the inbound processing (when the request message is received). For details on notes when adding the message context property, see *10.21.2(2) Notes when adding a user-defined message context property*.

#9

Indicates that the handler is associated with the Web Service implementation class or provider implementation class and whether the property can be referenced or changed for the outbound processing (when the response message is sent). For details on notes when adding the message context property, see *10.21.2(2) Notes when adding a user-defined message context property*.

#10

See *19.2.5(2)(e) Message context properties with HANDLER scope in the Web Service client*.

#11

Always returns `null`.

#12

See *19.2.5(2)(h) Message context properties related to the WSDL*.

#13

See *19.2.5(2)(i) Message context properties related to the WSDL operation name*.

#14

You can only add and reference the HTTP header `Accept-Encoding` that is used for linking with the HTTP response compression functionality, and the HTTP header `Content-Encoding` that is used for gzip compression of the HTTP request body. For details on `Accept-Encoding`, see *10.18 Linking with the HTTP response compression functionality*. For details on `Content-Encoding`, see *10.17 gzip compression of the HTTP request body*.

#15

See *19.2.5(2)(a) Message context properties that are irrelevant even when operated with the handler in the Web Services client*.

#16

See *19.2.5(2)(b) Path information*.

#17

See *19.2.5(2)(c) HTTP header*.

#18

See *19.2.5(2)(d) HTTP status code*.

#19

See *19.2.5(2)(g) Message context properties specified in the service endpoint address*.

#20

See *19.2.5(2)(f) Message context properties related to the SOAPAction header*.

#21

See *19.2.5(2)(k) How to set up the com.cosminexus.xml.ws.client.http.HostnameVerificationProperty property*.

(2) Notes on using a message context

This point describes the notes on using a message context.

(a) Message context properties that are irrelevant even when operated with the handler in the Web Service client

The property that stores the HTTP method map for the request message (such as the `javax.xml.ws.http.request.method` property) is obtained using the handler in the Web Service and is a relevant property. Therefore, when this property is referenced with the handler in the Web Service client, `null` is always returned.

(b) Path information

`null` is always stored for the `javax.xml.ws.http.request.pathinfo` property.

(c) HTTP header

- The `javax.xml.ws.http.response.headers` property that stores the HTTP header map for the response message is obtained using the handler in the Web Service client for inbound processing and is a relevant property. Therefore, when this property is referenced with the handler in the Web Service or the outbound handler in the Web Service client, `null` is always returned.
- When referencing the `javax.xml.ws.http.request.headers` property and `javax.xml.ws.http.response.headers` property of the message context by using a handler, the HTTP header name that is a key value of the obtained map `(Map<String,List<String>>object)` always has the first letter in capital regardless of the actual HTTP message sent and received (HTTP message that includes the SOAP message).

  Example: `Content-type`

(d) HTTP status code

The outbound handler in the Web Service client is processed before the HTTP communication is performed. Therefore, if the `javax.xml.ws.http.response.code` property storing the HTTP status code is referenced from the handler, `null` is always returned.

(e) Message context properties with HANDLER scope in the Web Service client

The standard message context properties include an APPLICATION scope and HANDLER scope, but only the message context properties with the APPLICATION scope can be referenced from the Web Service client. Therefore, with the Cosminexus JAX-WS functionality, the properties assigned #9 in the table in *19.2.5(1) Support range of the message context properties* cannot be used with the Web Services client. The operations are not guaranteed if these properties are referenced.

(f) Message context properties related to the SOAPAction header

The Cosminexus JAX-WS functionality does not support the `SOAPAction` header, so the `javax.xml.ws.soap.http.soapaction.use` and `javax.xml.ws.soap.http.soapaction.uri` properties cannot be used. The operations are not guaranteed if these properties are referenced.

(g) Message context properties specified in the service endpoint address

You cannot set up spaces and null in the `javax.xml.ws.service.endpoint.address` property that specifies the service endpoint address. The operations are not guaranteed if a space or null is set up. For details on the other values specified in the `javax.xml.ws.service.endpoint.address` property, see *20.2(3) Values specifiable in the location attribute of the soap: address element or soap12: address element*.

(h) Message context properties related to the WSDL

The dispatch-based Web Service client and provider-based Web Service do not contain the WSDL file, so `null` is always returned when the message context properties related to the WSDL are referenced.

(i) Message context properties related to the WSDL operation name

The `javax.xml.ws.wsdl.operation` property in the stub-based Web Service client can only be referenced with the request context that can be obtained with `javax.xml.ws.BindingProvider#getResponseContext`. If the property is referenced with the request context that can be obtained with `javax.xml.ws.BindingProvider#getRequestContext`, `null` is always returned. Also, even if a value is set in the `javax.xml.ws.wsdl.operation` property, the value does not affect the SOAP message to be sent.

(j) Specifying the service endpoint address for using the WS-RM 1.2 functionality

With the Web Service client that uses the WS-RM 1.2 functionality, specify the service endpoint address before you invoke the first Web Service. If the service endpoint is changed subsequent to the first communication, the WS-RM communication fails.

(k) How to set up the com.cosminexus.xml.ws.client.http.HostnameVerificationProperty property

Specify `true` or `false` in the message context for the `com.cosminexus.xml.ws.client.http.HostnameVerificationProperty` property using a `java.lang.String` type string, as shown in the following figure. The operations are not guaranteed if a string other than a `java.lang.String` type string is used to set up the value.

```
context.put("com.cosminexus.xml.ws.client.http.HostnameVerificationProperty", "true");
```

(l) Message context property in Web Services

The description of referencing and changing the message context property in Web Services is as follows:

- You can reference some message context properties from among the standard message context properties defined in the section *9.4.1.1* of the *HANDLER-scoped JAX-WS 2.2 specifications*. For details on referencable message context properties, see *Table 19-32 List of the message context properties* of the *19.2.5(1) Support range of the message context properties*.
- You cannot reference or change the CJW specific message context properties and standard message context properties defined in section *4.2.1.1* of the *APPLICATION-scoped JAX-WS 2.2 specifications*.
- You can reference and change the APPLICATION-scoped user-defined message context properties.

19.3 Support range of annotations

Specify the `javax.xml.ws.WebServiceRef` annotation when injecting ports and service classes.The operation is not gurenteed if you specify any other annotation.You cannot concurrently specify the `javax.xml.ws.WebServiceRef` annotation in the `setter` method compliant to the fields.

You can specify the `javax.xml.ws.WebServiceRef` annotation in the fields and the `setter` method of the Web Services clients executed on the J2EE server. If you specify any other annotation, the annotation is ignored.

19.3.1 javax.xml.ws.WebServiceRef annotation

The following table lists the support range and elements of the `javax.xml.ws.WebServiceRef` annotation.

Table 19-33: List of elements of the annotations of the JAX-WS

No.	Annotation		Support
	Annotation name	Element name	
1	`javax.xml.ws.WebServiceRef`	`lookup`	N
2		`mappedName`	N
3		`name`	N
4		`type`	N
5		`value`	Y
6		`wsdlLocation`	Y

Legend:

Y: Indicates the annotations and elements that you can specify.

N: Indicates the annotations and elements that you cannot specify (not supported).

The support range for each element is as follows:

(1) value elements(javax.xml.ws.WebServiceRef)

The `value` element specifies the service class that inherits `javax.xml.ws.Service`.

If the `javax.xml.ws.WebServiceRef` annotation is specified for methods or fields of the service class type, you cannot specify the `value` element. If you specify the `value` element, the operation is not gurenteed.

You must specify the `value` element when specifying the `javax.xml.ws.WebServiceRef` annotation for the fields or methods of the port type. If you do not specify the `value` element, the operation is not guerenteed.

(2) wsdlLocation element (javax.xml.ws.WebServiceRef)

The `wsdlLocation` element specifies the location of aWSDL document of Web Services.

You can specify the element in the following formats:

- URL
  Specify a URL of a WSDL document. The specification format is according to the `java.net.URL class` specifications. You can specify only http and https protocols.You cannot specify other protocols.
- Relative path
  Specify the WSDL document in the WAR file or EJB JAR file in the form of a relative path as described below. Note that the path must be a character string conforming to the rules of the RFC 2396 specifications.
  - When storing a Web Services client where the `javax.xml.ws.WebServiceRef` annotation is specified, in the WAR file, specify a relative path starting with WEB-INF. Store the WSDL document under the path that is specified in the WAR file.

    - When storing a Web Services client where the javax.xml.ws.WebServiceRef annotation is specified, in the EJB JAR file, specify a relative path starting with META-INF. Store the EJB JARfile under the path that is specified in the WAR file.
- Absolute path

  Specify the absolute path of aWSDL document. The path must be in the form of a system dependant absolute path .The specification format is according to the `java.io.File` class.

Notes when specifying the `wsdlLocation` element are as follows:

- When creating an instance of the Web Services client, create a service class by using a value specified in the `wsdlLocation` element and inject service classes and ports. To do so, you must specify a location of a valid WSDL document in the `wsdlLocation` element when creating an instance. If you specify a non-existing WSDL location or the location of WSDL containing incorrect contents, service classes and ports are not injected, and, the value of fields and the return value of the `setter` method remain `null`.
- If you do not specify the `wsdlLocation` element in the similar way as the default constructor of the service class, create an instance of a port or a service class by using the value of the `wsdlLocation` element of the `javax.xml.ws.WebServiceClient` annotation specified in a service class.

19.4 Support range of the handler chain configuration file

This section describes the support range of the syntax in the handler chain configuration file for each of the following elements:

- `javaee:handler-chains` element
- `javaee:handler-chain` element
- `javaee:handler` element
- `javaee:handler-name` element
- `javaee:handler-class` element
- `javaee:soap-header` element
- `javaee:soap-role` element

The support range of the contents not described in this section conforms to the "Java EE Web Services Metadata Handler Chain Schema" (standard schema) in the Java EE 5 specifications.

19.4.1 javaee:handler-chains element

This subsection describes the support range of the `javaee:handler-chains` element.

- You can only code one `javaee:handler-chains` element as the root element of the handler chain configuration file. You cannot omit this element.
- You can specify the `javaee:handler-chain` element as the child element. If you code a child element other than the `javaee:handler-chain` element, an error message (KDJW30019-E) is output to the standard error output and the log.
- You can only specify the standard attributes of the XML specifications and XML Schema specifications (such as the `xmlns` attribute). If you specify attributes other than the standard attributes of the XML specifications and XML Schema specifications, an error message (KDJW30019-E) is output to the standard error output and the log.

19.4.2 javaee:handler-chain element

This subsection describes the support range of the `javaee:handler-chain` element.

- You can only code one `javaee:handler-chain` element as the child element of the `javaee:handler-chains` element. You cannot omit this element. If omitted, an error message (KDJW30019-E) is output to the standard error output and the log.
- You can specify the `javaee:handler` element as the child element. If you code a child element other than the `javaee:handler` element, an error message (KDJW30019-E) is output to the standard error output and the log.
- You can only specify the standard attributes of the XML specifications and XML Schema specifications (such as the `xmlns` attribute). If you specify attributes other than the standard attributes of the XML specifications and XML Schema specifications, an error message (KDJW30019-E) is output to the standard error output and the log.

19.4.3 javaee:handler element

This subsection describes the support range of the `javaee:handler` element.

- You can code 1 to 64 `javaee:handler` elements as the child elements of the `javaee:handler-chain` element. If you omit this element or code 65 or more elements, an error message (KDJW30019-E) is output to the standard error output and the log.

- You can specify the following elements as the child elements. If you code elements other than the following elements, an error message (KDJW30019-E) is output to the standard error output and the log:
  - `javaee:handler-name` element
  - `javaee:handler-class` element
  - `javaee:soap-header` element
  - `javaee:soap-role` element
- Specify the `javaee:handler-name`, `javaee:handler-class`, `javaee:soap-header`, and `javaee:soap-role` elements in the above order.
- You can only specify the standard attributes of the XML specifications and XML Schema specifications (such as the `xmlns` attribute). If you specify attributes other than the standard attributes of the XML specifications and XML Schema specifications, an error message (KDJW30019-E) is output to the standard error output and the log.

19.4.4 javaee:handler-name element

This subsection describes the support range of the `javaee:handler-name` element.

- You can code 0 or 1 `javaee:handler-name` element as the child element of the `javaee:handler` element. If you code 2 or more `javaee:handler-name` elements, an error message (KDJW30019-E) is output to the standard error output and the log[#].
- You code the handler name in the value. However, this value is ignored with the JAX-WS engine, so you can code a value in the range that does not violate the restrictions of the `xsd:token` type in the XML Schema specifications[#].
- You cannot code child elements. If you code a child element, an error message (KDJW30019-E) is output to the standard error output and the log.
- You can only specify the standard attributes of the XML specifications and XML Schema specifications (such as the `xmlns` attribute). If you specify attributes other than the standard attributes of the XML specifications and XML Schema specifications, an error message (KDJW30019-E) is output to the standard error output and the log.

#

With the standard schema, the `javaee:handler-name` element is mandatory. If you specify `true` in the `com.cosminexus.jaxws.validation.handlerchain.strict` property in the action definition file, the JAX-WS engine checks whether this element is specified. If `true` is specified in the `com.cosminexus.jaxws.validation.handlerchain.strict` property, and if the `javaee:handler` element is omitted, an error message (KDJW30019-E) is output to the standard error output and the log.

19.4.5 javaee:handler-class element

This subsection describes the support range of the `javaee:handler-class` element.

- You can code only 1 `javaee:handler-class` element as the child element of the `javaee:handler` element. You cannot omit this element. If omitted, an error message (KDJW30019-E) is output to the standard error output and the log.
- You code the fully qualified name of the class implementing the handler in the value. If you code a non-existent class name, an error message is output to the standard error output and the log (KDJW00011-E).
- You cannot code child elements. If a child element is coded, an error message (KDJW30019-E) is output to the standard error output and the log.
- You can only specify the standard attributes of the XML specifications and XML Schema specifications (such as the `xmlns` attribute). If you specify attributes other than the standard attributes of the XML specifications and XML Schema specifications, an error message (KDJW30019-E) is output to the standard error output and the log.

19.4.6 javaee:soap-header element

This subsection describes the support range of the `javaee:soap-header` element.

- You can code the `javaee:soap-header` element as the child element of the `javaee:handler` element. You can omit this element.
- You code the name of the SOAP header processed by this handler in the value. However, this value is ignored with the JAX-WS engine, so you can code a value in the range that does not violate the restrictions of the `xsd:QName` type in the XML Schema specifications.
- You cannot code child elements. If a child element is coded, an error message (KDJW30019-E) is output to the standard error output and the log.
- You can only specify the standard attributes of the XML specifications and XML Schema specifications (such as the `xmlns` attribute). If you specify attributes other than the standard attributes of the XML specifications and XML Schema specifications, an error message (KDJW30019-E) is output to the standard error output and the log.

19.4.7 javaee:soap-role element

This subsection describes the support range of the `javaee:soap-role` element.

- You can code the `javaee:soap-role` element as the child element of the `javaee:handler` element. You can omit this element.
- You code the SOAP actor or role played by this handler in the value. However, you cannot code `http://www.w3.org/2003/05/soap-envelope/role/none`. If this value is coded, an error message (KDJW10007-E) is output to the standard error output and the log.
- You cannot code child elements. If a child element is coded, an error message (KDJW30019-E) is output to the standard error output and the log.
- You can only specify the standard attributes of the XML specifications and XML Schema specifications (such as the `xmlns` attribute). If you specify attributes other than the standard attributes of the XML specifications and XML Schema specifications, an error message (KDJW30019-E) is output to the standard error output and the log.

20 Support Range of WSDL Specification

This chapter describes the support range of the WSDL specifications that you must keep in mind when you develop a Web Service.

20.1 Support range of the WSDL 1.1 specifications

This section describes the support range of the WSDL 1.1 specifications for each WSDL element.

Notes

- For syntactic errors of the WSDL specifications supported with the Cosminexus JAX-WS functionality, such as when elements and attributes not mentioned in this chapter are specified, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.
- This section describes the constraints in the WSDL syntax. Even if there is no problem in the WSDL syntax, another error might occur if there is a problem in the mapping to Java. If the elements and attributes that are not referenced on WSDL are incorrect, an error might occur.

Furthermore, the notes on the WSDL extension elements and extension attributes are as follows:

- The extension elements for MIME binding (such as `mime:content`, `mime:mimeXml`) are not supported. Therefore, if you specify the MIME-related WSDL elements, an error message (KDJW51188-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.
- The following specifications are supported:

  - The extension elements for SOAP binding in the SOAP 1.1 specifications (such as `soap:header`, `soap:body`)

  - The extension elements for SOAP binding in the SOAP 1.2 specifications

  - The extension elements (such as `wsaw:UsingAddressing`) and extension attributes (such as `wsaw:Action`) in the WS-Addressing 1.0 specifications
- The binding declaration in the JAX-WS 2.2 specifications (such as `jaxws:bindings`)
- If you specify unsupported extension elements or extension attributes, such elements are ignored.

20.1.1 wsdl:definitions element

This subsection describes the support range of the `wsdl:definitions` element.

- You can code only one `wsdl:definitions` element as the root element of the WSDL file. This element cannot be omitted. If this element is omitted or if 2 or more elements are coded, a Cosminexus XML Processor error occurs.
- You can specify the following elements as the child element. If you specify elements other than the following WSDL elements, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. From among the extension elements that are not ignored by the `cjwsimport` command, if you specify extension elements other than the following elements, an error message (KDJW51053-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends:
  - `wsdl:documentation` element
  - `jaxws:bindings` element (binding declaration in the JAX-WS 2.2 specifications)
  - `wsdl:import` element
  - `wsdl:types` element
  - `wsdl:message` element
  - `wsdl:portType` element
  - `wsdl:binding` element
  - `wsdl:service` element
- The `wsdl:documentation` element and `jaxws:bindings` element must be specified in the above order. However, the other elements do not have a specified order.

  If the specification order of the `wsdl:documentation` element and `jaxws:bindings` element is incorrect, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.

- You can specify the following attributes. If you specify attributes other than those listed below, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends:
  - `name` attribute
  - `targetNamespace` attribute

(1) name attribute (wsdl:definitions element)

This point describes the support range of the `name` attribute included in the `wsdl:definitions` element.

- You can code 0 or 1 `name` attribute. If 2 or more attributes are coded, a Cosminexus XML Processor error occurs.
- For details on the values that can be specified, see *20.2(1) Values specifiable in the NCName type*.

(2) targetNamespace attribute (wsdl:definitions element)

This point describes the support range of the `targetNamespace` attribute included in the `wsdl:definitions` element.

- You can code only one `targetNamespace` attribute. This attribute cannot be omitted. If omitted, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. If 2 or more attributes are coded, a Cosminexus XML Processor error occurs.
- For details on the values that can be specified, see *15.1.1(2) Conditions for the namespace*.

20.1.2 wsdl:import element

This subsection describes the support range of the `wsdl:import` element.

- You can code 0 to 255 `wsdl:import` elements as the child elements of the `wsdl:definitions` element. If 256 or more elements are coded, an error message (KDJW51052-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.
- You can specify the `wsdl:documentation` element as the child element. If an element other than the `wsdl:documentation` element is specified, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. If you specify an extension element that is not ignored by the `cjwsimport` command, an error message (KDJW51054-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.
- You can specify the following attributes. If you specify attributes other than those listed below, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.
  - `namespace` attribute
  - `location` attribute

(1) namespace attribute (wsdl:import element)

This point describes the support range of the `namespace` attribute included in the `wsdl:import` element.

- You can code only one `namespace` attribute. This attribute cannot be omitted. If omitted, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. If 2 or more attributes are coded, a Cosminexus XML Processor error occurs.
- For details on the values that can be specified, see *26.3(1) namespace attribute (wsdl:import element)*.

(2) location attribute (wsdl:import element)

This point describes the support range of the `location` attribute included in the `wsdl:import` element.

- You can code only one `location` attribute. This attribute cannot be omitted. If omitted, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. If 2 or more attributes are coded, a Cosminexus XML Processor error occurs.
- You can specify any string that satisfies `xsd:anyURI`.
- For details on the values that can be specified, see *26.3(2) location attribute (wsdl:import element)*.

20.1.3 wsdl:types element

This subsection describes the support range of the `wsdl:types` element.

- You can code only one `wsdl:types` element as the child element of the `wsdl:definitions` element. This element cannot be omitted #. If this element is omitted or if 2 or more elements are coded, an error message (KDJW51049-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.

  #

  When you use the WSDL import functionality, one `wsdl:types` element is enough for all the WSDLs with an import relationship, so the element can be omitted in this case. Also, the total of the WSDLs with an import relationship forms the upper limit.
- You can specify the following elements as the child element. If you specify elements other than the following WSDL elements, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. If you specify an extension element that is not ignored by the `cjwsimport` command, an error message (KDJW51059-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.
  - `wsdl:documentation` element
  - `xsd:schema` element
- The `wsdl:documentation` element and `xsd:schema` element must be specified in the above order. If the specification order is incorrect, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.
- You cannot specify attributes. If you specify attributes, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.

20.1.4 wsdl:message element

This subsection describes the support range of the `wsdl:message` element.

- You can code one or more `wsdl:message` elements as the child element of the `wsdl:definitions` element. This element cannot be omitted #. If omitted, an error message (KDJW51050-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.

  #

  When you use the WSDL import functionality, one `wsdl:message` element is enough for all the WSDLs with an import relationship. So the element can be omitted in this case. Also, the total of the WSDLs with an import relationship forms the upper limit.
- You can specify the following elements as the child element. If you specify elements other than the following WSDL elements, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. If you specify extension elements that are not ignored by the `cjwsimport` command, an error message (KDJW51055-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.
  - `wsdl:documentation` element
  - `wsdl:part` element
- The `wsdl:documentation` element and `wsdl:part` element must be specified in the above order. If the specification order is incorrect, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.

- You can specify the `name` attribute. You cannot specify attributes other than the `name` attribute. If attributes other than the `name` attribute are specified, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.

(1) name attribute (wsdl:message element)

This point describes the support range of the `name` attribute included in the `wsdl:message` element.

- You can code only one `name` attribute. This attribute cannot be omitted. If omitted, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. If 2 or more attributes are coded, a Cosminexus XML Processor error occurs.
- For details on the values that can be specified, see *20.2(1) Values specifiable in the NCName type*. However, you cannot specify the same value as the `name` attribute of the other `wsdl:message` elements existing under the same `wsdl:definitions` element. If the same value is specified, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.

20.1.5 wsdl:part element

This subsection describes the support range of the `wsdl:part` element.

- You can code 0 to 255 `wsdl:part` elements as the child elements of the `wsdl:message` element. If 256 or more elements are coded, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.
- For details on the values that can be specified, see *20.2 Notes on creating the WSDL*.
- You can specify the `wsdl:documentation` element as the child element. If you specify an element other than the `wsdl:documentation` element, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. If you specify extension elements that are not ignored by the `cjwsimport` command, an error message (KDJW51056-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.
- You can specify the following attributes. If you specify attributes other than those listed below, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.
  - `name` attribute
  - `element` attribute

(1) name attribute (wsdl:part element)

This point describes the support range of the `name` attribute included in the `wsdl:part` element.

- You can code only one `name` attribute. This attribute cannot be omitted. If omitted, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. If 2 or more attributes are coded, a Cosminexus XML Processor error occurs.
- For details on the values that can be specified, see *15.1.5(2) Conditions for part names*.

(2) element attribute (wsdl:part element)

This point describes the support range of the `element` attribute included in the `wsdl:part` element.

- You can code only one `element` attribute. This attribute cannot be omitted. If omitted, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. If 2 or more attributes are coded, a Cosminexus XML Processor error occurs.
- Use `QName` to specify the element declaration declared beneath the referable `wsdl:types` element.

20.1.6 wsdl:portType element

This subsection describes the support range of the `wsdl:portType` element.

- You can code 1 to 255 `wsdl:portType` elements as the child element of the `wsdl:portType` element. This element cannot be omitted #. For details on the cases where the element is omitted or if 256 or more elements are coded, see *15.1.2(3) Number of port types coded*.

  #

  When you use the WSDL import functionality, one `wsdl:portType` element is enough for all the WSDLs with an import relationship. So the element can be omitted in this case. Also, the total for all the WSDLs with an import relationship forms the upper limit.
- You can specify the following elements as the child element. If you specify elements other than the following WSDL elements, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. From among the extension elements that are not ignored by the `cjwsimport` command, if you specify extension elements other than the following elements, an error message (KDJW51074-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends:
  - `wsdl:documentation` element
  - `jaxws:bindings` element (binding declaration in the JAX-WS 2.2 specifications)
  - `wsdl:operation` element
- The `wsdl:documentation` element, `jaxws:bindings` element, and `wsdl:operation` element must be specified in the above order. If the specification order is incorrect, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.
- You can specify the `name` attribute. You cannot specify attributes other than the `name` attribute. If you specify attributes other than the `name` attribute, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.

(1) name attribute (portType element)

This point describes the support range of the `name` attribute included in the `wsdl:portType` element.

- You can code only one `name` attribute. This attribute cannot be omitted. If omitted, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. If 2 or more attributes are coded, a Cosminexus XML Processor error occurs.
- For details on the values that can be specified, see *15.1.2(2) Conditions for the port type name*. However, you cannot specify the same value as the `name` attribute of the other `wsdl:portType` elements existing beneath the same `wsdl:definitions` element. If the same value is specified, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.

20.1.7 wsdl:operation element (For the child element of the wsdl:portType element)

This subsection describes the support range of the `wsdl:operation` element.

- You can code 1 to 255 `wsdl:operation` elements as the child elements of the `wsdl:portType` element. This element cannot be omitted. For details on the cases where this element is omitted or where 256 or more elements are coded, see *15.1.3(3) Number of operations and its child elements coded*.
- You can specify the following elements as the child element. If you specify elements other than the following WSDL elements, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. From among the extension elements that are not ignored by the `cjwsimport` command, if you specify extension elements other than the following elements, an error message (KDJW51080-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends:
  - `wsdl:documentation` element

  - `jaxws:bindings` element (binding declaration in the JAX-WS 2.2 specifications)
  - `wsdl:input` element
  - `wsdl:output` element
  - `wsdl:fault` element
- The `wsdl:documentation` element, `jaxws:bindings` element, `wsdl:input` element, `wsdl:output` element, and `wsdl:fault` element must be specified in the above order. If the specification order is incorrect, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.
- You can specify the `name` attribute. You cannot specify attributes other than the `name` attribute. If you specify attributes other than the `name` attribute, the operations are not guaranteed.

(1) name attribute (wsdl:operation element)

This point describes the support range of the `name` attribute included in the `wsdl:operation` element.

- You can code only one `name` attribute. This attribute cannot be omitted. If omitted, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. If 2 or more attributes are coded, a Cosminexus XML Processor error occurs.
- For details on the values that can be specified, see *15.1.3(2) Conditions for the operation name*. However, you cannot specify the same value as the `name` attribute of the other `wsdl:portType` elements existing beneath the same `wsdl:operation` element. If the same value is specified, an error message (KDJW51083-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.

20.1.8 wsdl:input element (For the grandchild element of the wsdl:portType element)

This subsection describes the support range of the `wsdl:input` element.

- You can code only one `wsdl:input` element as the child element of the `wsdl:operation` element. This element cannot be omitted. For details on the case where the element is omitted or if 2 or more elements are coded, see *15.1.3(3) Number of operations and its child elements coded*.
- You can specify the `wsdl:documentation` element as the child element. If you specify an element other than the `wsdl:documentation` element, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. If you specify extension elements that are not ignored by the `cjwsimport` command, an error message (KDJW51057-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.
- You can specify the following attributes. If attributes other than those listed below are specified, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends:
  - `name` attribute
  - `message` attribute
  - `wsaw:Action` attribute

  For details on the `wsaw:Action` attribute, see *37.5.6 Operations when the wsa:Action element is specified*.

(1) name attribute (wsdl:input element)

This point describes the support range of the `name` attribute included in the `wsdl:input` element.

- You can code 0 or 1 `name` attribute. If 2 or more attributes are coded, a Cosminexus XML Processor error occurs.
- For details on the values that can be specified, see *20.2(1) Values specifiable in the NCName type*.

(2) message attribute (wsdl:input element)

This point describes the support range of the `message` attribute included in the `wsdl:input` element.

- You can code only one `message` attribute as the child element of the `wsdl:input` element. This attribute cannot be omitted. If omitted, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. If 2 or more attributes are coded, a Cosminexus XML Processor error occurs.
- Use `QName` to specify `wsdl:message` declared beneath the referable `wsdl:definitions` element.

20.1.9 wsdl:output element (For the grandchild element of the wsdl:portType element)

This subsection describes the support range of the `wsdl:output` element.

- You can code only one `wsdl:output` element as the child element of the `wsdl:operation` element. For details on the case where the element is omitted or if 2 or more elements are coded, see *15.1.3(3) Number of operations and its child elements coded*.
- You can specify the `wsdl:documentation` element as the child element. If you specify an element other than the `wsdl:documentation` element, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. If you specify extension elements that are not ignored by the `cjwsimport` command, an error message (KDJW51058-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.
- You can specify the following attributes. If attributes other than those listed below are specified, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends:
  - `name` attribute
  - `message` attribute
  - `wsaw:Action` attribute

  For details on the `wsaw:Action` attribute, see *37.5.6 Operations when the wsa:Action element is specified*.

(1) name attribute (wsdl:output element)

This point describes the support range of the `name` attribute included in the `wsdl:output` element.

- You can code 0 or 1 `name` attribute. If 2 or more attributes are coded, a Cosminexus XML Processor error occurs.
- For details on the values that can be specified, see *20.2(1) Values specifiable in the NCName type*.

(2) message attribute (wsdl:output element)

This point describes the support range of the `message` attribute included in the `wsdl:output` element.

- You can code only one `message` attribute. This attribute cannot be omitted. If omitted, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. If 2 or more attributes are coded, a Cosminexus XML Processor error occurs.
- Use `QName` to specify `wsdl:message` declared beneath the referable `wsdl:definitions` element.

20.1.10 wsdl:fault element (For the grandchild element of the wsdl:portType element)

This subsection describes the support range of the `wsdl:fault` element.

- You can code 0 to 255 `wsdl:fault` elements as the child element of the `wsdl:operation` element. For details on the case where the element is omitted or if 256 or more elements are coded, see *15.1.3(3) Number of operations and its child elements coded*.
- You can specify the following elements as the child element. If you specify elements other than the following WSDL elements, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. From among the extension elements that are not ignored by the `cjwsimport` command, if you specify extension elements other than the following extension elements, an error message (KDJW51096-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends:
  - `wsdl:documentation` element
  - `jaxws:bindings` element (binding declaration in the JAX-WS 2.2 specifications)
- The `wsdl:documentation` element and `jaxws:bindings` element must be specified in the above order. If the specification order is incorrect, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.
- You can specify the following attributes. If attributes other than those listed below are specified, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends:
  - `name` attribute
  - `message` attribute
  - `wsaw:Action` attribute

  For details on the `wsaw:Action` attribute, see *37.5.6 Operations when the wsa:Action element is specified*.

(1) name attribute (wsdl:fault element)

This point describes the support range of the `name` attribute included in the `wsdl:fault` element.

- You can code only one `name` attribute. This attribute cannot be omitted. If omitted, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. If 2 or more attributes are coded, a Cosminexus XML Processor error occurs.
- For details on the values that can be specified, see *20.2(1) Values specifiable in the NCName type*.

(2) message attribute (wsdl:fault element)

This point describes the support range of the `message` attribute included in the `wsdl:fault` element.

- You can code only one `message` attribute. This attribute cannot be omitted. If omitted, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. If 2 or more attributes are coded, a Cosminexus XML Processor error occurs.
- Use `QName` to specify `wsdl:message` declared beneath the referable `wsdl:definitions` element.

20.1.11 wsdl:binding element

This subsection describes the support range of the `wsdl:binding` element.

- You can code 1 to 255 `wsdl:binding` elements as the child elements of the `wsdl:definitions` element. This element cannot be omitted [#]. If omitted or if 256 or more elements are coded, an error message (KDJW51100-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.

  #

  When you use the WSDL import functionality, one `wsdl:binding` element is enough for all the WSDLs with an import relationship. So the element can be omitted in this case. Also, the total `wsdl:binding` elements in all the WSDLs with an import relationship forms the upper limit.

- You can specify the following elements as the child element. If you specify elements other than the following WSDL elements, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. From among the extension elements that are not ignored by the `cjwsimport` command, if you specify extension elements other than the following elements, an error message is output to the standard error output and the log, and the processing of the `cjwsimport` command ends (KDJW51102-E):
  - `wsdl:documentation` element
  - `soap:binding` element (extension element for SOAP binding in the SOAP 1.1 specifications)#
  - `soap12:binding` element (extension element for SOAP binding in the SOAP 1.2 specifications)#
  - `jaxws:bindings` element (binding declaration in the JAX-WS 2.2 specifications)
  - `wsaw:UsingAddressing` element (extension element in the WS-Addressing 1.0 specifications)
  - `wsdl:operation` element

  #

  Select and specify either the `soap:binding` element or the `soap12:binding` element. You cannot specify both the elements.
- Specify the child elements of the `wsdl:binding` element in the above order. If the specification order is incorrect, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. However, you might switch the order of the following elements:
  - `soap:binding` element or `soap12:binding` element
  - `jaxws:bindings` element
  - `wsaw:UsingAddressing` element
- You can specify the following attributes. If attributes other than those listed below are specified, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends:
  - `name` attribute
  - `type` attribute

(1) name attribute (wsdl:binding element)

This point describes the support range of the `name` attribute included in the `wsdl:binding` element.

- You can code only one `name` attribute. This attribute cannot be omitted. If omitted, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. If 2 or more attributes are coded, a Cosminexus XML Processor error occurs.
- For details on the values that can be specified, see *20.2(1) Values specifiable in the NCName type*.

(2) type attribute (wsdl:binding element)

This point describes the support range of the `type` attribute included in the `wsdl:binding` element.

- You can code only one `type` attribute. This attribute cannot be omitted. If omitted, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. If 2 or more attributes are coded, a Cosminexus XML Processor error occurs.
- Use `QName` to specify `wsdl:message` declared beneath the referable `wsdl:definitions` element.

20.1.12 wsdl:operation element (For the child element of the wsdl:binding element)

This subsection describes the support range of the `wsdl:operation` element.

- You can code 1 to 255 `wsdl:operation` elements as the child elements of the `wsdl:binding` element. This element cannot be omitted. If omitted or if 256 or more elements are coded, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.
- You can specify the following elements as the child element. If you specify elements other than the following WSDL elements, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. From among the extension elements that are not ignored by the `cjwsimport` command, if you specify extension elements other than the following elements, an error message (KDJW51108-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends:
  - `wsdl:documentation` element
  - `soap:operation` element (extension element for SOAP binding in the SOAP 1.1 specifications)#
  - `soap12:operation` element (extension element for SOAP binding in the SOAP 1.2 specifications)#
  - `jaxws:bindings` element (binding declaration in the JAX-WS 2.2 specifications)
  - `wsaw:Anonymous` element (extension element in the WS-Addressing 1.0 specifications)
  - `wsdl:input` element
  - `wsdl:output` element
  - `wsdl:fault` element

  #

  Select and specify either the `soap:operation` element or the `soap12:operation` element. You cannot specify both the elements.
- Specify the child elements of the `wsdl:operation` element (for the child element of the `wsdl:binding` element) in the above order. If the specification order is incorrect, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. However, you might switch the order of the following elements:
  - `soap:operation` element or `soap12:operation` element
  - `jaxws:bindings` element
  - `wsaw:Anonymous` element
- You can specify the `name` attribute. You cannot specify attributes other than the `name` attribute. If you specify attributes other than the `name` attribute, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.
- Define the `wsdl:operation` element of the `wsdl:binding` element so that it corresponds to the `wsdl:operation` element defined in the `wsdl:portType` element. If the element is not defined in this manner, an error message (KDJW51112-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.
- When you specify the `wsdl:output` element as a child element, define the `wsdl:output` element so that it corresponds to the `wsdl:output` element defined in the `wsdl:operation` element of the `wsdl:portType` element.
  - If you code one `wsdl:output` element in the `wsdl:operation` element of the `wsdl:portType` element and omit the `wsdl:output` element from the `wsdl:operation` element of the `wsdl:binding` element, an error message is output in the standard error output and log and then the processing of the `cjwsimport` command ends (KDJW51222-E).
  - When you omit the `wsdl:output` element from the `wsdl:operation` element of the `wsdl:portType` element and code one `wsdl:output` element in the `wsdl:operation` element of the `wsdl:binding` element, the `wsdl:output` element is ignored and, the processing of the `cjwsimport` command continues.

(1) name attribute (wsdl:operation element)

This point describes the support range of the `name` attribute included in the `wsdl:operation` element.

- You can code only one `name` attribute. This attribute cannot be omitted. If omitted, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. If 2 or more attributes are coded, a Cosminexus XML Processor error occurs.
- For details on the values that can be specified, see *20.2(1) Values specifiable in the NCName type*.

20.1.13 wsdl:input element (For the grandchild element of the wsdl:binding element)

This subsection describes the support range of the `wsdl:input` element.

- You can code only one `wsdl:input` element as the child element of the `wsdl:operation` element. This element cannot be omitted. If omitted or if 2 or more elements are coded, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.
- You can specify the following elements as the child element. If you specify elements other than the following WSDL elements, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. From among the extension elements that are not ignored by the `cjwsimport` command, if you specify extension elements other than the following elements, an error message (KDJW51114-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.
  - `wsdl:documentation` element
  - `soap:header` element (extension element for SOAP binding in the SOAP 1.1 specifications)#
  - `soap:body` element (extension element for SOAP binding in the SOAP 1.1 specifications)#
  - `soap12:header` element (extension element for SOAP binding in the SOAP 1.2 specifications)#
  - `soap12:body` element (extension element for SOAP binding in the SOAP 1.2 specifications)#

  #

  Select and specify either the `soap:header` element or the `soap12:header` element, and the `soap:body` element or the `soap12:header` element. You cannot specify both or mix the elements of the SOAP 1.1 specifications and the SOAP 1.2 specifications.
- Specify the child elements of the `wsdl:input` element (for the grandchild element of the `wsdl:binding` element) in the above order. If the specification order is incorrect, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. However, you might switch the order of the following elements:
  - `soap:header` element or `soap12:header` element
  - `soap:body` element or `soap12:body` element
- You can specify the `name` attribute. You cannot specify attributes other than the `name` attribute. If you specify attributes other than the `name` attribute, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.

(1) name attribute (wsdl:input element)

This point describes the support range of the `name` attribute included in the `wsdl:input` element.

- You can code 0 or 1 `name` attribute. If 2 or more attributes are coded, a Cosminexus XML Processor error occurs.
- For details on the values that can be specified, see *20.2(1) Values specifiable in the NCName type*.

20.1.14 wsdl:output element (For the grandchild element of the wsdl:binding element)

This subsection describes the support range of the `wsdl:output` element.

- You can code only one `wsdl:output` element as the child element of the `wsdl:operation` element. If omitted or if 2 or more elements are coded, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.
- You can specify the following elements as the child element. If you specify elements other than the following WSDL elements, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. From among the extension elements that are not ignored by the `cjwsimport` command, if you specify extension elements other than the following extension elements, an error message (KDJW51119-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends:
  - `wsdl:documentation` element
  - `soap:header element (extension element for SOAP binding in the SOAP 1.1 specifications)`#
  - `soap:body element (extension element for SOAP binding in the SOAP 1.1 specifications)`#
  - `soap12:header element (extension element for SOAP binding in the SOAP 1.2 specifications)`#
  - `soap12:body element (extension element for SOAP binding in the SOAP 1.2 specifications)`#

  #

  Select and specify either the `soap:header` element or the `soap12:header` element, and the `soap:body` element or the `soap12:header` element. You cannot specify both or mix the elements of the SOAP 1.1 specifications and the SOAP 1.2 specifications.
- Specify the child elements of the `wsdl:output` element (for the grandchild element of the `wsdl:binding` element) in the above order. If the specification order is incorrect, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. However, you might switch the order of the following elements:
  - `soap:header` element or `soap12:header` element
  - `soap:body` element or `soap12:body` element
- You can specify the `name` attribute. You cannot specify attributes other than the `name` attribute. If you specify attributes other than the `name` attribute, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.

(1) name attribute (wsdl:output element)

This point describes the support range of the `name` attribute included in the `wsdl:output` element.

- You can code 0 or 1 `name` attribute. If 2 or more attributes are coded, a Cosminexus XML Processor error occurs.
- For details on the values that can be specified, see *20.2(1) Values specifiable in the NCName type*.

20.1.15 wsdl:fault element (For the grandchild element of the wsdl:binding element)

This subsection describes the support range of the `wsdl:fault` element.

- You can code 0 to 255 `wsdl:fault` elements as the child elements of the `wsdl:operation` element. If 256 or more elements are coded, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.
- You can specify the following elements as the child element. If you specify elements other than the following WSDL elements, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. From among the extension elements that are not ignored by the `cjwsimport` command, if you specify extension elements other than the following extension elements, an error

message (KDJW51123-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends:

- `wsdl:documentation` element
- `soap:fault element (extension element for SOAP binding in the SOAP 1.1 specifications)#`
- `soap12:fault` element (extension element for SOAP binding in the SOAP 1.2 specifications)#

#

Select and specify either the `soap:fault` element or the `soap12:fault` element. You cannot specify both the elements.

- Specify the child elements of the `wsdl:fault` element (for the grandchild element of the `wsdl:binding` element) in the above order. If the specification order is incorrect, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.
- You can specify the `name` attribute. You cannot specify attributes other than the `name` attribute. If you specify attributes other than the `name` attribute, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.

(1) name attribute (wsdl:fault element)

This point describes the support range of the `name` attribute included in the `wsdl:fault` element.

- You can code only one `name` attribute. If omitted, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. If 2 or more attributes are coded, a Cosminexus XML Processor error occurs.
- For details on the values that can be specified, see *20.2(1) Values specifiable in the NCName type*. However, you cannot specify the same value as the `name` attribute of the other `wsdl:fault` elements existing beneath the same `wsdl:operation` element. If you specify the same value, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.

20.1.16 wsdl:service element

This subsection describes the support range of the `wsdl:service` element.

- You can code 1 to 255 `wsdl:service` elements as the child elements of the `wsdl:definitions` element. This element cannot be omitted #. If omitted or if 256 or more elements are coded, an error message (KDJW51127-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.

  #

  When you use the WSDL import functionality, one `wsdl:service` element is enough for all the WSDLs with an import relationship. So the element can be omitted in this case. Also, the total `wsdl:service` elements in all the WSDLs with an import relationship forms the upper limit.
- You can specify the following elements as the child element. If you specify elements other than the following WSDL elements, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. From among the extension elements that are not ignored by the `cjwsimport` command, if you specify extension elements other than the following extension elements, an error message (KDJW51129-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends:
  - `wsdl:documentation element`
  - `jaxws:bindings element (binding declaration in the JAX-WS 2.2 specifications)`
  - `wsdl:port element`
- Specify the child elements of the `wsdl:service` element in the above order. If the specification order is incorrect, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.

- You can specify the `name` attribute. You cannot specify attributes other than the `name` attribute. If you specify attributes other than the `name` attribute, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.

(1) name attribute (wsdl:service element)

This point describes the support range of the `name` attribute included in the `wsdl:service` element.

- You can code only one `name` attribute. This attribute cannot be omitted. If omitted, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. If 2 or more attributes are coded, a Cosminexus XML Processor error occurs.
- For details on the values that can be specified, see *15.1.9(2) Conditions for the service name and port name*. However, you cannot specify the same value as the `name` attribute of the other `wsdl:service` elements existing beneath the same `wsdl:definitions` element. If the same value is specified, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.

20.1.17 wsdl:port element

This subsection describes the support range of the `wsdl:port` element.

- You can code 1 to 255 `wsdl:port` elements as the child elements of the `wsdl:service` element. This element cannot be omitted. If omitted or if 256 or more elements are coded, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.
- You can specify the following elements as the child element. If you specify elements other than the following WSDL elements, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. From among the extension elements that are not ignored by the `cjwsimport` command, if you specify extension elements other than the following elements, an error message (KDJW51135-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.
  - `wsdl:documentation` element
  - `soap:address` element (extension element for SOAP binding in the SOAP 1.1 specifications)#
  - `soap12:address` element (extension element for SOAP binding in the SOAP 1.2 specifications)#
  - `jaxws:bindings` element (binding declaration in the JAX-WS 2.2 specifications)
  - `wsaw:UsingAddressing` element (extension element in the WS-Addressing 1.0 specifications)

  #

  Select and specify either the `soap:address` element or the `soap12:address` element. You cannot specify both the elements.
- Specify the child elements of the `wsdl:port` element in the above order. If the specification order is incorrect, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. However, you might switch the order of the following elements:
  - `soap:address` element or `soap12:address` element
  - `jaxws:bindings` element
  - `wsaw:UsingAddressing` element
- You can specify the following attributes. If attributes other than those listed below are specified, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends:
  - `name` attribute
  - `binding` attribute

(1) name attribute (wsdl:port element)

This point describes the support range of the `name` attribute included in the `wsdl:port` element:

- You can code only one `name` attribute. This attribute cannot be omitted. If omitted, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. If 2 or more attributes are coded, a Cosminexus XML Processor error occurs.
- For details on the values that can be specified, see *15.1.9(2) Conditions for the service name and port name*. However, you cannot specify the same value as the `name` attribute of the other `wsdl:port` elements existing beneath the same `wsdl:service` element. If the same value is specified, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.

(2) binding attribute (wsdl:port element)

This point describes the support range of the `binding` attribute included in the `wsdl:port` element.

- You can code only one `binding` attribute. This attribute cannot be omitted. If omitted, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. If 2 or more attributes are coded, a Cosminexus XML Processor error occurs.
- Use `QName` to specify `wsdl:binding` declared beneath the referable `wsdl:definitions` element.

20.1.18 wsdl:documentation element

This subsection describes the support range of the `wsdl:documentation` element.

- You can code 0 or 1 `wsdl:documentation` element as the child element of the following elements. If 2 or more elements are coded, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. From among the WSDL elements and the extension elements that are not ignored by the `cjwsimport` command, if the `wsdl:documentation` element is coded as the child element of the following elements, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends:
  - `wsdl:definitions element`
  - `wsdl:import element`
  - `wsdl:types element`
  - `wsdl:message element`
  - `wsdl:part element`
  - `wsdl:portType element`
  - `wsdl:operation element`
  - `wsdl:input element`
  - `wsdl:output element`
  - `wsdl:fault element`
  - `wsdl:binding element`
  - `wsdl:service element`
  - `wsdl:port element`
- You can specify any element as the child element.
- You cannot specify attributes. If attributes are specified, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.

20.1.19 soap:binding element

This subsection describes the support range of the `soap:binding` element.

- You can code only one `soap:binding` element as the `wsdl:binding` element. This element cannot be omitted. If omitted or if 2 or more elements are coded, an error message (KDJW51143-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.
- You cannot code child elements. If coded, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.
- You can specify the following attributes. If attributes other than those listed below are coded, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends:
  - `transport attribute`
  - `style` attribute

(1) transport attribute (soap:binding element)

This point describes the support range of the `transport` attribute included in the `soap:binding` element.

- You can code only one `transport` attribute. This attribute cannot be omitted. If omitted, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. If 2 or more attributes are coded, a Cosminexus XML Processor error occurs.
- You code `http://schemas.xmlsoap.org/soap/http` as the value. If a value other than `http://schemas.xmlsoap.org/soap/http` is coded, an error message (KDJW51147-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.

(2) style attribute (soap:binding element)

This point describes the support range of the `style` attribute included in the `soap:binding` element.

- You can code 0 or 1 `style` attribute. If 2 or more attributes are coded, a Cosminexus XML Processor error occurs.
- You code `document` as the value. If a value other than `document` is coded, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.

20.1.20 soap:operation element

This subsection describes the support range of the `soap:operation` element.

- You can code only one `soap:operation` element as the child element of the `wsdl:operation` element. This element cannot be omitted. If omitted or if 2 or more elements are coded, an error message (KDJW51150-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.
- You cannot code child elements. If coded, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.
- You can specify the following attributes. If attributes other than those listed below are coded, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends:
  - `soapAction` attribute
  - `style` attribute

(1) soapAction attribute (soap:operation element)

This point describes the support range of the `soapAction` attribute included in the `soap:operation` element.

- You can code 0 or 1 `soapAction` attribute. If 2 or more attributes are coded, a Cosminexus XML Processor error occurs.
- You can specify any string that satisfies `xsd:anyURI`.
- The value specified in this attribute is ignored.

(2) style attribute (soap:operation element)

This point describes the support range of the `style` attribute included in the `soap:operation` element.

- You can code 0 or 1 `style` attribute. If 2 or more attributes are coded, a Cosminexus XML Processor error occurs.
- You code `document` as the value. If a value other than `document` is coded, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.

20.1.21 soap:body element

This subsection describes the support range of the `soap:body` element.

- You can code only one `soap:body` element as the child element of the `wsdl:input` and `wsdl:output` elements that are the grandchild elements of the `wsdl:binding` element. This element cannot be omitted. If omitted or if 2 or more elements are coded, an error message (KDJW51156-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.
- You cannot code child elements. If coded, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.
- You can specify the following attributes. If attributes other than those listed below are specified, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. However, if you specify the `namespace` attribute, a different error message (KDJW51208-E) is output to the standard error output and the log (KDJW51208-E).
  - `use attribute`
  - `parts attribute`

(1) use attribute (soap:body element)

This point describes the support range of the `use` attribute included in the `soap:body` element.

- You can code 0 or 1 `use` attribute. If 2 or more attributes are coded, a Cosminexus XML Processor error occurs.
- You code `literal` as the value. If a value other than `literal` is coded, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.

(2) parts attribute (soap:body element)

This point describes the support range of the `parts` attribute included in the `soap:body` element.

- You can code 0 or 1 `parts` attribute. If 2 or more attributes are coded, a Cosminexus XML Processor error occurs.
- Specify 0 or 1 `wsdl:part` element that is declared beneath the `wsdl:message` element referenced in the `soap:body` element. If 2 or more attributes are coded, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.
- For details on the notes on specifying the `parts` attribute, see *20.2(2) Coding the SOAP body and SOAP header and the referenced wsdl:part element*.

20.1.22 soap:header element

This subsection describes the support range of the `soap:header` element.

- You can code the `soap:header` element as the child element of the `wsdl:input` and `wsdl:output` elements that are the grandchild elements of the `wsdl:binding` element. You can also omit the `soap:header` element.
- You cannot code child elements. If coded, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.

- You can specify the following attributes. If attributes other than those listed below are specified, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. However, if you specify the `namespace` attribute, a warning message is output to the standard error output and the log, and the processing of the `cjwsimport` command continues (KDJW51009-W).
  - `message attribute`
  - `part attribute`
  - `use attribute`
- For details on the notes on specifying the `soap:header` element, see *20.2(2) Coding the SOAP body and SOAP header and the referenced wsdl:part element*.

(1) message attribute (soap:header element)

This point describes the support range of the `message` attribute included in the `soap:header` element.

- You can code only one `message` attribute. This attribute cannot be omitted. If omitted, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. If 2 or more attributes are coded, a Cosminexus XML Processor error occurs.
- Use `QName` to specify `wsdl:message` declared beneath the referable `wsdl:definitions` element.
- For details on the notes on specifying the `message` attribute, see *20.2(2) Coding the SOAP body and SOAP header and the referenced wsdl:part element*.

(2) part attribute (soap:header element)

This point describes the support range of the `part` attribute included in the `soap:header` element.

- You can code only one `part` attribute. This attribute cannot be omitted. If omitted, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. If 2 or more attributes are coded, a Cosminexus XML Processor error occurs.
- Specify the `wsdl:part` element declared beneath the `wsdl:message` element specified in the `message` attribute.
- For details on the notes on specifying the `part` attribute, see *20.2(2) Coding the SOAP body and SOAP header and the referenced wsdl:part element*.

(3) use attribute (soap:header element)

This point describes the support range of the `use` attribute included in the `soap:header` element.

- You can code only one `use` attribute. If omitted, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. If 2 or more attributes are coded, a Cosminexus XML Processor error occurs.
- You code `literal` as the value. If a value other than `literal` is coded, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.

20.1.23 soap:fault element

This subsection describes the support range of the `soap:fault` element.

- You can code only one `soap:fault` element as the child element of the `wsdl:fault` element that forms the grandchild element of the `wsdl:binding` element. If this element is omitted or if 2 or more elements are coded, an error message is output to the standard error output and the log, and the processing of the `cjwsimport` command ends (KDJW51051-E).
- You cannot code child elements. If coded, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.

- You can specify the following attributes. If attributes other than those listed below are specified, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. However, if you specify the namespace attribute, a different error message (KDJW51210-E) is output to the standard error output and the log.
  - `name attribute`
  - `use attribute`

(1) name attribute (soap:fault element)

This point describes the support range of the `name` attribute included in the `soap:fault` element.

- You can code only one `name` attribute. This attribute cannot be omitted. If omitted, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. If 2 or more attributes are coded, a Cosminexus XML Processor error occurs.
- Code the same value as the `name` attribute of the `wsdl:fault` element that is the parent element. If a different value is coded, a warning message (KDJW51027-W) is output to the standard error output and the log, and the processing of the `cjwsimport` command continues.
- For details on the values that can be specified, see *20.2(1) Values specifiable in the NCName type*.

(2) use attribute (soap:fault element)

This point describes the support range of the `use` attribute included in the `soap:fault` element.

- You can code 0 or 1 `use` attribute. If 2 or more attributes are coded, a Cosminexus XML Processor error occurs.
- You code `literal` as the value. If a value other than `literal` is coded, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.

20.1.24 soap:address element

This subsection describes the support range of the `soap:address` element.

- You can code only one `soap:address` element as the child element of the `wsdl:port` element. This element cannot be omitted. If omitted or if 2 or more elements are coded, an error message (KDJW51175-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.
- You cannot code child elements. If coded, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.
- You can specify the `location` attribute. If you specify an attribute other than the `location` attribute, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.

(1) location attribute (soap:address element)

This point describes the support range of the `location` attribute included in the `soap:address` element.

- You can code only one `location` attribute. This attribute cannot be omitted. If omitted, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. If 2 or more attributes are coded, a Cosminexus XML Processor error occurs.
- For details on the values that can be specified, see *20.2(3) Values specifiable in the location attribute of the soap:address element or soap12:address element*.

20.1.25 soap12:operation element

This subsection describes the support range of the `soap12:operation` element.

- You can code only one `soap12:operation` element as the child element of the `wsdl:operation` element. This element cannot be omitted. If omitted or if 2 or more elements are coded, an error message (KDJW51150-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.
- You cannot code child elements. If coded, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.
- You can specify the following attributes. If attributes other than those listed below are specified, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. However, if you specify the `soapActionRequired` attribute, the processing of the `cjwsimport` command continues.
  - `soapAction` attribute
  - `style` attribute

(1) soapAction attribute (soap12:operation element)

This point describes the support range of the `soapAction` attribute included in the `soap12:operation` element.

- You can code 0 or 1 `soapAction` attribute. If 2 or more attributes are coded, a Cosminexus XML Processor error occurs.
- You can specify any string that satisfies `xsd:anyURI`.
- The specified `soapAction` attribute is ignored with the JAX-WS engine.

(2) style attribute (soap12:operation element)

This point describes the support range of the `style` attribute included in the `soap12:operation` element.

- You can code 0 or 1 `style` attribute. If 2 or more attributes are coded, a Cosminexus XML Processor error occurs.
- You code `document` as the value. If a value other than `document` is coded, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.

20.1.26 soap12:binding element

This subsection describes the support range of the `soap12:binding` element.

- You can code only one `soap12:binding` element as the child element of the `wsdl:binding` element. This element cannot be omitted. If omitted or if 2 or more elements are coded, an error message (KDJW51143-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.
- You cannot code child elements. If coded, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.
- You can specify the following attributes. If attributes other than those listed below are specified, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends:
  - `transport` attribute
  - `style` attribute

(1) transport attribute (soap12:binding element)

This point describes the support range of the `transport` attribute included in the `soap12:binding` element.

- You can code only one `transport` attribute. This attribute cannot be omitted. If omitted, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. If 2 or more attributes are coded, a Cosminexus XML Processor error occurs.
- You code `http://schemas.xmlsoap.org/soap/http` or `http://www.w3.org/2003/05/soap/bindings/HTTP/` as the value. If a value other than `http://schemas.xmlsoap.org/soap/http` or

`http://www.w3.org/2003/05/soap/bindings/HTTP/` is coded, an error message (KDJW51147-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.

(2) style attribute (soap12:binding element)

This point describes the support range of the `style` attribute included in the `soap12:binding` element.

- You can code 0 or 1 `style` attribute. If 2 or more attributes are coded, a Cosminexus XML Processor error occurs.
- You code `document` as the value. If a value other than `document` is coded, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.

20.1.27 soap12:body element

This subsection describes the support range of the `soap12:body` element.

- You can code only one `soap12:body` element as the child element of the `wsdl:input` and `wsdl:output` elements that are the grandchild elements of the `wsdl:binding` element. This element cannot be omitted. If omitted or if 2 or more elements are coded, an error message (KDJW51156-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.
- You cannot code child elements. If coded, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.
- You can specify the following attributes. If attributes other than those listed below are specified, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. However, if you specify the `namespace` attribute, a different error message (KDJW51208-E) is output to the standard error output and the log.
  - `use attribute`
  - `parts attribute`

(1) use attribute (soap12:body element)

This point describes the support range of the `use` attribute included in the `soap12:body` element.

- You can code 0 or 1 `use` attribute. If 2 or more attributes are coded, a Cosminexus XML Processor error occurs.
- You code `literal` as the value. If a value other than `literal` is coded, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.

(2) parts attribute (soap12:body element)

This point describes the support range of the `parts` attribute included in the `soap12:body` element.

- You can code 0 or 1 `parts` attribute. If 2 or more attributes are coded, a Cosminexus XML Processor error occurs.
- You can code 0 or 1 `wsdl:part` element that is declared beneath the `wsdl:message` element referenced by the `soap12:body` element. If 2 or more elements are coded, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.
- For details on the notes on specifying the `parts` attribute, see 20.*2(2) Coding the SOAP body and SOAP header and the referenced wsdl:part element*.

20.1.28 soap12:header element

This subsection describes the support range of the `soap12:header` element.

- You can code the `soap12:header` element as the child element of the `wsdl:input` and `wsdl:output` elements that are the grandchild elements of the `wsdl:binding` element. You can also omit this element.

- You cannot code child elements. If coded, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.
- You can specify the following attributes. If attributes other than those listed below are specified, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. However, if you specify the `namespace` attribute, a warning message (KDJW51009-W) is output to the standard error output and the log, and the processing of the `cjwsimport` command continues.
  - `message` attribute
  - `part` attribute
  - `use` attribute
- For details on the notes on specifying the `soap12:header` element, see *20.2(2) Coding the SOAP body and SOAP header and the referenced wsdl:part element*.

(1) message attribute (soap12:header element)

This point describes the support range of the `message` attribute included in the `soap12:header` element.

- You can code only one `message` attribute. This attribute cannot be omitted. If omitted, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. If 2 or more attributes are coded, a Cosminexus XML Processor error occurs.
- Use `Qname` to specify `wsdl:message` declared beneath the referable `wsdl:definitions` element.
- For details on the notes on specifying the `message` attribute, see *20.2(2) Coding the SOAP body and SOAP header and the referenced wsdl:part element*.

(2) part attribute (soap12:header element)

This point describes the support range of the `part` attribute included in the `soap12:header` element.

- You can code only one `part` attribute. This attribute cannot be omitted. If omitted, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. If 2 or more attributes are coded, a Cosminexus XML Processor error occurs.
- Specify the `wsdl:part` element declared beneath the `wsdl:message` element specified in the `message` attribute.
- For details on the notes on specifying the `part` attribute, see *20.2(2) Coding the SOAP body and SOAP header and the referenced wsdl:part element*.

(3) use attribute (soap12:header element)

This point describes the support range of the `use` attribute included in the `soap12:header` element.

- You can code only one `use` attribute. This attribute cannot be omitted. If omitted, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. If 2 or more attributes are coded, a Cosminexus XML Processor error occurs.
- You code `literal` as the value. If a value other than `literal` is coded, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.

20.1.29 soap12:fault element

This subsection describes the support range of the `soap12:fault` element.

- You can code only one `soap12:fault` element as the child element of the `wsdl:fault` element that is the grandchild element of the `wsdl:binding` element. If this element is omitted or if 2 or more elements are coded, an error message (KDJW51051-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.

- You cannot code child elements. If coded, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.
- You can specify the following attributes. If attributes other than those listed below are specified, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends:
  - `name attribute`
  - `use` attribute

(1) name attribute (soap12:fault element)

This point describes the support range of the `name` attribute included in the `soap12:fault` element.

- You can code only one `name` attribute. This attribute cannot be omitted. If omitted, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. If 2 or more attributes are coded, a Cosminexus XML Processor error occurs.
- Code the same value as the `name` attribute of the `wsdl:fault` element that is the parent element. If a different value is coded, a warning message (KDJW51027-W) is output to the standard error output and the log, and the processing of the `cjwsimport` command continues.
- For details on the values that can be specified, see *20.2(1) Values specifiable in the NCName type*.

(2) use attribute (soap12:fault element)

This point describes the support range of the `use` attribute included in the `soap12:fault` element.

- You can code 0 or 1 `use` attribute. If 2 or more attributes are coded, a Cosminexus XML Processor error occurs.
- You code `literal` as the value. If a value other than `literal` is coded, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.

20.1.30 soap12:address element

This subsection describes the support range of the `soap12:address` element.

- You can code only one `soap12:address` element as the child element of the `wsdl:port` element. This element cannot be omitted. If omitted or if 2 or more elements are coded, an error message (KDJW51175-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.
- You cannot code child elements. If coded, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.
- You can specify the `location` attribute. If you specify an attribute other than the `location` attribute, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.

(1) location attribute (soap12:address element)

This point describes the support range of the `location` attribute included in the `soap12:address` element.

- You can code only one `location` attribute. This attribute cannot be omitted. If omitted, an error message (KDJW51029-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. If 2 or more attributes are coded, a Cosminexus XML Processor error occurs.
- For details on the values that can be specified, see *20.2(3) Values specifiable in the location attribute of the soap:address element or soap12:address element*.

20.1.31 xsd:schema element

This subsection describes the support range of the `xsd:schema` element.

- Mapping to a Java type is performed according to the JAXB 2.2 specifications, but the entire mapping is delegated to Cosminexus XML Processor.

(1) xmime:expectedContentTypes attribute (xsd:schema element)

When `xsd:base64Binary` is specified in the `type` attribute of the `xsd:element` element, which is a WSDL schema declaration, you can explicitly specify the MIME type by using the `xmime:expectedContentTypes` attribute to associate the Base64 format data with a Java type corresponding to the MIME type. The following table describes whether you can code the `xmime:expectedContentTypes` attribute for the `xsd:element` element, which is a WSDL schema declaration.

Table 20-1: Codability of the xmime:expectedContentTypes attribute

No.	Parameters of wsdl:message that reference the xsd:element element	Codability of the xmime:expectedContentTypes attribute for the xsd:element element
1	`wsdl:input`	Y#1
2	`wsdl:output`	Y#1
3	`wsdl:fault`	N#2

Legend:

Y: Can be coded.

N: Cannot be coded.

#1

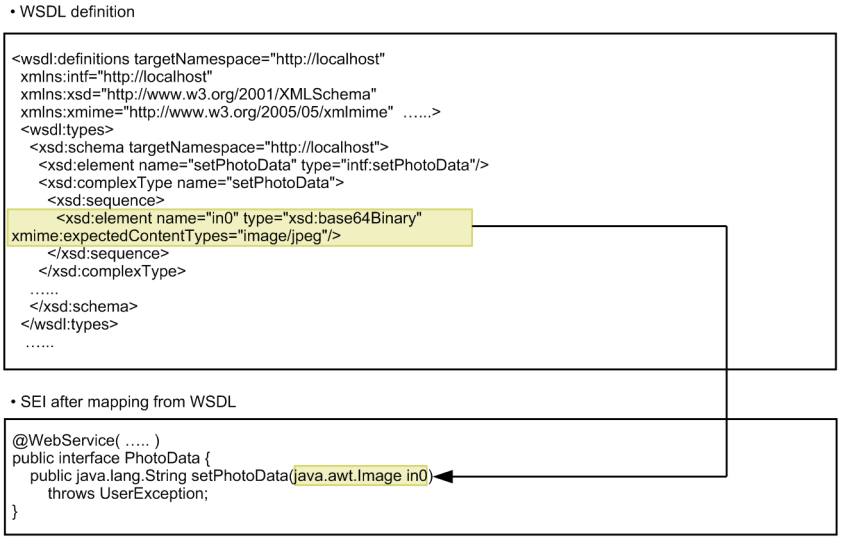
When the type of WSDL part is `inout`, specify the same MIME type as the value of the `xmime:expectedContentTypes` attribute of the `xsd:element` element referenced from the `wsdl:input` element, and as the value of the `xmime:expectedContentTypes` attribute of the `xsd:element` element referenced from the `wsdl:output` element. If a different MIME type is specified, the operations are not guaranteed.

#2

The operations are not guaranteed if you specify the `xsd:element` element in which the `xmime:expectedContentTypes` attribute is coded in the fault message.

To map WSDL to the Java type when the `xmime:expectedContentTypes` attribute is specified in the `xsd:element` element, map the `xsd:base64Binary` type with the `xmime:expectedContentTypes` attribute of the WSDL to the Java type. The following figure shows an example of mapping the WSDL to a Java type.

Figure 20-1: Example of mapping the WSDL to a Java type



Do not code parameters other than the `charset` parameter of `text/xml` and `application/xml` in the MIME type to be specified in the `xmime:expectedContentTypes` attribute. If a parameter other than the `charset` parameter of `text/xml` and `application/xml` are coded, the operations are not guaranteed.

The Java type to be mapped from WSDL changes depending on the MIME type specified in the `xmime:expectedContentTypes` attribute. The following table describes the relationship between the MIME types coded in the `xmime:expectedContentTypes` attribute and the associated Java types.

Table 20-2: Value of the xmime:expectedContentTypes attribute and the associated Java types

No.	Value of the xmime:expectedContentTypes attribute (MIME type)	Associated Java type
1	`application/xml`	`javax.xml.transform.Source`
2	`image/png`[#1]	`java.awt.Image`[#2]
3	`image/jpeg`[#1]	
4	Code the above MIME types delimited with commas (Example: `image/png, image/jpeg`)[#3]	
5	`image/*`[#3]	
6	`text/plain`	`java.lang.String`
7	`text/*`[#4]	`javax.activation.DataHandler`
8	`text/xml`[#5]	`javax.xml.transform.Source`
9	Other than above mentioned [#4, #6]	`javax.activation.DataHandler`

#1

To associate the `image` type that is not mentioned in the table with a Java type, specify `application/octet-stream` in the `xmime:expectedContentTypes` attribute and associate with the `javax.activation.DataHandler` class.

#2

Complies with the JAXB specifications. The `java.awt.Image` class is an abstract class that expresses the graphical image in the Java SE specifications and for which the data format is not defined. When the image data is instantiated by using this association, you can only store the decoded information in a concrete class instance. Therefore, when you send an image for which information can be reduced during encoding, such as the JPEG format, as an attachment, the instance at the receiving side might differ from the instance at the sending side and from the original data.

If you want to handle the image as is in the original format, use the MIME type (such as `application/octet-stream`) that is mapped to `javax.activation.DataHandler`.

#3

When similar types such as "`image/png, image/jpeg`" or "`image/*`" are specified as the MIME type, the `Content-Type` field value of the MIME header in the attachment part of the SOAP message to be sent is the initial value (`image/png`) when the `java.awt.Image` type is used.

#4

When `text/*` or a MIME type that is not mentioned in the table is specified as the MIME type, the `Content-Type` field value of the MIME header in the attachment part of the SOAP message to be sent is the initial value (MIME type of the `javax.activation.DataHandler` object) when the `javax.activation.DataHandler` type is used.

#5

When `text/xml` is specified as the MIME type, the `Content-Type` field value of the MIME header in the attachment part of the SOAP message to be sent is the initial value (`application/xml`) when the `javax.xml.transform.Source` type is used.

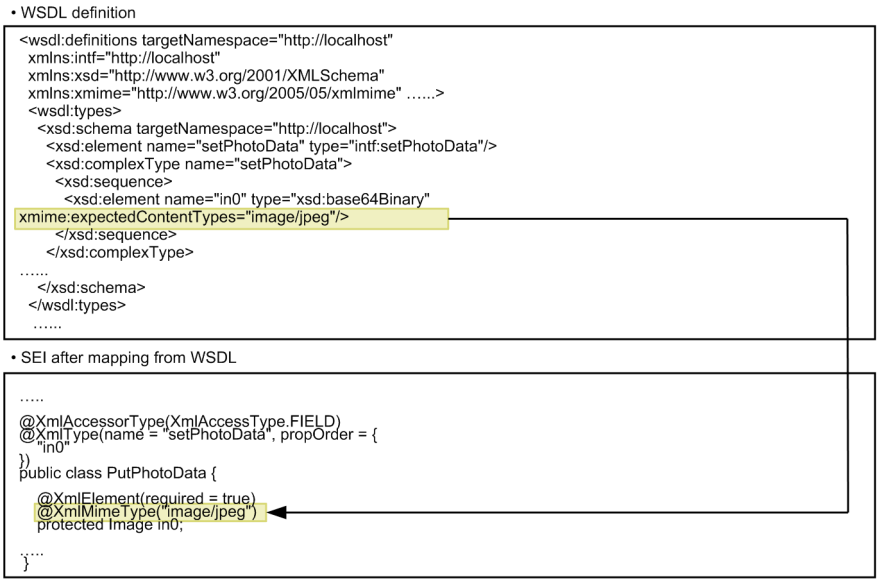
#6

This includes cases when a MIME type of a different type name is coded using commas as delimiters (example: `image/png, text/plain`).

The MIME type specified in the `xmime:expectedContentTypes` attribute is mapped to the value of the `javax.xml.bind.annotation.XmlMimeType` annotation that is annotated in the JavaBean class corresponding to the element coding the `xmime:expectedContentTypes` attribute, from among the JavaBean

classes that are automatically generated by the `cjwsimport` command. The following figure shows an example of mapping the automatically generated JavaBean class from the WSDL.

Figure 20-2: Mapping the automatically generated JavaBean class from the WSDL



(a) Importing the Namespace "xmime"

To map the WSDL to a Java type, you use the attribute `xmime:expectedContentTypes` existing in the Namespace `xmime`, but the Namespace `xmime` need not be imported using the `xsd:import` element with the JAX-WS.

When you use the WSDL created with the `cjwsgen` command, you must import the Namespace `xmime` as and when required. The following is an example of importing the Namespace `xmime`:

```
<wsdl:definitions targetNamespace="http://localhost"
 xmlns:intf="http://localhost"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xmime="http://www.w3.org/2005/05/xmlmime" ......>
 <wsdl:types>
 <xsd:schema targetNamespace="http://localhost">
  <xsd:import namespace="http://www.w3.org/2005/05/xmlmime"/>
  <xsd:element name="setPhotoData" type="intf:setPhotoData"/>
  <xsd:complexType name="setPhotoData">
  <xsd:sequence>
   <xsd:element name="in0" type="xsd:base64Binary" xmime:expectedContentTypes="image/
jpeg"/>
  </xsd:sequence>
  </xsd:complexType>
  ......
 </xsd:schema>
 </wsdl:types>
 ......
```

20.2 Notes on creating WSDL

This section describes the notes on creating WSDL.

(1) Values specifiable in the NCName type

With the Cosminexus JAX-WS functionality, as long as you do not violate the limitations for the `xsd:NCName` type in the XML Schema specifications, you can use one-byte alphanumeric characters (0 to 9, A to Z, a to z) and underscore (_). If you use characters other than one-byte alphanumeric characters and underscore, the operations are not guaranteed.

(2) Coding the SOAP body and SOAP header and the referenced wsdl:part element

This point describes the coding of the SOAP Body and SOAP Header that are coded as the child elements of the `wsdl:input` element and `wsdl:output` element, and the coding of the `wsdl:part` element referenced from the SOAP Body and SOAP Header.

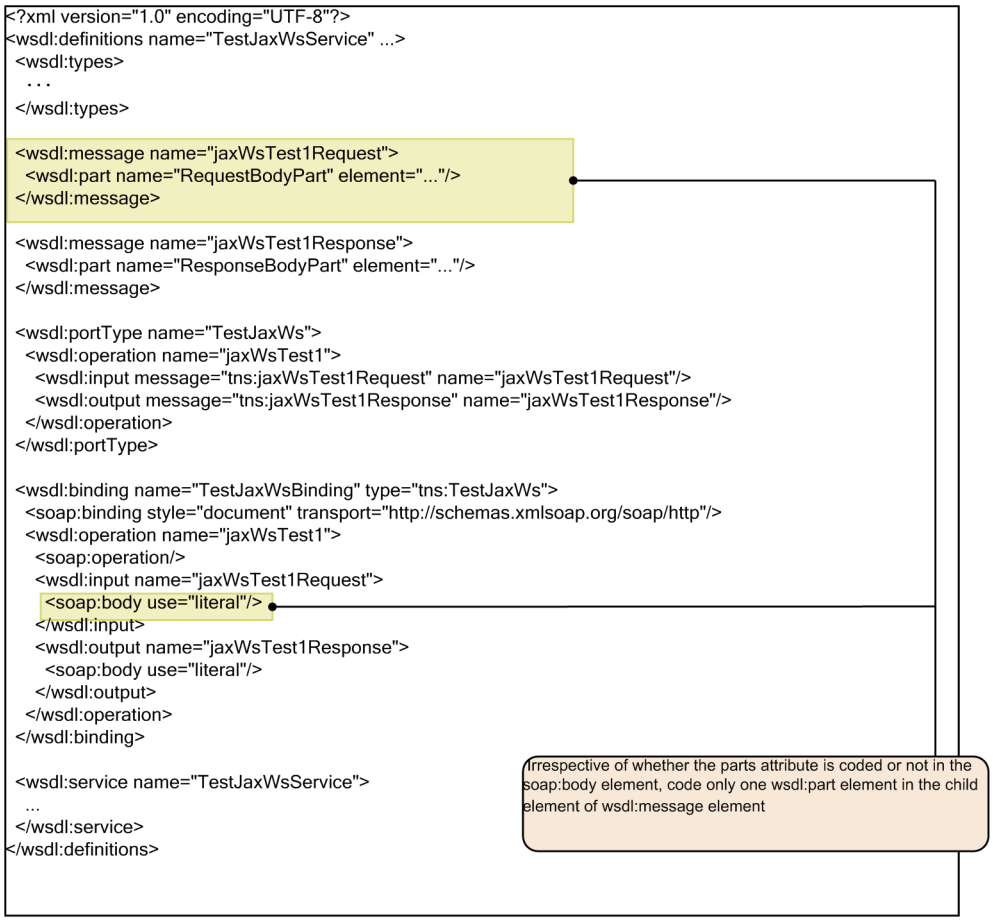
The description hereafter uses the SOAP 1.1 specifications for the examples. For the SOAP 1.2 specifications, substitute and read the Namespace and the element names, and the attribute values.

(a) When the SOAP Header is not defined

Whether you code the `parts` attribute in the `soap:body` element or not, code only one `wsdl:part` element in the child element of the `wsdl:message` element referenced from the `wsdl:input` element or `wsdl:output` element that are the parent elements.

The following figure shows an example of coding when the `soap:header` element is not coded.

Figure 20－3: Example of coding when the soap:header element is not coded



(b) When the SOAP Header is defined

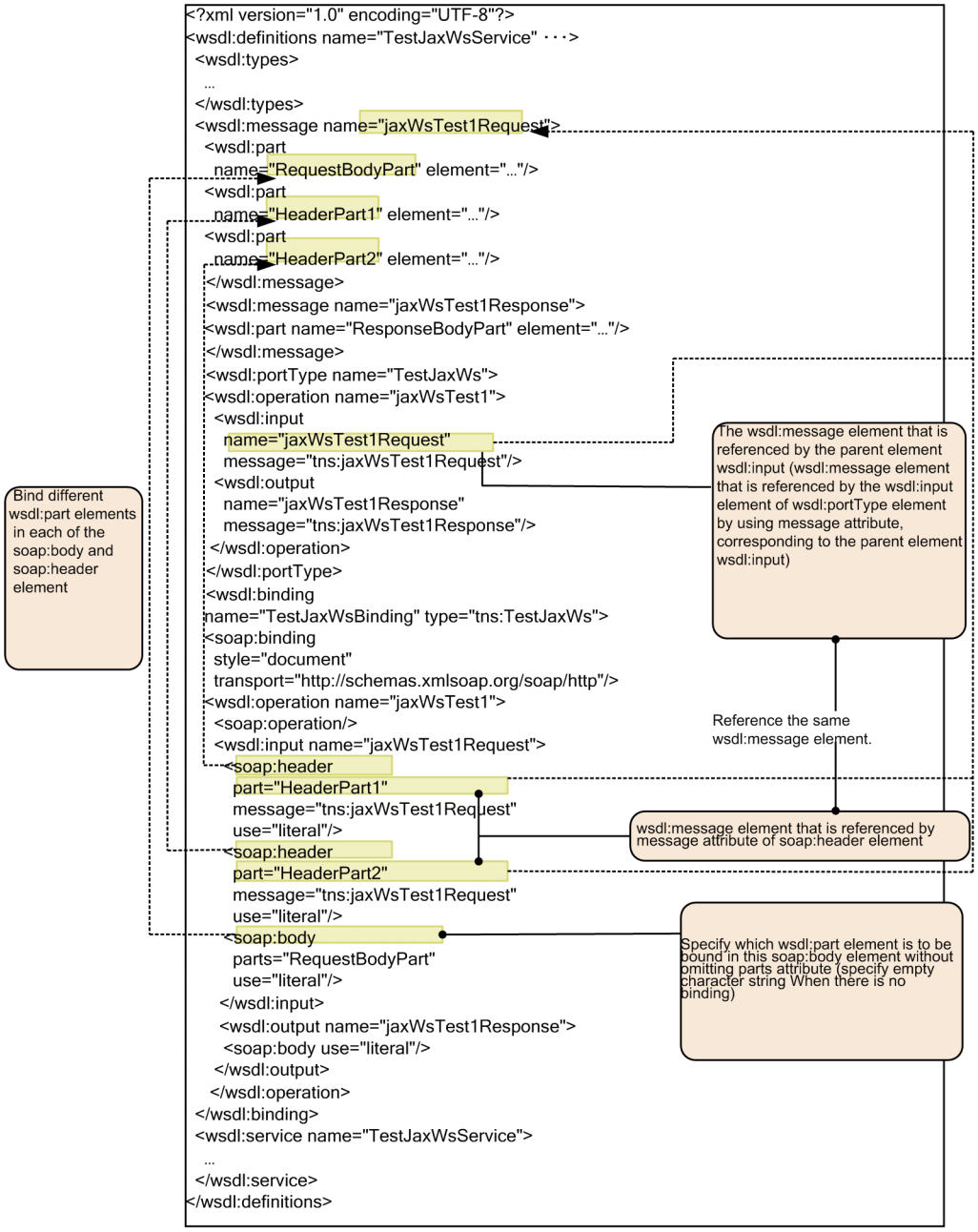
To code the `soap:header` element, define the coding according to the following contents:

- In the `message` attribute of the `soap:header` element, specify the `wsdl:message` element referenced from the `wsdl:input` element or `wsdl:output` element that are the parent elements.
- In the `part` attribute of the `soap:header` element, specify the `wsdl:part` element declared beneath the `wsdl:message` element specified in the `message` attribute. If an undeclared `wsdl:part` element is specified, an error message (KDJW51022-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends (KDJW51022-E).
- The `parts` attribute of the `soap:body` element cannot be omitted. Specify which of the multiple `wsdl:part` child elements of the `wsdl:message` element referenced from the `wsdl:input` element or `wsdl:output` element, which are the parent elements, will be bound to the `soap:body` element. If you do not want to bind the elements, specify an empty string.

  If the `wsdl:part` element of the `wsdl:message` elements that are not referenced or a non-existent `wsdl:part` element is specified in the `parts` attribute of the `soap:body` element, an error message (KDJW51021-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends. Also, if the `parts` attribute of the `soap:body` element is not specified, an error message (KDJW51179-E) is output to the standard error output and the log, and the processing of the `cjwsimport` command ends.

The following figure shows an example of coding when the `soap:header` element is coded.

Figure 20－4: Example of coding when the soap:header element is coded



(3) Values specifiable in the location attribute of the soap:address element or soap12:address element

You can specify a URL with the following format in the `location` attribute of the `soap:address` element or the `soap12:address` element:

- *Protocol*[#1]*://host-name*[#2]*/path-part*[#3]
  (Example) `http://hitachi.com/jaxws/service/UserInfoPort`
- *Protocol*[#1]*://host-name*[#2]*:port-number*[#4]*/path-part*[#3]
  (Example) `http://hitachi.com:80/jaxws/service/UserInfoPort`

#1

You can only specify http:// or https:// as the Namespace. You cannot specify other protocols. If you specify a protocol other than http:// and https:// is specified, the operations are not guaranteed.

#2

You can specify a string complying with the RFC2396 specifications. You can specify the string in the IPv4 and IPv6 format.

However, you cannot specify the following formats. If the following formats are specified, the operations are not guaranteed:

- Query string (Example) `http://example.com/?a=b`
- Anchor (Example) `http://example.com/index.html#anchor`
- Port number (Example) `http://example.com:8080/`
- User name/ Password (Example) `http://user:password@example.com`
- Percent-encoded characters (Example) `http://%E4%BD%BF%E7%94%A8`

#3

You can specify a string using only the numeric characters 0 to 9. If other characters are specified, the operations are not guaranteed.

#4

You can specify a string complying with the RFC2396 specifications. You can also specify percent-encoded characters ((Example) `http://%E4%BD%BF%E7%94%A8`).

21 Support Range of XML Catalogs 1.1

This chapter describes the support range of the XML Catalogs 1.1 specifications.

21.1 Support range list of the XML Catalogs 1.1 specifications

The following table describes the support range of the catalog file syntax, defined in the XML Catalogs 1.1 specifications. For the codes created by using the syntax that is not supported by the JAX-WS functionality of Application Server, the operation is not guaranteed.

Table 21－1: List of syntax (elements) supported in the catalog file

Item no.	Element	Supported
1	er:catalog	Y
2	er:group	N
3	er:public	Y
4	er:system	Y
5	er:rewriteSystem	N
6	er:systemSuffix	N
7	er:delegatePublic	N
8	er:delegateSystem	N
9	er:uri	N
10	er:rewriteURI	N
11	er:uriSuffix	N
12	er:delegateURI	N
13	er:nextCatalog	N

Legend:

Y: Supported in the JAX-WS functionality of Application Server.

N: Not supported in the JAX-WS functionality of Application Server.

Table 21－2: List of syntax (attributes) supported in the catalog file

Item no.	Element	Attribute	Supported
1	er:catalog	prefer	Y
		id	N
		xml:base	N
2	er:public	publicId	Y
		uri	Y
		id	N
		xml:base	N
3	er:system	systemId	Y
		uri	Y
		id	N
		xml:base	N

Legend:

Y: Supported in the JAX-WS functionality of Application Server.

N: Not supported in the JAX-WS functionality of Application Server.

21.2 Details of the Support Range of the XML Catalogs 1.1 Specifications

This section describes the support range of each syntax (element) in the catalog file, defined in the XML Catalogs 1.1 specifications.

21.2.1 er:catalog element

- You can code only one `er:catalog` element as the root element of a catalog file. You cannot omit this element. If omitted or coded two or more times, a warning message# is displayed, the catalog functionality is disabled, and the processing continues.
- You can specify the following elements as the child elements. If you code elements other than the following elements, a warning message# is displayed.
  - `er:public` element
  - `er:system` element

#

Depending on where you use the catalog functionality, the warning message and the output destination vary as follows:

- When starting a Web Services client: log file (KDJW30023-W)
- When developing a Web Services client: standard error output (KDJW51221-W)

(1) `prefer` attribute

- You can omit the `prefer` attribute or code one `prefer` attribute for the `er:catalog` element. If you omit this attribute, operation is the same as when the `public` attribute is specified.
- You can specify the following values as the attribute values. If a value other than the following values is specified, a warning message# is displayed.
  - `public`
  - `system`
- If the WSDL or the `xsd:import` element of the XML schema contains the `namespace` and `schemalocation` attributes; you cannot map the `namespace` attribute of the `xsd:import` element by using the `er:public` element, even if you specify the `prefer` attribute as `public`.

#

Depending on where you use the catalog functionality, the warning message and the output destination vary as follows:

- When starting a Web Services client: log file (KDJW30023-W)
- When developing a Web Services client: standard error output (KDJW51221-W)

21.2.2 er:public element

- This element maps the name space URI of an XML Schema specified in the `publicId` attribute to the URI pointing to the location of XML Schema specified in the `uri` attribute.
- You can omit a child element or code a maximum of 255 elements as the child elements of the `er:catalog` element. If more than 255 elements are specified, the operation is not guaranteed.

(1) publicId attribute

- You can code only one `publicId` as the attribute of the `er:public` element. You cannot omit this attribute. If omitted, a warning message[#] is displayed, the catalog functionality is disabled, and the processing continues.
- Specify the same namespace URI as that of the `namespace` attribute of the `xsd:import` element to the `publicId` attribute.

#

Depending on where you use the catalog functionality, the warning message and the output destination vary as follows:

- When starting a Web Services client: log file (KDJW30023-W)
- When developing a Web Services client: standard error output (KDJW51221-W)

(2) uri attribute

- You can code only one `uri` as the attribute of the `er:public` element. You cannot omit this attribute. If omitted, a warning message[#1] is displayed, the catalog functionality is disabled, and the processing continues.
- The conditions for the attribute values that you can specify are as follows:
  - Specify the relative path[#2] or the URL of the XML Schema to be mapped. When you specify the value, match the target namespace of the XML Schema to be mapped with the namespace of the `namespace` attribute of the `xsd:import` element. If the namespaces do not match, the operation is not guaranteed.
  - Specify the attribute by using the characters complying with `xsd:anyURI` stated in RFC 2396. Note that you cannot use RFC 2732 (IPv6).
  - The value is not case sensitive.
  - You can specify a character string of any length. Note that an error occurs if the length exceeds the limit of the OS.

Note:

If you specify a WSDL or an XML schema that does not exist or requires no access permission, a warning message (KDJW30024-W) is displayed, the catalog functionality is disabled, and the processing continues.

#1

Depending on where you use the catalog functionality, the warning message and the output destination vary as follows:

- When starting a Web Services client: log file (KDJW30023-W)
- When developing a Web Services client: standard error output (KDJW51221-W)

#2

Indicates a relative path from the directory that stores the catalog file.

21.2.3 er:system element

- This element maps the URI that is pointing to the location of a WSDL or an XML Schema specified in the `systemId` attribute to a different URI that is pointing to the location of a WSDL or an XML Schema.
- You can omit a child element or code up to 255 elements as the child elements of the `er:catalog` element. If more than 255 elements are specified, the operation is not guaranteed.

(1) systemId attribute

- You can code only one `systemId` attribute as the attribute of the `er:system` element. You cannot omit this attribute. If omitted, a warning message[#] is displayed, the catalog functionality is disabled, and the processing continues.
- The conditions for the attribute values that can be specified are as follows:

- Specify the absolute URL of the WSDL or the XML schema to be mapped. You cannot specify a relative URL or a relative path. Convert a relative URL or a relative path to an absolute URL to specify the attribute value.
- Specify the attribute by using the characters complying with `xsd:anyURI` specified in RFC 2396. Note that you cannot use RFC 2732 (IPv6).
- The value is not case sensitive.
- You can specify a character string of any length. Note that an error occurs if the length exceeds the limit of the OS.

#

Depending on where you use the catalog functionality, the warning message and the output destination vary as follows:

- When executing a Web Services client: log file (KDJW30023-W)
- When developing a Web Services client: standard error output (KDJW51221-W)

(2) uri attribute

- You can code only one `uri` attribute as the attribute of the `er:public` element. You cannot omit this attribute. If you omit, a warning message[#1] is displayed, the catalog functionality is disabled, and the processing continues.
- The conditions for the attribute values that you can specify are as follows:
  - Specify the relative path[#2] or URL of the WSDL or the XML schema to be mapped.
  - Specify the attribute by using the characters complying with `xsd:anyURI` specified in RFC 2396. Note that you cannot use RFC 2732 (IPv6).
  - The value is not case sensitive.
  - You can specify a character string of any length. Note that an error occurs if the length exceeds the limit of the OS.

Note:

- If you specify a WSDL or an XML Schema that does not exist or requires no access permission, a warning message (KDJW30024-W) is displayed, the catalog functionality is disabled, and the processing continues.

#1

Depending on where you use the catalog functionality, the warning message and the output destination vary as follows:

- When executing a Web Services client: log file (KDJW30023-W)
- When developing a Web Services client: standard error output (KDJW51221-W)

#2

Indicates a relative path from the directory that stores the catalog file.

22 Support Range of the SAAJ Specifications

This chapter describes the support range for the SAAJ specifications that you must keep in mind when you develop a Web Service.

22.1 Support range of the SAAJ 1.3 specifications

This section describes the support range of the interfaces and classes in the SAAJ 1.3 specifications. This section also describes the notes on using the interfaces and classes in the dispatch-based Web Service client.

The following table describes the support range of the interfaces in the SAAJ 1.3 specifications. For details on the interfaces, see the *JDK documentation*.

Table 22-1: Support range of the interfaces in the SAAJ 1.3 specifications

No.	Interface name	Method name/ Field name	Support
1	`Detail`	`addDetailEntry(Name name)`	Y
2		`addDetailEntry(QName qname)`	Y
3		Other methods	Y
4	`DetailEntry`	No method	Y
5	`Name`	All methods	Y
6	`Node`	`getValue()`	Y
7		`recycleNode()`	Y
8		`setParentElement(SOAPElement parent)`	Y
9		`setValue(String value)`	Y
10		Other methods	Y
11	`SOAPBody`	`addBodyElement(Name name)`	Y
12		`addBodyElement(QName qname)`	Y
13		`addDocument(Document document)`	Y
14		`addFault(Name faultCode, String faultString, Locale locale)`	Y
15		`addFault(Name faultCode, String faultString)`	Y
16		`addFault(QName faultCode, String faultString, Locale locale)`	Y
17		`addFault(QName faultCode, String faultString)`	Y
18		Other methods	Y
19	`SOAPBodyElement`	No method	Y
20	`SOAPConstants`	All fields	Y
21	`SOAPElement`	`addAttribute(Name name, String value)`	Y
22		`addAttribute(QName qname, String value)`	Y
23		`addChildElement(Name name)`	Y
24		`addChildElement(SOAPElement element)`	Y
25		`addChildElement(String localName)`	Y
26		`addChildElement(String localName, String prefix)`	Y
27		`addChildElement(String localName, String prefix, String uri)`	Y
28		`addChildElement(QName qname)`	Y

No.	Interface name	Method name/ Field name	Support
29	SOAPElement	addNamespaceDeclaration(String prefix, String uri)	Y
30		addTextNode(String text)	Y
31		createQName(String localName, String prefix)	Y
32		getAttributeValue(Name name)	Y
33		getAttributeValue(QName qname)	Y
34		getChildElements(Name name)	Y
35		getChildElements(QName qname)	Y
36		getEncodingStyle()	Y
37		getNamespacePrefixes()	Y
38		getNamespaceURI(String prefix)	Y
39		removeAttribute(Name name)	Y
40		removeAttribute(QName qname)	Y
41		setElementQName(QName newName)	Y
42		Other methods	Y
43	SOAPEnvelope	createName(String localName)	Y
44		createName(String localName, String prefix, String uri)	Y
45		Other methods	Y
46	SOAPFault	getFaultCode()	Y
47		getFaultCodeAsName()	Y
48		getFaultCodeAsQName()	Y
49		getFaultString()	Y
50		setFaultCode(Name faultCodeQName)	Y
51		setFaultCode(QName faultCodeQName)	Y
52		setFaultCode(String faultCode)	Y
53		setFaultString(String faultString)	Y
54		setFaultString(String faultString, Locale locale)	Y
55		addFaultReasonText(String text, Locale locale)	Y
56		getFaultReasonLocales()	Y
57		getFaultReasonText(Locale locale)	Y
58		getFaultReasonTexts()	Y
59		getFaultStringLocale()	Y
60		setFaultRole(String uri)	Y
61		Other methods	Y
62	SOAPFaultElement	No method	Y

No.	Interface name	Method name/ Field name	Support
63	SOAPHeader	addHeaderElement(Name name)	Y
64		addHeaderElement(QName qname)	Y
65		addUpgradeHeaderElement(String supportedSoapUri)	Y
66		examineHeaderElements(String actor)	Y
67		examineMustUnderstandHeaderElements(String actor)	Y
68		extractHeaderElements(String actor)	Y
69		Other methods	Y
70	SOAPHeaderElement	setActor(String actorURI)	Y
71		setRole(String uri)	Y
72		Other methods	Y
73	Text	All methods	Y

Legend:
Y: Supported in the Cosminexus JAX-WS functionality.

The following table describes the support range of the classes in the SAAJ 1.3 specifications. For details on the classes, see the *JDK documentation*.

Table 22-2: Support range of the classes in the SAAJ 1.3 specifications

No.	Class name	Method name/ Field name	Support
1	AttachmentPart	addMimeHeader(String name, String value)	Y
2		getAllMimeHeaders()	Y
3		getContentLocation()	N
4		setContentLocation(String contentLocation)	N
5		setBase64Content(InputStream content, String contentType)	Y
6		setContent(Object object, String contentType)	Y
7		setContentId(String contentId)	Y
8		setContentType(String contentType)	Y
9		setMimeHeader(String name, String value)	Y
10		setRawContent(InputStream content, String contentType)	Y
11		setRawContentBytes(byte[] content, int offset, int len, String contentType)	Y
12		Other methods	Y
13	MessageFactory	newInstance(String protocol)	Y
14		Other methods	Y
15	MimeHeader	MimeHeader(String name, String value) constructor	Y
16		Other methods	Y

No.	Class name	Method name/ Field name	Support
17	MimeHeaders	addHeader(String name, String value)	Y
18		setHeader(String name, String value)	Y
19		Other methods	Y
20	SAAJMetaFactory	All methods	Y
21	SAAJResult	SAAJResult(SOAPMessage message) constructor	Y
22		SAAJResult(SOAPElement rootNode) constructor	Y
23		Other methods	Y
24	SOAPConnection	All methods	Y
25	SOAPConnectionFactory	All methods	Y
26	SOAPElementFactory#	All methods	N
27	SOAPFactory	newInstance(String protocol)	Y
28		createElement(Element domElement)	Y
29		createElement(String localName, String prefix, String uri)	Y
30		createFault(String reasonText, QName faultCode)	Y
31		createName(String localName)	Y
32		createName(String localName, String prefix, String uri)	Y
33		Other methods	Y
34	SOAPMessage	addAttachmentPart(AttachmentPart AttachmentPart)	Y
35		createAttachmentPart(Object content, String contentType)	Y
36		getAttachment(SOAPElement element)	Y
37		getAttachments(MimeHeaders headers)	N
38		getProperty(String property)	Y
39		removeAttachments(MimeHeaders headers)	Y
40		setContentDescription(String description)	Y
41		setProperty(String property, Object value)	Y
42		writeTo(OutputStream out)	Y
43		Other methods	Y
44	SOAPPart	addMimeHeader(String name, String value)	Y
45		getContentId()	N
46		getContentLocation()	N
47		getMimeHeader(String name)	Y
48		setContent(Source source)	Y
49		setContentId(String contentId)	N

No.	Class name	Method name/ Field name	Support
50	SOAPPart	setContentLocation(String contentLocation)	N
51		setMimeHeader(String name, String value)	Y
52		Other methods	Y

Legend:

Y: Supported in the Cosminexus JAX-WS functionality.

N: Not supported in the Cosminexus JAX-WS functionality.

#

This class is not recommended, and therefore, if you use this class, the operations are not guaranteed.

22.1.1 Detail interface

This subsection describes the notes on using the methods of the Detail interface.

- Do not specify null in the arguments of the addDetailEntry(Name name) and addDetailEntry(QName qname) methods. If null is specified, the operations are not guaranteed.

22.1.2 Node interface

This subsection describes the notes on using the methods of the Node interface.

- You cannot obtain the child node value of the target node of the getValue() method. To obtain the child node value of the target node, issue the method for the child node.
- You can invoke the recycleNode() method although detachNode() is not invoked for the target node. If you invoke the method, the executed operations are the same as those for detachNode().
- With the setParentElement(SOAPElement parent) method, you cannot set the nodes belonging to different DOM Documents in a parent-child relationship.
- If you specify null in the arguments of the setValue(String value) method, null is set up as the value of the target node.

22.1.3 SOAPBody interface

This subsection describes the notes on using the methods of the SOAPBody interface.

- Do not specify null in the arguments of the following methods. If null is specified, the operations are not guaranteed.
  - addBodyElement(Name name)
  - addBodyElement(QName qname)
  - addDocument(Document document)
  - addFault(Name faultCode, String faultString, Locale locale)
  - addFault(Name faultCode, String faultString)
  - addFault(QName faultCode, String faultString, Locale locale)
  - addFault(QName faultCode, String faultString)
- Do not specify null in the locale argument of the addFault(Name faultCode, String faultString, Locale locale) and addFault(QName faultCode, String faultString, Locale locale) methods. If specified, the operations are not guaranteed.
- Specify the fault code defined in the standard specifications in the arguments of the addFault(QName faultCode, String faultString, Locale locale) and addFault(QName faultCode,

`String faultString)` methods. If you specify a fault code that is not defined in the standard specifications, the operations are not guaranteed.

- If a null character is set up in the `faultString` argument of the `addFault(QName faultCode, String faultString, Locale locale)` or `addFault(QName faultCode, String faultString)` methods and a SOAP fault is sent, `null` is obtained when the `getFaultString()` and `getFaultReasonTexts().next()` methods are issued for the received SOAP fault.

22.1.4 SOAPElement interface

This subsection describes the notes on using the methods of the `SOAPElement` interface.

- Do not specify `null` in the `name` argument of the `addAttribute(Name name, String value)` method. If specified, the operations are not guaranteed.
- Do not specify `null` in the `qname` argument of the `addAttribute(QName qname, String value)` method. If specified, the operations are not guaranteed.
- Do not specify `null` in the arguments of the following methods. If `null` is specified, the operations are not guaranteed.
  - `addChildElement(Name name)`
  - `addChildElement(SOAPElement element)`
  - `addChildElement(QName qname)`
  - `addTextNode(String text)`
  - `getAttributeValue(Name name)`
  - `getAttributeValue(QName qname)`
  - `getChildElements(Name name)`
  - `getChildElements(QName qname)`
  - `removeAttribute(Name name)`
  - `removeAttribute(QName qname)`
  - `setElementQName(QName newName)`
- Do not specify `null` or null characters in the `localName` argument of the following methods. If `null` or null characters are specified, the operations are not guaranteed.
  - `addChildElement(String localName)`
  - `addChildElement(String localName, String prefix)`
  - `addChildElement(String localName, String prefix, String uri)`
- Do not specify `null` or null characters in the `uri` argument of the `addChildElement(String localName, String prefix, String uri)` method. If specified, the operations are not guaranteed.
- Do not specify `null` in the `prefix` argument of the `addNamespaceDeclaration(String prefix, String uri)` method. If specified, the operations are not guaranteed.
- If `null` or null characters are specified in the `uri` argument of the `addNamespaceDeclaration(String prefix, String uri)` method, a Namespace declaration is added with a `null` Namespace URI.
- Do not specify `null` in the `localName` argument of the `createQName(String localName, String prefix)` method. If specified, the operations are not guaranteed.
- If the `getEncodingStyle()` method is issued when the encoding style is not specified, `null` is returned.
- If the `getNamespacePrefixes()` method is issued for the target `SOAPElement` belonging to the default Namespace (`xmlns=""`), the return value does not include the default Namespace prefix.
- If the prefix specified in the argument for the `getNamespaceURI(String prefix)` method is not declared in the target `SOAPElement`, or if `null` or null characters are specified in the argument, `null` is returned.

22.1.5 SOAPEnvelope interface

This subsection describes the notes on using the methods of the `SOAPEnvelope` interface.

- Do not specify `null` or null characters in the `localName` argument of the `createName(String localName)` and `createName(String localName, String prefix, String uri)` methods. If specified, the operations are not guaranteed.
- If `null` or null characters are specified in the `uri` argument of the `createName(String localName, String prefix, String uri)` method, a `Name` object is generated with a `null` Namespace URI.

22.1.6 SOAPFault interface

This subsection describes the notes on using the methods of the `SOAPFault` interface.

- If the following methods are issued for the `SOAPFault` objects for which the fault codes are not explicitly specified, the value that is automatically set up by the Cosminexus JAX-WS functionality is returned:
  - `getFaultCode()`
  - `getFaultCodeAsName()`
  - `getFaultCodeAsQName()`
- If the following methods are issued for the `SOAPFault` objects for which the fault codes are not explicitly specified, the fault string "Fault string, and possibly fault code, not set" that is automatically set up by the Cosminexus JAX-WS functionality is returned:
  - `getFaultString()`
  - `getFaultReasonTexts()`
- In the SOAP 1.1 format, if the `getFaultStringLocale()` method is issued for the `SOAPFault` objects for which the fault codes are not explicitly specified, `null` is returned.
- In the SOAP 1.2 format, if the `getFaultReasonLocales()` method or the `getFaultStringLocale()` method is issued for the `SOAPFault` objects for which the fault codes are not explicitly specified, the value that is automatically set up by the Cosminexus JAX-WS functionality is returned.
- Do not specify `null` in the arguments of the following methods. If `null` is specified, the operations are not guaranteed.
  - `setFaultCode(Name faultCodeQName)`
  - `setFaultCode(QName faultCodeQName)`
  - `setFaultCode(String faultCode)`
  - `setFaultString(String faultString)`
  - `setFaultString(String faultString, Locale locale)`
  - `setFaultRole(String uri)`
- Specify a Namespace-qualified fault code in the arguments of the following methods. If a Namespace-unqualified fault code is specified with the local name only, the operations are not guaranteed.
  - `setFaultCode(Name faultCodeQName)`
  - `setFaultCode(QName faultCodeQName)`
  - `setFaultCode(String faultCode)`
- Do not specify `null` in the `locale` argument of the `setFaultString(String faultString, Locale locale)` method. If specified, the operations are not guaranteed.
- Do not specify `null` in the `text` argument of the `addFaultReasonText(String text, Locale locale)` method. If specified, the operations are not guaranteed.
- If `null` is specified in the arguments of the `getFaultReasonText(Locale locale)` method, `null` is returned.

- The locale obtained by the `getFaultReasonLocales()` method or the `getFaultStringLocale()` method might differ from the locale set up in the `xml:lang` attribute of the received SOAP fault.
- Specify a string with the URI format in the `uri` argument of the `setFaultRole(String uri)` method. If a string with a non-URI format is specified, the operations are not guaranteed.

22.1.7 SOAPHeader interface

This subsection describes the notes on using the methods of the `SOAPHeader` interface.

- Do not specify `null` in the arguments of the `addHeaderElement(Name name)` and `addHeaderElement(QName qname)` methods. If `null` is specified, the operations are not guaranteed.
- Do not specify `null` in the `supportedSoapUri` argument of the `addUpgradeHeaderElement(String supportedSoapUri)` method. If specified, the operations are not guaranteed.
- Do not specify `null` in the `actor` argument of the `examineHeaderElements(String actor)` and `extractHeaderElements(String actor)` methods. If specified, the operations are not guaranteed.
- If `null` is specified in the `actor` argument of the `examineMustUnderstandHeaderElements(String actor)` method, `Iterator` containing all the set up `SOAPHeaderElement` is returned.

22.1.8 SOAPHeaderElement interface

This subsection describes the notes on using the methods of the `SOAPHeaderElement` interface.

- If `null` is specified in the `actorURI` argument of the `setActor(String actorURI)` method, a null `actor` attribute is set up as the value in the SOAP 1.1 format, and a null `role` attribute is set up as the value in the SOAP 1.2 format.
- In the SOAP 1.2 format, if `null` is specified in the `uri` argument of the `setRole(String uri)` method, a `role` attribute is set up with a `null` value.

22.1.9 AttachmentPart class

This subsection describes the notes on using the methods of the `AttachmentPart` class.

- Even if the MIME header is set up using the `addMimeHeader(String name, String value)` or `setMimeHeader(String name, String value)` methods, the MIME header does not appear on the sent or received SOAP messages.
- The value specified in the `value` argument of the `addMimeHeader(String name, String value)` or `setMimeHeader(String name, String value)` methods is set up as the MIME header value.
- The `Content-Transfer-Encoding` header cannot be obtained with the `getAllMimeHeaders()` method. With the Cosminexus JAX-WS functionality, an attachment is always sent in the binary format, and therefore, the `Content-Transfer-Encoding` header value of `AttachmentPart` becomes binary.
- When the `getContentLocation()` method is used, the operations are not guaranteed.
- When the `setContentLocation(String contentLocation)` method is used, the operations are not guaranteed. The `Content-Id` header set up with the `AttachmentPart#setContentId` method must be used instead of the `Content-Location` header.
- The value specified in the `contentType` argument of the following methods is set up in the `Content-Type` header value. Therefore, you must specify the MIME type suitable to the attachment type. If an invalid MIME type is specified, the operations are not guaranteed.
  - `setBase64Content(InputStream content, String contentType)`
  - `setContentType(String contentType)`
  - `setRawContent(InputStream content, String contentType)`

  - `setRawContentBytes(byte[] content, int offset, int len, String contentType)`
- Do not specify `null` or a MIME type that is not defined in the standard specifications in the `contentType` argument of the `setContent(Object object, String contentType)` method. If specified, the operations are not guaranteed.
- In the first argument of the `setContent(Object object, String contentType)` method, specify an object suitable to the MIME type specified in the second argument. If an unsuitable object is specified, the operations are not guaranteed. Also, if `null` is specified, the operations are not guaranteed.
- If `null` or null characters are specified in the `contentId` argument of the `setContentId(String contentId)` method that value is set up in the `Content-Id` header value.
- Specify the correct offset in the `offset` argument and the correct size in the `len` argument of the `setRawContentBytes(byte[] content, int offset, int len, String contentType)` method. If an invalid value is specified, the operations are not guaranteed.

22.1.10 MessageFactory class

This subsection describes the notes on using the methods of the `MessageFactory` class.

- Do not specify `DYNAMIC_SOAP_PROTOCOL` in the `newInstance(String protocol)` method. If specified, the operations are not guaranteed.

22.1.11 MimeHeader class

This subsection describes the notes on using the methods of the `MimeHeader` class.

- Do not specify characters defined as unavailable with the MIME headers, such as RFC 822 and RFC 2045, in the arguments of the `MimeHeader(String name, String value)` constructor. If specified, the operations are not guaranteed.

22.1.12 MimeHeaders class

This subsection describes the notes on using the methods of the `MimeHeaders` class.

- Do not specify `null` in the `value` argument of the `addHeader(String name, String value)` and `setHeader(String name, String value)` methods. If specified, the operations are not guaranteed.
- Even if a value is set up in the `Content-Length` header or `Content-Type` header using the `addHeader(String name, String value)` or `setHeader(String name, String value)` methods, the value is overwritten when the message is sent or received.

22.1.13 SAAJResult class

This subsection describes the notes on using the methods of the `SAAJResult` class.

- Do not specify `null` in the arguments of the `SAAJResult(SOAPMessage message)` constructor. If `null` is specified, the operations are not guaranteed.
- Do not specify `null` in the `rootNode` argument of the `SAAJResult(SOAPElement rootNode)` constructor. If specified, the operations are not guaranteed.

22.1.14 SOAPFactory class

This subsection describes the notes on using the methods of the `SOAPFactory` class.

- Do not specify `DYNAMIC_SOAP_PROTOCOL` in the `newInstance(String protocol)` method. If specified, the operations are not guaranteed.
- If `null` is specified in the arguments of the `createElement(Element domElement)` method, `null` is returned.
- Do not specify a null character in the `localName` argument of the `createElement(String localName, String prefix, String uri)` method. If specified, the operations are not guaranteed.
- If `null` or a null character is specified in the `prefix` argument of the `createElement(String localName, String prefix, String uri)` method, a `SOAPElement` object is generated with a `null` prefix.
- Do not specify `null` in the `uri` argument of the `createElement(String localName, String prefix, String uri)` method. If specified, the operations are not guaranteed. Also, if `null` is specified in the `uri` argument, a `SOAPElement` object is generated with a `null` Namespace URI.
- Do not specify `null` in the arguments of the `createFault(String reasonText, QName faultCode)` method. If `null` is specified, the operations are not guaranteed.
- Do not specify a fault code that is not defined in the standard specifications in the `faultCode` argument of the `createFault(String reasonText, QName faultCode)` method. If specified, the operations are not guaranteed.
- Do not specify `null` in the `localName` argument of the `createName(String localName)` or `createName(String localName, String prefix, String uri)` methods. If specified, the operations are not guaranteed.
- If `null` or null characters are specified in the `uri` argument of the `createName(String localName, String prefix, String uri)` method, a `Name` object is generated with a `null` Namespace URI.

22.1.15 SOAPMessage class

This subsection describes the notes on using the methods of the `SOAPMessage` class.

- Do not specify `null` in the arguments of the following methods. If `null` is specified, the operations are not guaranteed.
  - `addAttachmentPart(AttachmentPart AttachmentPart)`
  - `getAttachment(SOAPElement element)`
  - `writeTo(OutputStream out)`
- Do not specify a blank `AttachmentPart` object in the `AttachmentPart` argument of the `addAttachmentPart(AttachmentPart AttachmentPart)` method. If specified, the operations are not guaranteed.
- In the first argument of the `createAttachmentPart(Object content, String contentType)` method, specify an object suitable to the MIME type specified in the second argument. If an unsuitable object is specified, the operations are not guaranteed. Also, if `null` is specified, the operations are not guaranteed.
- To reference `AttachmentPart` from the element value and the `href` attribute specified in the arguments of the `getAttachment(SOAPElement element)` method, code the CID URL schema (RFC 2392 rule) indicating the existing `AttachmentPart`. If the CID URL schema indicating a non-existent `AttachmentPart` is coded, the operations are not guaranteed.
- When the `getAttachments(MimeHeaders headers)` method is used, the operations are not guaranteed. You must obtain `AttachmentPart` with the `SOAPMessage#getAttachments()` or `getAttachment(SOAPElement)` methods.
- If `null` is specified in the arguments of the `getProperty(String property)` method, `null` is returned.
- The Cosminexus JAX-WS functionality only supports utf-8 as the character encoding for the SOAP messages. However, if the property value is acquired by specifying `SOAPMessage.CHARACTER_SET_ENCODING` in the arguments of the `getProperty(String property)` method, `null` might be returned.
- If the target property is not set up with the `SOAPMessage#setProperty` method, `null` is returned in the `getProperty(String property)` method.

- If `null` is specified in the arguments of the `removeAttachments(MimeHeaders headers)` method, all `AttachmentPart` are deleted.
- If `null` is specified in the arguments of the `setContentDescription(String description)` method, the value is not set up in the `Content-Description` header. If `null` is specified, `null` is set up in the `Content-Description` header.
- Specify `SOAPMessage.CHARACTER_SET_ENCODING` or `SOAPMessage.WRITE_XML_DECLARATION` in the `property` argument of the `setProperty(String property, Object value)` method. Other properties, even if specified, are ignored.
- When you specify `SOAPMessage.CHARACTER_SET_ENCODING` in the `property` argument of the `setProperty(String property, Object value)` method, specify utf-8 in the `value` argument. If a value other than utf-8 is specified, the operations are not guaranteed.
- When you specify `SOAPMessage.WRITE_XML_DECLARATION` in the `property` argument of the `setProperty(String property, Object value)` method, specify "true" or "false". If a value other than "true" or "false" is specified, the operations are not guaranteed.
- Do not specify `null` in the `property` argument of the `setProperty(String property, Object value)` method. If specified, the operations are not guaranteed.
- When a SOAP message is sent and received using Dispatch/ Provider, the properties cannot be set with the `setProperty(String property, Object value)` method. If you want to set up the properties, send and receive the SOAP message with `SOAPConnection`.

22.1.16 SOAPPart class

This subsection describes the notes on using the methods of the `SOAPPart` class.

- The value specified in the `value` argument of the `addMimeHeader(String name, String value)` or `setMimeHeader(String name, String value)` methods is set up in the MIME header value.
- Even if the MIME header is set up in `SOAPPart` using the `addMimeHeader(String name, String value)` or `setMimeHeader(String name, String value)` methods, the MIME header does not appear on the sent and received SOAP message.
- If a MIME header name that is not specified in the `SOAPPart` object or `null` is specified in the arguments of the `getMimeHeader(String name)` method, `null` is returned.
- In the arguments of the `setContent(Source source)` method, specify a `Source` object with contents suitable for both XML and SOAP. If a `Source` object with invalid contents is specified, the operations are not guaranteed. Also, if `null` is specified, the operations are not guaranteed.
- If the following methods are used, the operations are not guaranteed:
  - `getContentId()`
  - `getContentLocation()`
  - `setContentId(String contentId)`
  - `setContentLocation(String contentLocation)`

22.1.17 Support range for using attachments

With a Web Service developed by using Provider Implementation Class or a dispatch-based Web Service client, you can generate and send or receive SOAP messages with attachments according to the SAAJ specifications. The size and number of attachments that can be sent and received at a time changes according to the amount of memory of the execution environment, but there are no restrictions. If you increase the amount of memory, you can also send and receive large attachments or a large number of attachments at a time.

For details on the memory usage when an attachment is sent and received, see the appendix *C.3 Memory usage per request when attachments are used*.

(1) MIME type

The MIME types corresponding to the attachments are determined according to the attachment extensions. If the MIME type of the attachment is not clearly specified, an appropriate MIME type is automatically set up according to the attachment extension. When the MIME type is clearly specified using methods such as the `AttachmentPart#setContentType()` method, you must specify an appropriate MIME type corresponding to the attachment extension. If an invalid MIME type is specified, the operations are not guaranteed.

The following table lists the appropriate combinations of the attachment extensions and the MIME types. For extensions other than those listed in the following table, the MIME type used is `application/octet-stream`.

Table 22-3: Attachment extensions and MIME types

No.	Attachment extensions	Corresponding MIME types
1	`html`, `htm`	`text/html`
2	`txt`, `text`	`text/plain`
3	`gif`, `GIF`	`image/gif`
4	`ief`	`image/ief`
5	`jpeg`, `jpg`, `jpe`, `JPG`	`image/jpeg`
6	`tiff`, `tif`	`image/tiff`
7	`xwd`	`image/x-xwindowdump`
8	`ai`, `eps`, `ps`	`application/postscript`
9	`rtf`	`application/rtf`
10	`tex`	`application/x-tex`
11	`texinfo`, `texi`	`application/x-texinfo`
12	`t`, `tr`, `roff`	`application/x-troff`
13	`au`	`audio/basic`
14	`midi`, `mid`	`audio/midi`
15	`aifc`	`audio/x-aifc`
16	`aif`, `aiff`	`audio/x-aiff`
17	`wav`	`audio/x-wav`
18	`mpeg`, `mpg`, `mpe`	`video/mpeg`
19	`qt`, `mov`	`video/quicktime`
20	`avi`	`video/x-msvideo`

(2) Notes on reading attachments

To read a file and then send or receive the file as an attachment, you must specify the object read using `javax.activation.FileDataSource` in the attachment instead of the object read using `java.io.FileInputStream`. An example is as follows:

```
AttachmentPart apPart = request.createAttachmentPart();
FileDataSource source = new FileDataSource("D:\\attachment.txt");
apPart.setDataHandler(new DataHandler(source));
request.addAttachmentPart(apPart);
```

The operations are not guaranteed if an object read using `java.io.FileInputStream` is specified.

(3) Notes on using DOM APIs

When you use the DOM APIs to create the SOAP messages, do not use the following methods. If used, the operations are not guaranteed.

- `org.w3c.dom.createEntityReference(String name)`
- `org.w3c.dom.createProcessingInstruction(String target, String data)`

(4) Notes on attaching multiple files

To send multiple attachments at one time, you must set up a unique `Content-ID` for every `AttachmentPart` object. If you attempt to send multiple attachments without specifying `Content-ID` or by specifying duplicated `Content-IDs`, only the attachment specified last is sent.

The following is an example of setting up a unique `Content-ID` for multiple `AttachmentPart` objects:

```
AttachmentPart apPart1 = request.createAttachmentPart();
FileDataSource source1 = new FileDataSource("D:\\attachment1.txt");
apPart1.setDataHandler(new DataHandler(source1));
apPart1.setContentId("001");
request.addAttachmentPart(apPart1);

AttachmentPart apPart2 = request.createAttachmentPart();
FileDataSource source2 = new FileDataSource("D:\\attachment2.txt");
apPart2.setDataHandler(new DataHandler(source2));
apPart2.setContentId("002");
request.addAttachmentPart(apPart2);

AttachmentPart apPart3 = request.createAttachmentPart();
FileDataSource source3 = new FileDataSource("D:\\attachment3.txt");
apPart3.setDataHandler(new DataHandler(source3));
apPart3.setContentId("003");
request.addAttachmentPart(apPart3);
```

23 Support Range of the WS-RM Specifications

This chapter describes the support range of the WS-RM specifications. For details on the WS-RM 1.2 functionality, see *34. WS-RM 1.2 Functionality*.

23.1 Support range of the WS-RM 1.2 specifications

The following table describes the support range of the WS-RM 1.2 specifications. Note that the major division in the table indicates the corresponding location (chapter, section, or subsection) in the WS-RM 1.2 specifications, and the subdivision indicates the contents of the corresponding location in the WS-RM 1.2 specifications.

Table 23－1: Support range of the WS-RM 1.2 specifications

Division				Support
Major division	Subdivision			
2.4	Delivery guarantee	`AtLeastOnce`		N
		`AtMostOnce`		N
		`ExactlyOnce`		Y
		`InOrder`		N
3	`RM` element			Y
3.1	Extended element/ extended attribute consideration [#1]			Y
3.2	Piggy-Backing			Y
3.3	WS-Addressing usage			Y
3.4	Sequence generating			Y
3.4	Sequence-generating request	`wsrm:CreateSequence`		Y
		`wsrm:AcksTo`[#2]		A
		`wsrm:Expires`[#3]		N
		`wsrm:Offer`		Y
		Extended element/ extended attribute [#1]		Y
3.4	Sequence-generating response	`wsrm:CreateSequenceResponse`		Y
		`wsrm:Identifier`		Y
		`wsrm:Expires`[#3]		N
		`wsrm:IncompleteSequenceBehavior`	`DiscardEntireSequence`	N
			`DiscardFollowingFirstGap`	N
			`NoDiscard`	Y
		`wsrm:Accept`		Y
		Extended element/ extended attribute [#1]		Y
3.5	Sequence closing			Y
3.5	Sequence closing request	`wsrm:CloseSequence`		Y
		`wsrm:Identifier`		Y
		`wsrm:LastMsgNumber`		Y
		Extended element/ extended attribute [#1]		Y

Division			Support
Major division	Subdivision		
3.5	Sequence closing response	wsrm:CloseSequenceResponse	Y
		wsrm:Identifier	Y
		Extended element/ extended attribute #1	Y
3.6	Sequence ending		Y
3.6	Sequence ending request	wsrm:TerminateSequence	Y
		wsrm:Identifier	Y
		wsrm:LastMsgNumber	Y
		Extended element/ extended attribute #1	Y
3.6	Sequence ending response	wsrm:TerminateSequenceResponse	Y
		wsrm:Identifier	Y
		Extended element/ extended attribute #1	Y
3.7	Sequence		Y
	Sequence element	wsrm:Sequence	Y
		wsrm:Identifier	Y
		wsrm:MessageNumber	Y
3.8	Ack request		Y
3.8	Ack request element	wsrm:AckRequested	Y
		wsrm:Identifier	Y
		Extended element/ extended attribute #1	Y
3.9	Ack		Y
3.9	Ack element	wsrm:SequenceAcknowledgement	Y
		wsrm:Identifier	Y
		wsrm:AcknowledgementRange	Y
		wsrm:None#4	A
		wsrm:Final	Y
		wsrm:Nack	N
		Extended element/ extended attribute #1	Y
4	Fault		Y
4	SOAP 1.1 support		Y
4	SOAP 1.2 support		Y
4.1	wsrm:SequenceFault fault		Y
4.2	wsrm:SequenceTerminated fault		Y
4.3	wsrm:UnknownSequence fault		Y

Division		Support
Major division	Subdivision	
4.4	`wsrm:InvalidAcknowledgement` fault	Y
4.5	`wsrm:MessageNumberRollover` fault	Y
4.6	`wsrm:CreateSequenceRefused` fault	Y
4.7	`wsrm:SequenceClosed` fault	Y
4.8	`wsrm:WSRMRequired` fault	N
5	Security threats and countermeasures	N
6	Secured sequence	N

Legend:

Y: Supported in the Cosminexus WS-RM 1.2 functionality.

N: Not supported in the Cosminexus WS-RM 1.2 functionality.

A: Supported in the Cosminexus WS-RM 1.2 functionality, but with some restrictions.

#1

The extended elements and extended attributes are not added with the Cosminexus WS-RM 1.2 functionality. The extended elements and extended attributes included in the received messages are ignored.

#2

The only element value that is available is an anonymous URI.

#3

The sequence validity period settings using the `wsrm:Expires` element is not supported. Set the sequence validity period by specifying `net35rmpInactivityTimeout` in the WSDL.

#4

The `wsrm:None` element is not sent with the Cosminexus WS-RM 1.2 functionality. When there is no `Ack` to be returned, the HTTP status code 202 is returned. If `Ack` is included in a received message, the message is processed normally.

23.2 Support range of the WS-RM Policy 1.2 specifications

The following table describes the support range of the WS-RM Policy 1.2 specifications. Note that the major division in the table indicates the corresponding location (chapter, section, or subsection) in the WS-RM Policy 1.2 specifications, and the subdivision indicates the contents of the corresponding location in the WS-RM Policy 1.2 specifications.

Table 23-2: Support range of the WS-RM Policy 1.2 specifications

Division			Support
Major division	Subdivision		
2.2	Assertion element	/wsrmp:RMAssertion	Y
		/wsrmp:RMAssertion/@wsp:Optional	N
		/wsrmp:RMAssertion/wsp:Policy	Y
		/wsrmp:RMAssertion/wsp:Policy/ wsrmp:SequenceSTR	N
		/wsrmp:RMAssertion/wsp:Policy/ wsrmp:SequenceTransportSecurity	N
		/wsrmp:RMAssertion/wsp:Policy/ wsrmp:DeliveryAssurance	Y
		/wsrmp:RMAssertion/wsp:Policy/ wsrmp:DeliveryAssurance/wsp:Policy	Y
		/wsrmp:RMAssertion/wsp:Policy/ wsrmp:DeliveryAssurance/wsp:Policy/ wsrmp:ExactlyOnce	Y
		/wsrmp:RMAssertion/wsp:Policy/ wsrmp:DeliveryAssurance/wsp:Policy/ wsrmp:AtLeastOnce	N
		/wsrmp:RMAssertion/wsp:Policy/ wsrmp:DeliveryAssurance/wsp:Policy/ wsrmp:AtMostOnce	N
		/wsrmp:RMAssertion/wsp:Policy/ wsrmp:DeliveryAssurance/wsp:Policy/ wsrmp:InOrder	N
		Extended element/ extended attribute #	Y
2.3	Assertion attachment	/wsdl:definitions/wsdl:service/wsdl:port	N
		/wsdl:definitions/wsdl:binding	Y
		/wsdl:definitions/wsdl:binding/wsdl:operation/ wsdl:input	N
		/wsdl:definitions/wsdl:binding/wsdl:operation/ wsdl:output	N
		/wsdl:definitions/wsdl:binding/wsdl:operation/ wsdl:fault	N
2.5	Sequence security policy		N

Legend:

Y: Supported in the Cosminexus WS-RM Policy 1.2 functionality.

N: Not supported in the Cosminexus WS-RM Policy 1.2 functionality.

#

The extended elements and extended attributes are not added with the Cosminexus WS-RM Policy 1.2 functionality. The extended elements and extended attributes included in the received messages are ignored.

23.3 com.sun.xml.ws.Closeable class

The following table describes the support range of the `com.sun.xml.ws.Closeable` class.

Table 23-3: List of methods of the com.sun.xml.ws.Closeable class

No.	Return value type	Method name/Explanation	
1	`void`	`close()`	
		Explanation	To close the sequence and exit, you must invoke the `close()` method by casting the port object to the `com.sun.xml.ws.Closeable` type on the client machine. You cannot invoke a Web Service method after invoking the `close()` method. To communicate again you need to re-acquire the port object.
		Exception	`javax.xml.ws.WebServiceException`: This exception occurs when you use a Web Service method after invoking the `close()` method.

23.4 Settings using WS-Policy

In the WS-RM 1.2 functionality, unique settings are coded in the WSDL as WS-Policy in addition to the WS-RM Policy settings. This section describes the properties to be added in the WSDL.

For details on how to add the WS-RM Policy, see *34.4 How to add the WS-RM Policy*.

Table 23-4: Properties to be added in the WSDL

No.	Property	Explanation	Unit	Range	Default value
1	<net35rmp:InactivityTimeout Milliseconds = "*set-value*"/>	Specifies the sequence validity period. To set up the re-transmission interval for the application messages, specify a value that is equal to or more than the re-transmission interval for the application messages. If there is no communication during the set period The validity period of the sequence expires and the sequence closes automatically. If a message is sent after the validity period expires The SequenceTerminatedException or UnknownSequenceException exceptions, which are the child classes of WebServiceException, occur. If a message is received after the validity period expires The SequenceTerminated fault or UnknownSequence fault is sent back. If the communication continues You must re-acquire the port object and re-generate the sequence. If the specified value is out of range A warning message is output and the operation is executed with the default value (KDJR16017-W).	millise conds	1 to 9,223,372,036,854,775,807	600,000
2	<cwsrm:MaxMessageNumber value="*set-value*"/>	Specifies the maximum number of messages that can be handled in 1 sequence. If sent messages exceed the set number of messages The MessageNumberRolloverException exception, which is a child class of WebServiceException, occurs. If received messages exceed the set number of messages The MessageNumberRollover fault is sent back. If the communication continues To terminate the existing sequence, invoke the close method by casting the port object to com.sun.xml.ws.Closeable. Then you must re-acquire the port object and re-generate the sequence.	--	1 to 100,000	10,000

No.	Property	Explanation	Unit	Range	Default value
2	<cwsrm:MaxMessageNumber value="*set-value*"/>	If the specified value is out of range A warning message is output and the operation is executed with the default value (KDJR16017-W).	--	1 to 100,000	10,000
3	<metro:AckRequestInterval Milliseconds="*set-value*"/>	Specifies the interval for sending the Ack messages that are automatically sent by the WS-RM 1.2 functionality in the background. Depending on the timing, an interval that is about twice the set value might be opened. If the specified value is out of range, a warning message is output and the operation is executed with the default value (KDJR16017-W).	milliseconds	1 to 9,223,372,036,854,775,807	2,000
4	<metro:RetransmissionConfig> <metro:Interval Milliseconds="*set-value*"/> </metro:RetransmissionConfig>	Specifies the re-transmission interval for the application messages. If the specified value is out of range, a warning message is output and the operation is executed with the default value (KDJR16017-W).	milliseconds	1 to 9,223,372,036,854,775,807	2,000
5	<metro:RetransmissionConfig> <metro:MaxRetries>*set-value*</metro:MaxRetries> </metro:RetransmissionConfig>	Specifies the number of re-transmitted application messages. If the specified re-transmission count is exceeded, the re-transmission process is stopped, and the error that occurred is returned to the application. If 0 is specified The re-transmission count becomes infinite. If the specified value is out of range A warning message is output and the operation is executed with the default value (KDJR16017-W).	--	0 to 9,223,372,036,854,775,807	3

Legend:

--: None

The following is an example of WS-RM Policy in which the sequence validity period is set at 300,000 milliseconds (5 min) and the maximum number of messages in the sequence is set at 1,000:

```
<wsp:Policy wsu:Id="WSRM_policy">
 <wsp:ExactlyOne>
  <wsp:All>
   <wsrmp:RMAssertion>
    <wsp:Policy>
     <wsrmp:DeliveryAssurance>
      <wsp:Policy>
       <wsrmp:ExactlyOnce/>
      </wsp:Policy>
     </wsrmp:DeliveryAssurance>
    </wsp:Policy>
   </wsrmp:RMAssertion>
   <wsaw:UsingAddressing/>
   <net35rmp:InactivityTimeout Milliseconds="300000"/>
   <cwsrm:MaxMessageNumber value="1000"/>
  </wsp:All>
 </wsp:ExactlyOne>
</wsp:Policy>
```

24 Support Range of JAX-RS Specifications

This chapter describes the support range of the JAX-RS specifications that you must consider when developing RESTful Web services (Web resources).

24.1 Support range of JAX-RS 1.1 specifications

This section describes the support range of the JAX-RS 1.1 specifications.

The following table describes the support range of the JAX-RS 1.1 specifications:

Table 24－1: Support range of the JAX-RS 1.1 specifications

Classification			Support	Remarks
Major classification#	Minor classification			
2	Application	Built-in implementation	Y	The JAX-RS functionality of Cosminexus supports the servlet-based mechanism (described in the last paragraph of *section 2.3.2* of the *JAX-RS 1.1 specifications*). Also, you do not need to implement the *Application* when implementing RESTful Web Services, because the JAX-RS functionality provides a default built-in *Application* implementation (customized implementation of the *Application* is not supported). For details on the built-in *Application* implementation and deployment, see *3.5.1 Configuring WAR files*.
		Customized implementation	N	
3.1, 3.3	Request method identifier	Built-in implementation	Y	The JAX-RS functionality of Cosminexus supports the following standard request method identifiers defined in the JAX-RS 1.1 specifications: • `GET` annotation • `POST` annotation • `PUT` annotation • `DELETE` annotation • `HEAD` annotation The JAX-RS functionality of Cosminexus does not support a customized implementation of the request method identifiers, which are optional in the JAX-RS 1.1 specifications.
		Customized implementation	N	
3.1	Root resource class		Y	For details on the root resource class, see *17.1.1 Root resource class*.
3.1	Life cycle of a root resource class		Y	For details on the life cycle of a root resource class, see *17.1.1(1) Life cycle*.
3.1.2	Injection to the `constructor` parameter of a root resource class		Y	For details on injection to the `constructor` parameter of a root resource class, see *17.1.1(2) Constructors*. For details on the injectable types, or combination with the `DefaultValue` annotation, see *17.1.4 Parameter types*.
3.2	Injection to the `field` and `bean` property		Y	For details on injection to the `field` and `bean` property, see *17.1.1(3) Field and bean property*. For details on injectable types, or combination with the `DefaultValue` annotation, see *17.1.4 Parameter types*.
3.3	Resource method		Y	For details on the resource method, see *17.1.1(4) Resource method*.
3.3.2	Injection to the parameters of the resource method		Y	For details on injection to the parameter of a resource method, see *17.1.1(4) (b) Annotation of the parameter*. For details on injectable types, or combination with the

Classification			Support	Remarks
Major classification#	Minor classification			
3.3.2	Injection to the parameters of the resource method		Y	`DefaultValue` annotation, see *17.1.4 Parameter types*.
3.3.2(1)	Mapping from an entity body of an HTTP request to `entity` parameter		Y	For details on `entity` parameters, see *17.1.1(4) (c) Entity parameter*.
3.3.3	Mapping from a return value to the HTTP response entity body		Y	For details on return value, see *17.1.1(4) (d) Return value*.
3.2, 3.3.4	Exception handling		Y	For details on exceptions, see *17.1.5 Exception mapping*.
3.3.5	Processing of `HEAD HTTP` and `OPTIONS HTTP` requests		Y	For details on processing of HEAD HTTP requests, see *24.3.2(3) javax.ws.rs.HEAD annotations*. For details on processing of OPTIONS HTTP requests, see *24.3.2(4) javax.ws.rs.OPTIONS annotations*.
3.4, 3.7.3	URI template and regular expressions		Y	For details on a URI template, see *17.1.6 URI template*.
3.4.1	Sub-resource method		Y	For details on a sub-resource method, see *17.1.1 (5) Sub-resource method*.
3.4.1	Sub-resource locator and Sub-resource		Y	For details on a sub-resource locator, see *17.1.1 (6) Sub-resource locator*. For details on sub-resource class, see *17.1.7 Sub-resource class*.
3.5	Media type declaration		Y	For details on a media type declaration, see *17.1.9 Media type declaration*.
3.6	Annotation inheritance		Y	For details on an annotation inheritance, see *17.1.11 Annotation inheritance*.
3.7	Mapping from an HTTP request to a resource method		Y	--
3.8	Determining a media type of an HTTP response		Y	--
4.2	Entity provider (Message body reader and writer)	Built-in implementation	Y	You do not need to implement an entity provider when implementing RESTful Web Services, because the JAX-RS functionality of Cosminexus provides a built-in entity provider (customized implementation of the entity provider is not supported). Built-in entity providers include entity providers that are supported additionally by the JAX-RS functionality, besides the entity providers for which support is mandatory according to the JAX-RS 1.1 specifications. For details on the built-in entity providers and supported types, see *17.1.1(4)(c) Entity parameters*.
		Customized implementation	N	
4.3	Context provider		N	The JAX-RS functionality of Cosminexus processes the standard context of the JAX-RS 1.1 specifications appropriately, even without having to implement a context provider.
4.4	Exception mapping provider		Y	For details on the exception mapping provider, see *17.2.2 Exception mapping provider*.
5	Context		Y	The JAX-RS functionality of Cosminexus supports the standard context type. For details on context, see *24.4 Context*.

Classification		Support	Remarks
Major classification#	Minor classification		
5.2.1	`javax.ws.rs.core.Application`	N	--
5.2.2	`javax.ws.rs.core.UriInfo`	Y	For details on `javax.ws.rs.core.UriInfo`, see *24.4.1 javax.ws.rs.core.UriInfo*.
5.2.3	`javax.ws.rs.core.HttpHeaders`	Y	For details on `javax.ws.rs.core.HttpHeaders`, see *24.4.2 javax.ws.rs.core.HttpHeaders*.
5.2.4	`javax.ws.rs.core.Request`	Y	For details on `javax.ws.rs.core.Request`, see *24.4.3 javax.ws.rs.core.Request*.
5.2.5	`javax.ws.rs.core.SecurityContext`	Y	For details on `javax.ws.rs.core.SecurityContext`, see *24.4.4 javax.ws.rs.core.SecurityContext*.
5.2.6	`javax.ws.rs.ext.Providers`	Y	For details on `javax.ws.rs.ext.Providers`, see *24.4.5 javax.ws.rs.core.ext.Providers*.
6.1	`javax.servlet.ServletConfig`	Y	For details on `javax.servlet.ServletConfig`, see *24.4.6 javax.servlet.ServletConfig*.
6.1	`javax.servlet.ServletContext`	Y	For details on `javax.servlet.ServletContext`, see *24.4.7 javax.servlet.ServletContext*.
6.1	`javax.servlet.http.HttpServletRequest`	Y	For details on `javax.servlet.http.HttpServletRequest`, see *24.4.8 javax.servlet.http.HttpServletRequest*.
6.1	`javax.servlet.http.HttpServletResponse`	Y	For details on `javax.servlet.http.HttpServletResponse`, see *24.4.9 javax.servlet.http.HttpServletResponse*.
6.1	Execution environment based on a Servlet container (Web container)	Y	The JAX-RS functionality provides an execution environment for RESTful Web Services based on the servlet container. For details, see also *1.4.2(2) (c) Application*.
6.1	Injection of the types defined in the servlet specifications	Y	See the following additional items: • *javax.servlet.ServletConfig* • *javax.servlet.ServletContext* • *javax.servlet.http.HttpServletRequest* • *javax.servlet.http.HttpServletResponse*
6.1	Processing the request entity streaming and committing a response within a method	N	The JAX-RS functionality of Cosminexus does not support the operations described in the latter half of the *sub-section 6.1* of the *JAX-RS 1.1 specifications*.
6.2	Execution environment based on a Java EE container (EJB container)	N	--
7	Runtime delegate	Y	The JAX-RS functionality of Cosminexus is implemented according to the runtime delegate mechanism of the JAX-RS 1.1 specifications.
Apx.A	Annotation	Y	For details on API, see *24.2 Support range of API*.
Apx.B	HTTP header	Y	--
JavaDoc	API	Y	For details on API, see *24.2 Support range of API*.

Legend:

Y: Supported

N: Not supported

--: Not applicable

#:

Indicates the corresponding places (chapters, sections, sub-sections) in the JAX-RS 1.1 specifications.

24.2 Support range of API

This section describes the support range of the interfaces and classes of the JAX-RS API and the points you must consider when using the classes and interfaces of the JAX-RS 1.1 specifications.

The following table describes the support range of the interfaces and classes of the JAX-RS API specifications. For details on the interfaces and classes, see *JAX-RS API documentation*.

Table 24-2: Support range of classes and interfaces of the JAX-RS 1.1 specifications

No.	Interfaces or Classes	Constructors/Methods/Fields	Support
javax.ws.rs package			
1	WebApplicationException	All methods	Y
2	ApplicationPath	--	N
3	Consumes	--	Y
4	CookieParam	--	Y
5	DefaultValue	--	Y
6	DELETE	--	Y
7	Encoded	--	Y
8	FormParam	--	Y
9	GET	--	Y
10	HEAD	--	Y
11	HeaderParam	--	Y
12	HttpMethod	--	N
13	MatrixParam	--	Y
14	OPTIONS	--	Y
15	Path	--	Y
16	PathParam	--	Y
17	POST	--	Y
18	Produces	--	Y
19	PUT	--	Y
20	QueryParam	--	Y
javax.ws.rs.core package			
21	HttpHeaders	getAcceptableMediaTypes()	Y
22		getCookies()	Y
23		getLanguages()	Y
24		getRequestHeader(String name)	Y
25		Other than the aforementioned methods	Y
26	MultivaluedMap<*K*,*V*>	All methods	Y
27	PathSegment	getPath()	Y
28		Other than the aforementioned methods	Y

No.	Interfaces or Classes	Constructors/Methods/Fields	Support
29	Request	evaluatePreconditions(java.util.Date lastModified)	Y
30		evaluatePreconditions(EntityTag eTag)	Y
31		evaluatePreconditions(java.util.Date lastModified, EntityTag eTag)	Y
32		getMethod()	Y
33		selectVariant(java.util.List<Variant> variants)	Y
34		Other than the aforementioned methods	Y
35	Response.StatusType	All methods	Y
36	SecurityContext	isUserInRole(String role)	Y
37		Other than the aforementioned methods	Y
38	StreamingOutput	All methods	Y
39	UriInfo	getMatchedResources()	N
40		getMatchedURIs()	N
41		getMatchedURIs(boolean decode)	N
42		getPath()	Y
43		getPath(boolean decode)	Y
44		getPathParameters()	Y
45		getPathParameters(boolean decode)	Y
46		getPathSegments()	Y
47		getPathSegments(boolean decode)	Y
48		getQueryParameters()	Y
49		getQueryParameters(boolean decode)	Y
50		getRequestUri()	Y
51		getRequestUriBuilder()	Y
52		Other than the aforementioned methods	Y
53	Application	All methods	N
54	CacheControl	All methods	Y
55	Cookie	valueOf(String value)	Y
56		Other than the aforementioned methods	Y
57	EntityTag	Entity(String value)	Y
58		valueOf(String value)	Y
59		Other than the aforementioned methods	Y
60	GenericEntity<*T*>	All methods	Y
61	MediaType	MediaType(String type, String subtype, java.util.Map < String,String> parameters)	Y

No.	Interfaces or Classes	Constructors/Methods/Fields	Support
62	MediaType	equals(Object obj)	Y
63		getParameters()	Y
64		isCompatible(MediaType other)	Y
65		valueOf(String type)	Y
66		Other methods	Y
67	NewCookie	valueOf(String value)	Y
68		Other than the aforementioned methods	Y
69	Response	created(URI location)	Y
70		fromResponse(Response response)	Y
71		notModified(EntityTag tag)	Y
72		notModified(String tag)	Y
73		ok(Object entity, String type)	Y
74		seeOther(URI location)	Y
75		status(int status)	Y
76		temporaryRedirect(URI location)	Y
77		Other than the aforementioned methods	Y
78	Response.ResponseBuilder	build()	Y
79		status(int status)	Y
80		Other method	Y
81	UriBuilder	build(Object... values)	Y
82		clone()	Y
83		fragment(String fragment)	Y
84		fromPath(String path)	Y
85		fromUri(String uri)	Y
86		fromUri(java.net.URI uri)	Y
87		host(String host)	Y
88		newInstance()	Y
89		path(String path)	Y
90		port(int port)	Y
91		queryParam(String name, Object... values)	Y
92		replacePath(String path)	Y
93		replaceQuery(String query)	Y
94		replaceQueryParam(String name, Object... values)	Y
95		scheme(String scheme)	Y
96		schemeSpecificPart(String ssp)	Y

No.	Interfaces or Classes	Constructors/Methods/Fields	Support
97	`UriBuilder`	`segment(String... segments)`	Y
98		`uri(java.net.URI uri)`	Y
99		`userInfo(String ui)`	Y
100		`Other than the aforementioned methods`	N
101	`Variant`	All methods	Y
102	`Variant.VariantListBuilder`	All methods	N
103	`Response.Status`	All methods	Y
104	`Response.Status.Family`	All methods	Y
105	`UriBuilderException`	All methods	Y
106	`Context`	--	Y
`javax.ws.rs.ext package`			
107	`ContextResolver<`*T*`>`	All methods	N
108	`ExceptionMapper<`*E extends Throwable*`>`	All methods	Y
109	`MessageBodyReader<`*T*`>`	All methods	N
110	`MessageBodyWriter<`*T*`>`	All methods	N
111	`Providers`	All methods	Y
112	`RuntimeDelegate.HeaderDelegate<`*T*`>`	All methods	Y#
113	`RuntimeDelegate`	All methods	Y#
114	`Provider`	--	Y

Legend:

Y: Supported by the JAX-RS functionality of Cosminexus

N: Not supported by the JAX-RS functionality of Cosminexus

--: No applicable methods or fields exist

#:

The user never uses directly

24.2.1 HttpHeaders interface

You must note the following points when using the methods of the `HttpHeaders` interface:

- A return value of the `getCookies()` method includes the version information of Cookie (Example: "cookieName=$Version=0;cookieName=cookieValue").
- If you specify a language code other than those laid down in the ISO 639 standards, in the `Content-Language` header of a received HTTP message, the operation of the `getLanguages()` method is not guaranteed.
- When using the `getRequestHeader(String name)` method, `null` is returned when you try to acquire a value of a header that does not exist in the Http request.

24.2.2 PathSegment interface

You must note the following points when using the methods of the `PathSegment` interface:

- When you use the `getPath()` method, a character code (Example: "%20") in the target path is returned in a decoded format.

24.2.3 Request interface

You must note the following points when using the methods of the `Request` interface:

- Do not specify `null` in the arguments of the following methods. If you specify `null`, the operation is not guaranteed.
  - `evaluatePreconditions(java.util.Date lastModified)`
  - `evaluatePreconditions(EntityTag eTag)`
  - `evaluatePreconditions(java.util.Date lastModified, EntityTag eTag)`
- Do not specify the EntityTag of `weak` in the argument `EntityTag eTag` of the following methods. If you specify EntityTag of `weak`, a `ResponseBuilder` instance with the HTTP status `412` (Precondition Failed) is returned.
  - `evaluatePreconditions(EntityTag eTag)`
  - `evaluatePreconditions(java.util.Date lastModified, EntityTag eTag)`
- If the `If-None-Match` header is set and no matching resource exists, the `If-Modified-Since` header will be ingored even if set.
- Do not call the `getMethod()` method from outside the scope of a request. If you call the `getMethod()` method from outside the scope of a request, the operation is not guaranteed.
- Even if you call the `selectVariant(java.util.List<Variant> variants)` method before calling either of the four `evaluatePreconditions()` methods with different arguments, the `Vary HTTP` header is not included in the `Response.ResponseBuilder` object, unlike the explanations in the API documentation of the JAX-RS 1.1 standard specifications.

24.2.4 SecurityContext interface

You must note the following points when using the methods of the `SecurityContext` interface:

- Do not specify `null` in the arguments of the `isUserInRole(String role)` method. If you specify `null`, the operation is not guaranteed.

24.2.5 UriInfo interface

You must note the following points when using the methods of the `UriInfo` interface:

- The `getAbsolutePath()` method, unlike the description in the API documentation of the JAX-RS 1.1 standard specifications, is not a shortcut of `UriInfo#getBase().resolve(uriInfo.getPath())` but a shortcut of `UriInfo#getBaseUri().resolve(uriInfo.getPath())`. No such method as `getBase()` exists.
- The operation is not guaranteed if you use the following methods:
  - `getMatchedResources()`
  - `getMatchedURIs()`
  - `getMatchedURIs(boolean decode)`
- The return values of the `getPath()` and `getPath(boolean decode)` methods include the matrix parameter information, and not the query parameter information.
- `MultivaluedMap`< *String, String*> acquired by the `getPathParameters()` and `getPathParameters(boolean decode)` methods differs from the description in the API documentation of the JAX-RS 1.1 standard specifications, and you can change the same.

- `java.util.List<`*PathSegment*`>` acquired by the `getPathSegments()` and `getPathSegments(boolean decode)` methods differs from the description in the API documentation of the JAX-RS 1.1 standard specifications, and you can change the same.
- `MultivaluedMap<` *String, String*`>` acquired by the `getQueryParameters()` and `getQueryParameters(boolean decode)` methods differs from the description in the API documentation of the JAX-RS 1.1 standard specifications, and you can change the same.

24.2.6 Cookie class

You must note the following points when using the methods of the `Cookie` class:

- Specify the arguments of the `valueOf(String value)` method in the following formats. The operation is not guaranteed if the arguments are specified in a format different from the following formats:
  - `Cookie.valueOf("$Version=xxxx;name=xxxx;$Path=xxxx;$Domain=xxxx;");`#
  - `Cookie.valueOf("name=xxxx;$Domain=xxxx;$Path=xxxx");`#
  - `Cookie.valueOf("$Version=xxxx;name=xxxx;");`#
  - `Cookie.valueOf("name=xxxx");`#

  #:

  Enter a value of the propertyin *xxxx*.

24.2.7 EntityTag class

You must note the following points when using the methods of the `EntityTag` class:

- You cannot generate the `EntityTag` instance of `weak` with the `EntityTag(String value)` constructor. Do not specify strings such as "*W/"tagValue*"" in arguments of the `EntityTag(String value)` constructor.
- When you use the `valueOf(String value)` method, quote the tag that you want to specify in the method argument. The operation is not guaranteed if you do not use quotation marks. Note the following specification example:

  String value = "\"entityTag\"";

  EntityTag entityTag = EntityTag.valueOf(value);
- When generating the `EntityTag` instance of `weak` by using the `valueOf(String value)` method, add the capital letter `W` before the tag to be specified in the argument. Note the following example:

  String value = "W/\"weakEntityTag\"";

  EntityTag entityTag = EntityTag.valueOf(value);

24.2.8 MediaType class

You must note the following points when using the methods of the `MediaType` class:

- Use the values that are compliant with the standard specifications. The operation is not guaranteed if you specify other non-compliant values.
- Do not register multiple identical keys (including strings that differ only in terms of upper case and lower case) in the map to be specified in the `parameters` argument of the `MediaType(String type, String subtype, java.util.Map<String, String> parameters)` constructor. The operation is not guaranteed if you register multiple identical keys.
- The keys `type`, `subtype`, and `parameters`, which are the attributes of the `MediaType` object to be compared, are not case sensitive when you use the `equals(Object obj)` method. The value of `parameters` is case-sensitive.
- The `type` and `subtype`, which are attributes of the `MediaType` object of an argument are not case sensitive when you use the `isCompatible(MediaType other)` method.

- Do not use the following characters in arguments of the `valueOf(String type)` method. If you use the following characters, the operation is not guaranteed.
  `: ( ) < > @ , ; : / " [ ] ? = { }`
  space, linefeed, carriage return, and horizontal tabulation

24.2.9 NewCookie class

You must note the following points when using the methods of the `NewCookie` class:

- Specify the arguments of the `valueOf(String value)` method in the following format. The operation is not guaranteed if you specify the arguments in a format different from the following formats:
  - `NewCookie.valueOf("name=xxxx;Version=xxxx;Comment=xxxx;Domain=xxxx;Path=xxxx;Max-Age=xxxx;Secure");`#
  - `NewCookie.valueOf("name=xxxx;Path=xxxx;Domain=xxxx;Comment=xxxx;Max-Age=xxxx;Secure");`#
  - `NewCookie.valueOf("name=xxxx;Comment=xxxx;Max-Age=xxxx;Secure");`#
  - `NewCookie.valueOf("name=xxxx;Path=xxxx;Domain=xxxx;Version=xxxx;");`#
  - `NewCookie.valueOf("name=xxxx;Path=xxxx;Domain=xxxx;");` #
  - `NewCookie.valueOf("name=xxxx;");` #

  #:
  Enter a value of the property *xxxx*.

24.2.10 Response class

You must note the following points when using the methods of the `Response` class:

- Do not specify `null` in the argument of the `fromResponse(Response response)` method. If you specify `null`, the operation is not guaranteed.
- Do not specify `null` in the arguments of the following methods. If you specify `null`, the `IllegalArgumentException` is not thrown unlike the description in the API documentation of the JAX-RS 1.1 standard specifications.
  - `created(URI location)`
  - `notModified(EntityTag tag)`
  - `notModified(String tag)`
  - `seeOther(URI location)`
  - `temporaryRedirect(URI location)`
- Specify the values that are compliant with the standard specifications, in the argument type of `ok(Object entity, String type)`. If you specify other non-compliant values, the operation is not guaranteed.
- Do not specify a number less than `100` or greater than `599` in the arguments of the `status(int status)` method. If you specify a number less than `100` or greater than `599`, `IllegalArgumentException` will not be thrown unlike the description in the API document of the JAX-RS 1.1 standard specifications.

24.2.11 Response.ResponsBuilder class

You must note the following points when using the methods of the `Response.ResponsBuilder` class:

- After you call `build()`, unlike the description of the API documentation of the JAX-RS 1.1 standard specifications, the `ResponseBuilder` interface will not be set to the `ok` status but will be reset to the `no content` status.

- Do not specify a number less than `100` or greater than `599` in the argument of the `status(int status)` method. If you specify a number less than `100` or greater than `599`, `IllegalArgumentException` will not be thrown unlike the description in the API documentation of the JAX-RS 1.1 standard specifications.

24.2.12 The UriBuilder class

Notes on using the methods of the `UriBuilder` class are as follows.

Note that the URI of the `UriBuilder` object in the description is, when calling any method, the URI that is already built in the `UriBuilder` object when any of the previous methods were called.

- When using the methods of the `UriBuilder` class, see the *JAX-RS API documentation* or *RFC 2396, 2732* and then use the characters and formats that can be used in the arguments of each method. If the characters in an argument or format of an argument is invalid, an exception might be thrown when calling methods of the `UriBuilder` class or the `build(Object... values)` method. Note that any invalid characters are automatically percent encoded in the following methods (already percent encoded characters are not further percent encoded).
  - `fragment(java.lang.String fragment)`
  - `host(String host)`
  - `path(String path)`
  - `queryParam(java.lang.String name, java.lang.Object... values)`
  - `replacePath(String path)`
  - `replaceQuery(java.lang.String query)`
  - `replaceQueryParam(java.lang.String name, java.lang.Object... values)`
  - `segment(String... segments)`
  - `userInfo(String ui)`
- Unknown URIs are not supported. Use the methods of the `URI Builder` claass only for the hierarchical URIs.
- The template parameters are not supported. The operation is not guaranteed if arguments of the methods contain template parameters.
- Do not specify arguments in the `build(Object... values)` method. The operation is not guaranteed if arguments are specified and then called.
- The order of query parameters is not maintained.
- When using the `port(int port)` method, no exception is thrown even if you specify a variable value greater than 65535. The `IllegalArgumentException` exception, however, is thrown if you specify a variable value smaller than `-1`.
- When using the `replaceQuery(String query)` method, the operation is not guaranteed if nothing is available to replace with the URI of the UriBuilder object.
- If you specify the `query` parameter that is not included in the URI of the UriBuilder object in the arguments of the `replaceQueryParam(String name, Object... values)` method, the specified parameter is added to the URI.
- If you specify `null` in the value arguments of the `replaceQueryParam(String name, Object... values)` method, the `IllegalArgumentException` exception is thrown without eliminating the existing `query` parameter, which differs from the description in the API documentation of the JAX-RS 1.1 standard specifications.
- If you specify `null` in the arguments of the `replacePath(String path)` method, the `IllegalArgumentException` exception is thrown without eliminating the existing `query` parameter, which differs from the description in the API documentation of the JAX-RS 1.1 standard specifications.

24.2.13 Provider annotation

You must note the following points when using the `Provider` annotation:

- You can use the `Provider` annotation only in a class wherein the `ExceptionMapper` interface is implemented.

24.3 Annotations

This section describes the details on the annotations of the JAX-RS 1.1 specifications supported by Cosminexus.

24.3.1 Injectable annotation

This subsection describes the Injectable annotations supported by the JAX-RS engine.

(1) Precautions:

(a) When using multiple annotations

You can use only one injectable annotation for a parameter, a field, or a bean property. If you specify multiple annotations for a parameter, a field, or a bean property, only the annotation on the extreme right side is enabled. If the annotation on the extreme right does not comply with the JAX-RS 1.1 specifications, all other annotations are ignored and injection is not performed.

(b) When using for the bean property

When using an injection purpose annotation for a bean property, annotate at the method level for the setter method.

(c) When using for field

In a root resource class or an exception mapping provider, do not reference or change the field annotated by an injectable annotation from the constructor and the setter method of the bean property.

The injection to a constructor, a field, or a bean property is concurrently performed when the JAX-RS engine instantiates the root resource class and the exception mapping provider. Therefore, if you reference the field annotated by an injectable annotation from the constructor and the setter method of bean property, the value acquired is undetermined. Also, if you change the field annotated by an injectiable annotation, either the injection to the field fails, or even if the injection succeeds, the value is undetermined.

(2) javax.ws.rs.HeaderParam annotation

Use the javax.ws.rs.HeaderParam annotation to acquire the value of an HTTP header. As the annotation value, specify the HTTP header name included in the HTTP request.

The following table lists the items for which you can specify the javax.ws.rs.HeaderParam annotation:

Table 24-3: Items for which you can specify the javax.ws.rs.HeaderParam annotation

Web resource or provider	Parameter of constructor	Field	bean property	Parameter of method
Root resource class	Y	Y	Y	Y
Sub-resource class	N	N	N	Y
Exception mapping provider	N	N	N	--

Legend:
Y: Can be specified
N: Cannot be specified
--: No applicable parameter exists

For details on the Web resources or providers, see the respective sections in *17. Web Resources and Providers*.

(3) javax.ws.rs.CookieParam annotation

Use the javax.ws.rs.CookieParam annotation to acquire the value of HTTP Cookie. As the annotation value, specify the Cookie name included in the HTTP request.

The following table lists the items for which you can specify the `javax.ws.rs.CookieParam` annotation:

Table 24-4: Items for which you can specify the `javax.ws.rs.CookieParam` annotation

Web resource or provider	Parameter of constructor	Field	bean property	Parameter of method
Application subclass	N	N	N	--
Root resource class	Y	Y	Y	Y
Sub-resource class	N	N	N	Y
Entity provider	N	N	N	--
Context provider	N	N	N	--
Exception mapping provider	N	N	N	--

Legend:
Y: Can be specified
N: Cannot be specified
--: No applicable parameter exists

For details on the Web resources or providers, see the respective sections in *17. Web Resources and Providers*.

(4) `javax.ws.rs.MatrixParam` annotation

Use the `javax.ws.rs.MatrixParam` annotation to acquire the value of a `URI matrix` parameter. As the annotation value, specify the matrix parameter name included in the HTTP request.

The following table lists the items for which you can specify the `javax.ws.rs.MatrixParam` annotation:

Table 24-5: Items for which you can specify the `javax.ws.rs.Matrix parameter` annotation

Web resource or provider	Parameter of constructor	Field	bean property	Parameter of method
Application sub-class	N	N	N	--
Root resource class	Y	Y	Y	Y
Sub-resource class	N	N	N	Y
Entity provider	N	N	N	--
Context provider	N	N	N	--
Exception mapping provider	N	N	N	--

Legend:
Y: Can be specified
N: Cannot be specified
--: No applicable parameter exists

For details on the Web resources or providers, see the respective sections in *17. Web Resources and Providers*.

(5) `javax.ws.rs.QueryParam` annotation

Use the `javax.ws.rs.QueryParam` annotation to acquire the value of a `URI query` parameter. As the annotation value, specify the query parameter name included in the HTTP request.

The following table lists the items for which you can specify the `javax.ws.rs.QueryParam` annotation:

Table 24-6: Items for which you can specify the `javax.ws.rs.QueryParam` annotation

Web resource or provider	Parameter of constructor	Field	bean property	Parameter of method
Application sub-class	N	N	N	--
Root resource class	Y	Y	Y	Y
Sub-resource class	N	N	N	Y
Entity provider	N	N	N	--
Context provider	N	N	N	--
Exception mapping provider	N	N	N	--

Legend:
Y: Can be specified
N: Cannot be specified
--: No applicable parameter exists

For details on the Web resources or providers, see the respective sections in *17. Web Resources and Providers*.

(6) `javax.ws.rs.PathParam` annotation

Use the `javax.ws.rs.PathParam` annotation to acquire the value of a URI path. Specify a template parameter as the annotation value.

The following table lists the items for which you can specify the `javax.ws.rs.PathParam` annotation:

Table 24-7: Items for which you can specify the `javax.ws.rs.PathParam` annotation

Web resource or provider	Parameter of constructor	Field	bean property	Parameter of method
Application sub-class	N	N	N	--
Root resource class	Y	Y	Y	Y
Sub-resource class	N	N	N	Y
Entity provider	N	N	N	--
Context provider	N	N	N	--
Exception mapping provider	N	N	N	--

Legend:
Y: Can be specified
N: Cannot be specified
--: No applicable parameter exists

For details on the Web resources or providers, see the respective sections in *17. Web Resources and Providers*.

(7) `javax.ws.rs.FormParam` annotation

Use the `javax.ws.rs.FormParam` annotation to acquire the value of the `form` parameter included in the entity body of the HTTP request. Specify the `form` parameter name as the annotation value.

The following table lists the items for which you can specify the `javax.ws.rs.FormParam` annotation:

Table 24-8: Items for which you can specify the `javax.ws.rs.FormParam` annotation

Web resource or provider	Parameter of constructor	Field	bean property	Parameter of method
Root resource class	Y	Y	Y	Y
Sub-resource class	N	N	N	Y
Exception mapping provider	N	N	N	--

Legend:

Y: Can be specified

N: Cannot be specified

--: No applicable parameter exists

For details on the Web resources or providers, see the respective sections in *17. Web Resources and Providers*.

The default upper limit of the count of the `form` parameters that are included in the entity body is `10000`. If the number of parameters specified in a request exceeds a specific value, an error (KDJJ10042-E) occurs and the `javax.ws.rs.WebApplicationException` exception that has `413` set as the HTTP status code is thrown. The `javax.ws.rs.WebApplicationException` exception can be handled by an exception mapping provider. Change the parameter count as and when required by editing the `webserver.connector.limit.max_parameter_count` property in the User Property file (`usrconf.properties`) of the J2EE server. For details on the User Property file of the J2EE server, see *2.4 usrconf.properties (User Property file of the J2EE server)* in the *uCosminexus Application Server Definition Reference Guide*.

(8) `javax.ws.rs.core.Context` annotation

Use the `javax.ws.rs.Context` annotation to inject the value of a context. The following table lists the items for which you can specify the `javax.ws.rs.Context` annotation:

Table 24-9: Items for which you can specify the `javax.ws.rs.Context` annotation (1)

Web resource or provider		Context Application type	Context UriInfo type	Context HttpHeaders type	Context Request type	Context Security Context type	Context Providers type
Root resource class	Parameter of constructor	N	Y	Y	Y	Y	Y
	Field	N	Y	Y	Y	Y	Y
	bean property	N	Y	Y	Y	Y	Y
	Parameter of method	N	Y	Y	Y	Y	Y
Sub-resource class	Parameter of constructor	N	N	N	N	N	N
	Field	N	N	N	N	N	N
	bean property	N	N	N	N	N	N
	Parameter of method	N	Y	Y	Y	Y	Y
Exception mapping provider	Parameter of constructor	N	N	N	N	N	Y
	Field	N	Y	Y	Y	Y	Y
	bean property	N	Y	Y	Y	Y	Y

Legend:

Y: Can be specified

N: Cannot be specified

Table 24-10: Items for which you can specify the `javax.ws.rs.Context` annotation (2)

Web resource or provider		Context HttpServletRequest type	HttpServlet Response type	ServletContext type	ServletConfig type
Root resource class	Parameter of constructor	Y#	Y	Y	Y
	Field	Y#	Y	Y	Y
	bean property	Y#	Y	Y	Y
	Parameter of method	Y#	Y	Y	Y
Sub-resource class	Parameter of constructor	N	N	N	N
	Field	N	N	N	N
	bean property	N	N	N	N
	Parameter of method	Y#	Y	Y	Y
Exception mapping provider	Parameter of constructor	Y#	Y	Y	Y
	Field	Y#	Y	Y	Y
	bean property	Y#	Y	Y	Y

Legend:

Y: Can be specified

N: Cannot be specified

#: An entity body or a query parameter cannot be acquired.

When you call the `getReader()` method of an `HttpServletRequest` instance injected by the `Context` annotation, with the resource method that contains an entity parameter, the `java.lang.IllegalStateException` that can be processed by an exception mapping provider is thrown.

(9) `javax.ws.rs.DefaultValue` annotation

You can use the `javax.ws.rs.DefaultValue` annotation in combination with the following annotations. When you use the `javax.ws.rs.DefaultValue` annotation, you can specify a default value if a value to be injected in the parameter annotated by the respective annotations does not exist in the HTTP request.

- `MatrixParam` annotation
- `QueryParam` annotation
- `CookieParam` annotation
- `HeaderParam` annotation
- `FormParam` annotation

For instance, if a query or a matrix parameter corresponding to an instance does not exist in a URI request, or if a target form parameter does not exist in the entity body of a request, then the default value that you specified in the `DefaultValue` annotation is used.

(10) `javax.ws.rs.Encoded` annotation

Use the `javax.ws.rs.Encoded` annotation to disable the automatic URL decoding of a URL encoded value. The URL encoded values of the parameters, fields, and bean properties annotated by the following injection annotations are automatically decoded:

- `javax.ws.rs.MatrixParam` annotation
- `javax.ws.rs.QueryParam` annotation
- `javax.ws.rs.PathParam` annotation

You can use the `javax.ws.rs.Encoded` annotation in:

- Root resource class (class level)
- Sub-resource class (class level)
- A constructor of a root resource class (constructor level)
- A constructor parameter of a root resource class (parameter level)
- A resource method, sub-resource method, or a sub-resource locator of a resource class (method level)
- Respective parameters of resource methods and sub-resource methods of a resource class, and of sub-resource locators (parameter level)
- Field and bean property of a root resource class (field level, property level)

When using at the parameter level, field level, or property level, use the `javax.ws.rs.Encoded` annotation in combination with the previously described injectable annotations (`javax.ws.rs.MatrixParam`, `javax.ws.rs.QueryParam`, `javax.ws.rs.PathParam`).

When using the `javax.ws.rs.Encoded` annotation at the class level, you can disable the automatic URL decoding of all the URL encoded values for all the parameters, fields, and bean properties of the concerned class.

When using the `javax.ws.rs.Encoded` annotation at the constructor level, you can disable the automatic URL decoding of the URL encoded values for all the parameters of the concerned constructor.

When using the `javax.ws.rs.Encoded` annotation at the method level, you can disable the automatic URL decoding of the URL encoded values for all the parameters of the respective resource methods, sub-resource methods, or sub-resource locators.

24.3.2 Built-in request method identifier

This subsection describes the request method identifiers supported by the JAX-RS engine.

(1) `javax.ws.rs.DELETE` annotation

The `javax.ws.rs.DELETE` annotation indicates that an annotated method processes an `HTTP DELETE` request. You can use the `javax.ws.rs.DELETE` annotation in:

- A `public` method of a root resource class
- A `public` method of a sub-resource class

(2) `javax.ws.rs.GET` annotation

The `javax.ws.rs.GET` annotation indicates that an annotated method processes an `HTTP GET` request. You can use the `javax.ws.rs.GET` annotation in:

- A `public` method of a root resource class
- A `public` method of a sub-resource class

(3) `javax.ws.rs.HEAD` annotation

The `javax.ws.rs.HEAD` annotation indicates that an annotated method processes an `HTTP HEAD` request. You can use the `javax.ws.rs.HEAD` annotation in:

- A `public` method of a root resource class
- A `public` method of a sub-resource class

When an `HTTP HEAD` request is received, the JAX-RS engine operates in the following priority sequence:

1. Calls a method annotated by the `HEAD` annotation, if such a method exists.

2. Calls a method annotated by the GET annotation, if no method annotated by the HEAD annotation exists. Note that if the method annotated by the GET annotation returns a value, the return value is ignored.

The following is an example of the root resource class that processes the HTTP HEAD request .

```
package com.sample.resources;

import javax.ws.rs.HEAD;
import javax.ws.rs.Path;
import javax.ws.rs.core.Response;

//Root resource class
@Path("/root")
public class Resource {

  //Resource method
  @HEAD
  public Response getValue() {

    String customHeader = "foo";
    String customHeaderValue = "bar";
    int httpStatus = 200;

    //Build the Response object by using ResponseBuilder

    return Response.status(httpStatus).header(customHeader,
        customHeaderValue).build();
  }
}
```

Consider the context root of the Web application (WAR file) containing the root resource class com.sample.resources.Resource to be *example*, and that the Web application is published on a host named *sample.com*. In this example, the HTTP HEAD request corresponding to the URL http://sample.com/example/root is dispatched to the getValue() method. Even if the Response object returned by the getValue() method contains an entity body, the object is ignored.

(4) javax.ws.rs.OPTIONS annotation

The javax.ws.rs.OPTIONS annotation indicates that an annotated method processes an HTTP OPTIONS request. You can use the javax.ws.rs.OPTIONS annotation in:

- A public method of a root resource class
- A public method of a sub-resource class

The JAX-RS engine handles the HTTP OPTIONS request as follows:

1. Calls a method annotated by the javax.ws.rs.OPTIONS annotation.
2. If a method annotated by the javax.ws.rs.OPTIONS annotation does not exist, the JAX-RS engine responds automatically by using the annotation information of the Web resource.

The following is an example of the root resource class that processes the HTTP OPTIONS request:

```
package com.sample.resources;

import javax.ws.rs.OPTIONS;
import javax.ws.rs.Path;
import javax.ws.rs.core.Response;

//Root resource class
@Path("/root")
public class Resource {

  //Resource method
  @OPTIONS
  public Response getValue() {
    String entity = "Some Contents";
    String customHeader = "foo";
    String customHeaderValue = "bar";
    int httpStatus = 200;

    //Build the Response object by using ResponseBuilder
```

```
        return Response.status(httpStatus).header(customHeader,
            customHeaderValue).entity(entity).build();

    }
}
```

Consider the context root of the Web application (WAR file) containing the root resource class `com.sample.resources.Resource` to be *example*, and that the Web application is published on a host named *sample.com*. In this example, the `HTTP OPTIONS` request corresponding to the URL `http://sample.com/example/root` is dispatched to the `getValue()` method.

(5) `javax.ws.rs.POST` annotation

The `javax.ws.rs.POST` annotation indicates that an annotated method processes an `HTTP POST` request. You can use the `javax.ws.rs.POST` annotation in:

- A `public` method of a root resource class
- A `public` method of a sub-resource class

(6) `javax.ws.rs.PUT` annotation

The `javax.ws.rs.PUT` annotation indicates that an annotated method processes an `HTTP PUT` request. You can use the `javax.ws.rs.PUT` annotation in:

- A `public` method of a root resource class
- A `public` method of a sub-resource class

24.3.3 Path specifying an annotation

This subsection describes a path specifying an annotation supported by the JAX-RS engine.

(1) `javax.ws.rs.Path` annotation

The `javax.ws.rs.Path` annotation specifies the path of a resource. The annotated class is considered as a root resource class when you use the `javax.ws.rs.Path` annotation at the class level. The annotated method is considered as a sub-resource method or a sub-resource locator when you use the `javax.ws.rs.Path` annotation at the method level. You can use the `javax.ws.rs.Path` annotation in:

- A root resource class (class level)
- A public method (method level) annotated by the request method identifier of a resource class (root resource class or sub-resource class): This is a sub-resource method.
- A public method (method level) that is not annotated by the request method identifier of a resource class (root resource class or sub-resource class): This method is a sub-resource locator.

24.3.4 Annotation for declaring the media type

This subsection describes the annotations for declaring the media types, supported by the JAX-RS engine.

(1) `javax.ws.rs.Consumes` annotation

The `javax.ws.rs.Consumes` annotation specifies a list of MIME media types supported by a Web resource in an HTTP request. You can use the `javax.ws.rs.Consumes` annotation in:

- A root resource class (class level)
- A resource method (method level)

- A sub-resource method (method level)
- A sub-resource class (class level)

The `javax.ws.rs.Consumes` annotation is ignored if used in the exception mapping provider.

(2) `javax.ws.rs.Produces` annotation

The `javax.ws.rs.Produces` annotation specifies a list of MIME media types supported by a Web resource in an HTTP response. You can use the `javax.ws.rs.Produces` annotation in:

- A root resource class (class level)
- A resource method (method level)
- A sub-resource method (method level)
- A sub-resource class (class level)

The `javax.ws.rs.Produces` annotation is ignored if used in the exception mapping provider.

24.4 Context

With the root resource class, sub-resource class, or an exception mapping provider, you can use the `Context` annotation to acquire the context defined according to the JAX-RS 1.1 specifications. The JAX-RS engine supports the following types of context:

- `javax.ws.rs.core.UriInfo`
- `javax.ws.rs.core.HttpHeaders`
- `javax.ws.rs.core.Request`
- `javax.ws.rs.core.SecurityContext`
- `javax.ws.rs.core.ext.Providers`
- `javax.servlet.ServletConfig`
- `javax.servlet.ServletContext`
- `javax.servlet.http.HttpServletRequest`
- `javax.servlet.http.HttpServletResponse`

Although the information included in the context differs for each HTTP request, because the provider instances are singleton, the information included in `javax.ws.rs.core.ext.Providers` is always the same for each WAR file.

The following subsection describes the types of context supported by the JAX-RS engine.

24.4.1 `javax.ws.rs.core.UriInfo`

The `javax.ws.rs.core.UriInfo` contains each component (query parameter or matrix parameter) that configures a URI and provides information about each HTTP request. Note that you can acquire the post-normalization information by using each method of the `javax.ws.rs.core.UriInfo` context.

The following example shows the usage of `javax.ws.rs.core.UriInfo` that is injected in the fields of a root resource class:

```
package com.sample.resources;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.UriInfo;

//Root resource class
@Path("/root")
public class Resource {
  //A field in which the UriInfo is injected by using the Context annotation
  private @Context UriInfo uriInfo;

  //Resource method
  @GET
  public String getValue() {
    String value = this.uriInfo.getQueryParameters().getFirst("query");
    return value;
  }
}
```

Consider the context root of the Web application (WAR file) containing the root resource class `com.sample.resources.Resource` to be *example*, and that the Web application is published on a host named *sample.com*. In this example, with the `HTTP GET` request corresponding to the URL , first, the `javax.ws.rs.core.UriInfo` context is injected into the `uriInfo` field and then the `getValue()` method that can process the `HTTP GET` request is called. Therefore, if you acquire the query parameter `query` from the `uriInfo` field by using the `getValue()` method, the value `10` is acquired.

24.4.2 javax.ws.rs.core.HttpHeaders

The javax.ws.rs.core.HttpHeaders contains the HTTP header of an HTTP request.

The following example shows the usage of javax.ws.rs.core.HttpHeaders that is injected into the field of a root resource class:

```
package com.sample.resources;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.HttpHeaders;

//Root resource class
@Path("/root")
public class Resource {
  //A field in which the HttpHeaders is injected by using the Context annotation
  private @Context HttpHeaders httpHeaders;

//Resource method
  @GET
  public String getValue () {
    String value = this.httpHeaders.getRequestHeader("Accept").get(0);
    return value;
  }
}
```

Consider the context root of the Web application (WAR file) containing the root resource class com.sample.resources.Resource to be *example*, and that the Web application is published on a host named *sample.com*. In the above example, with the HTTP GET request corresponding to the URL http://sample.com/example/root wherein application/xml is specified in the HTTP Accept header, first, the javax.ws.rs.core.HttpHeaders context is injected into the httpHeaders field and then the getValue() method that can process the HTTP GET request is called. Therefore, if you acquire the Accept-HTTP header value from the httpHeaders field by using the getValue() method, the application/xml value is acquired.

24.4.3 javax.ws.rs.core.Request

The javax.ws.rs.core.Request provides the functionality required for the *Content Negotiation* laid down in RFC 2616.

The following example describes the usage of javax.ws.rs.core.Request that is injected into the field of a root resource class:

```
package com.sample.resources;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.EntityTag;
import javax.ws.rs.core.Request;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.Response.ResponseBuilder;

//Root resource class
@Path("/root")
public class Resource {
  //field in which the Request is injected by using the Context annotation
  private @Context Request request;
  //HTTP Entity Tag to be used in the content negotiation
  private EntityTag eTag = new EntityTag("a-resource-status-specific-tag");

  //Resource method
  @GET
  public Response getData() {
ResponseBuilder rb = null;
    //perform the content negotiation (evaluation of preconditions)
    //  null if matched with the precondition, if not matched,
    //  the ResponseBuilder object is acquired in which
    //  a suitable ETag HTTP header or a status code (412: Precondition Failed) is set
    rb = request.evaluatePreconditions(this.eTag);
```

```
    if (rb != null) {
      //if not matched with the precondition, generate an HTTP response
      // from the ResponseBuilder object and return the response as it is
      return rb.build();
    } else {
      //if matched with the precondition, return the request data
      String data = "Some Information";
      return Response.ok().entity(data).build();
    }
  }
}
```

Consider the context root of the Web application (WAR file) containing the root resource class `com.sample.resources.Resource` to be *example*, and that the Web application is published on a host named *sample.com*. In the above example, with the `HTTP GET` request corresponding to the URL `http://sample.com/example/root` in which `a-resource-status-specific-tag` is specified in the `If-Match HTTP` header, first, the `javax.ws.rs.core.Request` context is injected into the request field and then the `getData()` method that can process the `HTTP GET` request is called. For this, content negotiation (in the above example, comparison between `HTTP EntityTag` of the resource and the `If-Match HTTP` header of the HTTP request) is performed with the `getData()` method and the value `Some Information` is acquired.

24.4.4 javax.ws.rs.core.SecurityContext

The `javax.ws.rs.core.SecurityContext` context saves the security information related to the HTTP request being processed.

The following example shows the usage of `javax.ws.rs.core.SecurityContext` that is injected into the field of a root resource class:

```
package com.sample.resources;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.SecurityContext;

//Root resource class
@Path("/root")
public class Resource {
  //A field in which the SecurityContext is injected by using the Context annotation
  private @Context SecurityContext securityContext;

  //Resource method
  @GET
  public String getValue () {
    String value = "Authentication Scheme: "
        + this.securityContext.getAuthenticationScheme()
        + ", User Principal: " + this.securityContext.getUserPrincipal()
        + ", Is secure: " + this.securityContext.isSecure()
        + ", Is user in role: " + this.securityContext.isUserInRole("admin");

    return value;
  }
}
```

The following is an example of `web.xml` containing the security information:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app ...>
  ...
  <security-constraint>
     <web-resource-collection>
       <web-resource-name>Test Resource</web-resource-name>
       <url-pattern>/*</url-pattern>
       <http-method>GET</http-method>
     </web-resource-collection>
     <auth-constraint>
       <role-name>admin</role-name>
     </auth-constraint>
  </security-constraint>
  <login-config>
     <auth-method>BASIC</auth-method>
```

```
      <realm-name>jaxrs_server</realm-name>
   </login-config>
   <security-role>
      <role-name>admin</role-name>
   </security-role>
</web-app>
```

Consider the context root of the Web application (WAR file) containing the root resource class `com.sample.resources.Resource` to be *example*, and that the Web application is published on a host named *sample.com*. In the above example, with the `HTTP GET` request corresponding to the URL `http://sample.com/example/root` in which an appropriate authentication information is specified, first, the `javax.ws.rs.core.SecurityContext` context is injected into the `securityContext` field and then the `getValue()` method that can process the `HTTP GET` request is called. With the `getValue()` method, the security information is acquired based on the `web.xml` settings and the actual authentication information.

24.4.5 javax.ws.rs.core.ext.Providers

The `javax.ws.rs.core.ext.Providers` saves the providers that operate in a deployed Web resource.

The following example describes the usage of `javax.ws.rs.core.ext.Providers` that is injected into the field of a root resource class:

```
package com.sample.resources;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.core.Context;
import javax.ws.rs.ext.Providers;

//Root resource class
@Path("root")
public class Resource{

  //A field in which the Providers is injected by using the Context annotation
  private @Context Providers providers;

  //resource method
  @GET
  public String getValue() {
    //acquire the exception mapping provider from the providers field
    return this.providers.getExceptionMapper(RuntimeException.class);
  }
}
```

Consider the context root of the Web application (WAR file) containing the exception `mapping provider com.sample.providers.EntityProviderReader` that can process the root resource class `com.sample.resources.Resource` and the `java.lang.RuntimeException` to be *example* and that the Web application is published on a host named *sample.com*. In the above example, with the `HTTP GET` request corresponding to the URL `http://sample.com/example/root`, first, the `javax.ws.rs.core.ext.Providers` context is injected into the `providers` field and then the `getValue()` method that can process the `HTTP GET` request is called. Therefore, if you use the `getValue()` method and acquire the exception mapping provider that can process the `java.lang.RuntimeException` from the `providers` field, the `com.sample.providers.EntityProviderReader` instance is acquired.

24.4.6 javax.servlet.ServletConfig

The `javax.servlet.ServletConfig` class is defined based on the Servlet 3.0 specifications.

The following example shows the usage of `javax.servlet.ServletConfig` that is injected to the field of a root resource class:

```
package com.sample.resources;

import javax.servlet.ServletConfig;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
```

```
import javax.ws.rs.core.Context;

//Root resource class
@Path("/root")
public class Resource {
  //A field in which the ServletConfig is injected by using the Context annotation
  private @Context ServletConfig config;

  //Resource method
  @GET
  public String getValue() {
    return this.config.getInitParameter("TestParam");
  }
}
```

The following is an example of `web.xml` that contains an additional initialization parameter (`init-param` element):

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app ...>
  <servlet>
    <servlet-name>CosminexusJaxrsServlet</servlet-name>
    <servlet-class>com.cosminexus.jersey.spi.container.servlet.ServletContainer</
servlet-class>
    <init-param>
      <param-name>com.cosminexus.jersey.config.property.packages</param-name>
      <param-value>com.sample.resources</param-value>
    </init-param>
    <init-param>
      <param-name>TestParam</param-name>
      <param-value>TestValue</param-value>
    </init-param>
  </servlet>
  <servlet-mapping>
    <servlet-name>CosminexusJaxrsServlet</servlet-name>
    <url-pattern>/*</url-pattern>
  </servlet-mapping>
</web-app>
```

Consider the context root of the Web application (WAR file) containing the root resource class `com.sample.resources.Resource` to be *example*, and that the Web application is published on a host named *sample.com*. In the above example, with the `HTTP GET` request corresponding to the URL `http://sample.com/example/root`, first, the `javax.servlet.ServletConfig` context is injected in the `config` field and then the `getValue()` method that can process the `HTTP GET` request is called. Therefore, if you use the `getValue()` method and acquire the initialization parameter `TestParam` from the config field, the value `TestValue` is acquired.

24.4.7 `javax.servlet.ServletContext`

The `javax.servlet.ServletContext` class is defined based on the Servlet 3.0 specifications.

The following example shows the usage of `javax.servlet.ServletContext` that is injected in the field of a root resource class:

```
package com.sample.resources;

import javax.servlet.ServletContext;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.core.Context;

//Root resource class
@Path("/root")
public class Resource {

  //A field in which the ServletContext is injected by using the Context annotation
  private @Context ServletContext context;

  //Resource method
  @GET
  public String getValue() {
    return this.context.getInitParameter("Hitachi");
```

```
    }
  }
```

The following is an example of `web.xml` that contains the initialization parameter (`context-param` element) of the context scope:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app ...>
  ...
  <context-param>
    <param-name>TestParam</param-name>
    <param-value>TestValue</param-value>
  </context-param>
</web-app>
```

Consider the context root of the Web application (WAR file) containing the root resource class `com.sample.resources.Resource` to be *example*, and that the Web application is published on a host named *sample.com*. In the above example, with the `HTTP GET` request corresponding to the URL `http://sample.com/example/root`, first, the `javax.servlet.ServletContext` context is injected in the context field and then the `getValue()` method that can process the `HTTP GET` request is called. Therefore, if you use the `getValue()` method and acquire the initialization parameter `TestParam` of the context scope from the context field, the value `TestValue` is acquired.

24.4.8 javax.servlet.http.HttpServletRequest

The `javax.servlet.http.HttpServletRequest` class is defined based on the Servlet 3.0 specifications.

The following example shows the usage of `javax.servlet.http.HttpServletRequest` that is injected into the field of a root resource class:

```
package com.sample.resources;

import javax.servlet.http.HttpServletRequest;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.core.Context;

//Root resource class
@Path("/root")
public class Resource {

  //A field in which the HttpServletRequest is injected by using the Context annotation
  private @Context HttpServletRequest httpRequest;

  //Resource method
  @GET
  public String getValue() {
    return this.httpRequest.getParameter("Hitachi");

  }
}
```

Consider the context root of the Web application (WAR file) containing the root resource class `com.sample.resources.Resource` to be *example*, and that the Web application is published on a host named *sample.com*. In the above example, with the `HTTP GET` request corresponding to the URL `http://sample.com/example/root?TestParam=TestValue`, first, the `javax.servlet.http.HttpServletRequest` context is injected in the `HttpRequest` field and then the `getValue()` method that can process the `HTTP GET` request is called. Therefore, if you acquire the request parameter `TestParam` from the `httpRequest` field by using the `getValue()` method, the value `TestValue` is acquired.

24.4.9 javax.servlet.http.HttpServletResponse

The `javax.servlet.http.HttpServletResponse` class is defined based on the Servlet 3.0 specifications.

The following example shows the usage of `javax.servlet.http.HttpServletResponse` that is injected into the field of a root resource class:

```
package com.sample.resources;

import java.io.IOException;
import javax.servlet.http.HttpServletResponse;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.core.Context;

//Root resource class
@Path("/root")
public class Resource {

  //A field in which the HttpServletResponse is injected by using the Context
annotation
  private @Context HttpServletResponse httpResponse;

  //Resource method
  @GET
  public void getValue() throws IOException {

    String entity = "Response mentioned using HttpServletResponse";

    httpResponse.setHeader("abc","xyz");
    httpResponse.getOutputStream().write(entity.getBytes());
    httpResponse.getOutputStream().flush();
    httpResponse.getOutputStream().close();

    return ;
  }
}
```

Consider the context root of the Web application (WAR file) containing the root resource class `com.sample.resources.Resource` to be *example*, and that the Web application is published on a host named `sample.com`. In the above example, with the `HTTP GET` request corresponding to the URL `http://sample.com/example/root`, first, `javax.servlet.http.HttpServletResponse` is injected in the `HttpResponse` field and then the `getValue()` method that can process the `HTTP GET` request is called. Therefore, the location where the `HttpResponse` class is used for building a response, and the `getValue()` method of the root resource class `com.sample.resources.Resource` is the same.

25 Support Range of the Client APIs for RESTful Web Services

This chapter describes the specifications and the support range of the client APIs for RESTful Web Services.

Note that in this chapter, the client APIs for RESTful Web Services are referred to as *client APIs*.

25.1 Support range of the client API interfaces and classes

This section describes the support range of the interfaces and classes of the client APIs. The following table lists the support range of the client API interfaces and classes.

Table 25-1: Support range of the client API interfaces and classes

No.	Interface or class	Constructor/method/field
`com.cosminexus.jersey.api.client` package		
1	`Client`	*create()*
2		*create(ClientConfig cc)*
3		*destroy()*
4		*getProperties()*
5		*handle(ClientRequest request)*
6		*resource(String u)*
7		*resource(URI u)*
8		*setChunkedEncodingSize(Integer chunkSize)*
9		*setConnectTimeout(Integer interval)*
10		*setFollowRedirects(Boolean redirect)*
11		*setReadTimeout(Integer interval)*
12	`ClientHandlerException`	Method of the parent class[#1]
13	`ClientRequest`	*clone()*
14		*create()*
15		*getEntity()*
16		*getHeaders()*
17		*getHeaderValue(Object headerValue)*
18		*getMethod()*
19		*getProperties()*
20		*getPropertyAsFeature(String name)*
21		*getPropertyAsFeature(String name, boolean defaultValue)*
22		*getURI()*
23		*setEntity(Object entity)*
24		*setMethod(String method)*
25		*setURI(java.net.URI uri)*
26	`ClientRequest.Builder`	*accept(MediaType... types)*
27		*accept(String... types)*
28		*acceptLanguage(Locale... locales)*
29		*acceptLanguage(String... locales)*

No.	Interface or class	Constructor/method/field
30	`ClientRequest.Builder`	*build(URI uri, String method)*
31		*cookie(Cookie cookie)*
32		*entity(Object entity)*
33		*entity(Object entity, MediaType type)*
34		*entity(Object entity, String type)*
35		*header(String name, Object value)*
36		*type(MediaType type)*
37		*type(String type)*
38	`ClientResponse`	*bufferEntity()*
39		*close()*
40		*getAllow()*
41		*getClient()*
42		*getClientResponseStatus()*
43		*getCookies()*
44		*getEntity(Class<T> c)*
45		*getEntity(GenericType<T> gt)*
46		*getEntityInputStream()*
47		*getEntityTag()*
48		*getHeaders()*
49		*getLanguage()*
50		*getLastModified()*
51		*getLength()*
52		*getLocation()*
54		*getResponseDate()*
55		*getStatus()*
56		*getType()*
57		*hasEntity()*
58	`ClientResponse.Status`	*ACCEPTED*
59		*BAD_GATEWAY*
60		*BAD_REQUEST*
61		*CONFLICT*
62		*CREATED*
63		*EXPECTATION_FAILED*
64		*FORBIDDEN*
65		*FOUND*
66		*GATEWAY_TIMEOUT*

No.	Interface or class	Constructor/method/field
67	`ClientResponse.Status`	*GONE*
68		*HTTP_VERSION_NOT_SUPPORTED*
69		*INTERNAL_SERVER_ERROR*
70		*LENGTH_REQUIRED*
71		*METHOD_NOT_ALLOWED*
72		*MOVED_PERMANENTLY*
73		*NO_CONTENT*
74		*NON_AUTHORITIVE_INFORMATION*
75		*NOT_ACCEPTABLE*
76		*NOT_FOUND*
77		*NOT_IMPLEMENTED*
78		*NOT_MODIFIED*
79		*OK*
80		*PARTIAL_CONTENT*
81		*PAYMENT_REQUIRED*
82		*PRECONDITION_FAILED*
83		*PROXY_AUTHENTICATION_REQUIRED*
84		*REQUEST_ENTITY_TOO_LARGE*
85		*REQUEST_TIMEOUT*
86		*REQUEST_URI_TOO_LONG*
87		*REQUESTED_RANGE_NOT_SATIFIABLE*
88		*RESET_CONTENT*
89		*SEE_OTHER*
90		*SERVICE_UNAVAILABLE*
91		*TEMPORARY_REDIRECT*
92		*UNAUTHORIZED*
93		*UNSUPPORTED_MEDIA_TYPE*
94		*USE_PROXY*
95		*fromStatusCode(int statusCode)*
96		*getFamily()*
97		*getReasonPhrase()*
98		*getStatusCode()*
99		*toString()*
100		*valueOf(String name)*
101		*values()*
102	`GenericType`	*GenericType()*

No.	Interface or class	Constructor/method/field
103	`GenericType`	*GenericType(Type genericType)*
104		*getRawClass()*
105		*getType()*
106	`UniformInterfaceException`	*getResponse()*
107	`WebResource`	*accept(MediaType... types)*
108		*accept(String... types)*
109		*acceptLanguage(Locale... locales)*
110		*acceptLanguage(String... locales)*
111		*cookie(Cookie cookie)*
112		*delete()*
113		*delete(Class<T> c)*
114		*delete(Class<T> c, Object requestEntity)*
115		*delete(GenericType<T> gt)*
116		*delete(GenericType<T> gt, Object requestEntity)*
117		*delete(Object requestEntity)*
118		*entity(Object entity)*
119		*entity(Object entity, MediaType type)*
120		*entity(Object entity, String type)*
121		*get(Class<T> c)*
122		*get(GenericType<T> gt)*
123		*getRequestBuilder()*
124		*getURI()*
125		*getUriBuilder()*
126		*head()*
127		*header(String name, Object value)*
128		*method(String method)*
129		*method(String method, Class<T> c)*
130		*method(String method, Class<T> c, Object requestEntity)*
131		*method(String method, GenericType<T> gt)*
132		*method(String method, GenericType<T> gt, Object requestEntity)*
133		*method(String method, Object requestEntity)*
134		*options(Class<T> c)*
135		*options(GenericType<T> gt)*
136		*path(String path)*
137		*post()*
138		*post(Class<T> c)*

No.	Interface or class	Constructor/method/field
139	`WebResource`	*post(Class<T> c, Object requestEntity)*
140		*post(GenericType<T> gt)*
141		*post(GenericType<T> gt, Object requestEntity)*
142		*post(Object requestEntity)*
143		*put()*
144		*put(Class<T> c)*
145		*put(Class<T> c, Object requestEntity)*
146		*put(GenericType<T> gt)*
147		*put(GenericType<T> gt, Object requestEntity)*
148		*put(Object requestEntity)*
149		*queryParam(String key, String value)*
150		*queryParams(MultivaluedMap<String, String> params)*
151		*type(MediaType type)*
152		*type(String type)*
153		*uri(java.net.URI uri)*
154	`WebResource.Builder`	*accept(MediaType... types)*
155		*accept(String... types)*
156		*acceptLanguage(Locale... locales)*
157		*acceptLanguage(String... locales)*
158		*cookie(Cookie cookie)*
159		*delete()*
160		*delete(Class<T> c)*
161		*delete(Class<T> c, Object requestEntity)*
162		*delete(GenericType<T> gt)*
163		*delete(GenericType<T> gt, Object requestEntity)*
164		`delete(Object requestEntity)`
165		*entity(Object entity)*
166		*entity(Object entity, MediaType type)*
167		*entity(Object entity, String type)*
168		*get(Class<T> c)*
169		*get(GenericType<T> gt)*
170		*head()*
171		*header(String name, Object value)*
172		*method(String method)*
173		*method(String method, Class<T> c)*
174		*method(String method, Class<T> c, Object requestEntity)*

No.	Interface or class	Constructor/method/field
175	`WebResource.Builder`	*method(String method, GenericType<T> gt)*
176		*method(String method, GenericType<T> gt, Object requestEntity)*
177		*method(String method, Object requestEntity)*
178		*options(Class<T> c)*
179		*options(GenericType<T> gt)*
180		*post()*
181		*post(Class<T> c)*
182		*post(Class<T> c, Object requestEntity)*
183		*post(GenericType<T> gt)*
184		*post(GenericType<T> gt, Object requestEntity)*
185		*post(Object requestEntity)*
186		*put()*
187		*put(Class<T> c)*
188		*put(Class<T> c, Object requestEntity)*
189		*put(GenericType<T> gt)*
190		*put(GenericType<T> gt, Object requestEntity)*
191		*put(Object requestEntity)*
192		*type(MediaType type)*
193		*type(String type)*
`com.cosminexus.jersey.api.client.config` package		
194	`DefaultClientConfig`	*PROPERTY_BUFFER_RESPONSE_ENTITY_ON_EXCEPTION*
195		*PROPERTY_CHUNKED_ENCODING_SIZE*
196		*PROPERTY_CONNECT_TIMEOUT*
197		*PROPERTY_FOLLOW_REDIRECTS*
198		*PROPERTY_READ_TIMEOUT*
199		*getPropertyAsFeature(String featureName)*
200		*getFeatures()*
201		*getFeature(String featureName)*
202		*getProperties()*
203		*getProperty(String propertyName)*
`com.cosminexus.jersey.client.urlconnection` package		
204	`HTTPSProperties`	*PROPERTY_HTTPS_PROPERTIES*
205		*HTTPSProperties()*
206		*HTTPSProperties(HostnameVerifier hv)*
207		*HTTPSProperties(HostnameVerifier hv, SSLContext c)*
208		*getHostnameVerifier()*

No.	Interface or class	Constructor/method/field
209	`HTTPSProperties`	*getSSLContext()*
`com.cosminexus.jersey.core.util` package		
210	`MultivaluedMapImpl`#2	*MultivaluedMapImpl()*
211		*add(String key, String value)*
212		*getFirst(String key)*
213		*putSingle(String key, String value)*

#1

The `ClientHandlerException` class is a derived class of the `RuntimeException` class. Use the `ClientHandlerException` class within the scope of the methods of the `RuntimeException` class.

#2

For details on the constructor and method specifications of the `MultivaluedMapImpl` class, see the documentation for the JAX-RS APIs. For details on the notes, see *25.14 Specifications for the constructors and methods of the MultivaluedMapImpl class and notes*.

25.1.1 Supported properties and features

This subsection describes the properties and features supported by the client APIs.

(1) Features

The following table lists the features and data types supported by the JAX-RS engine.

Table 25-2: Features and data types supported by the JAX-RS engine

No.	Feature	Data type
1	`com.sun.jersey.api.json.POJOMappingFeature` (`JSONConfiguration.FEATURE_POJO_MAPPING`)	`Boolean`

You can use this feature by adding the same to a changeable feature map. For details on how to use this feature, see *getFeatures() method*.

(2) Properties

The following table lists the properties and data types supported by the JAX-RS engine.

Table 25-3: Properties and data types supported by the JAX-RS engine

No.	Property	Data type
1	`com.sun.jersey.client.property.followRedirects` (`ClientConfig.PROPERTY_FOLLOW_REDIRECTS`)	`Boolean`
2	`com.sun.jersey.client.property.readTimeout` (`ClientConfig.PROPERTY_READ_TIMEOUT`)	`Integer`
3	`com.sun.jersey.client.property.connectTimeout` (`ClientConfig.PROPERTY_CONNECT_TIMEOUT`)	`Integer`
4	`com.sun.jersey.client.property.chunkedEncodingSize` (`ClientConfig.PROPERTY_CHUNKED_ENCODING_SIZE`)	`Integer`
5	`com.sun.jersey.client.property` `.bufferResponseEntityOnException`	`Boolean`

No.	Property	Data type
5	(ClientConfig.PROPERTY_BUFFER_RESPONSE_ENTITY_ON_EXCEPTION)	Boolean
6	com.sun.jersey.client.impl.urlconnection .httpsProperties (HTTPSProperties.PROPERTY_HTTPS_PROPERTIES)	HTTPSProperties

You can use these properties by adding the same to a changeable property map. For details on how to use these properties, see the following methods:

- `Client` class
  getProperties() method
- `ClientRequest` class
  getProperties() method
- `DefaultClientConfig` class
  getProperties() method

25.1.2 Information included in the ClientRequest class and the Web resource class

The following table describes the information included in the `ClientRequest` class and the Web resource class.

Table 25-4: Information included in the ClientRequest class and the Web resource class

Class	Included information
`ClientRequest` class	Web resource URI
	Name of an HTTP method
	Entity body of an HTTP request
	HTTP header
	Property map
Web resource class	Web resource URI
	Entity body of an HTTP request
	HTTP header

25.2 Method specifications and notes for the Client class

This section describes the specifications for the methods of the `Client` class and the notes on using the methods.

create() method

Description

This method generates a client (`Client` object).

Syntax

```
public static Client create()
```

Parameter

None.

Return value

The method returns the generated client.

Notes

- The execution result of this method is the same as when you specify only the generated `DefaultClientConfig` object in the `cc` parameter and invoke the `create(ClientConfig cc)` method.

create(ClientConfig cc) method

Description

This method generates a client (`Client` object) with the specified settings.

Syntax

```
public static Client create(ClientConfig cc)
```

Parameter

`cc`

This parameter sets up the client.

Return value

The method returns the generated client.

Notes

- Use this method when you want to initialize the `Client` object with specific properties or features. For details on the properties and features, see *25.1.1 Supported properties and features*. For details on how to set up the properties and features, see *getFeatures() method*and *getProperties() method*.
- Specify only the `DefaultClientConfig` object in the `cc` parameter.

destroy() method

Description

This method destroys the client. Consequently, all the system resources associated with the client are destroyed.

Invoke this method if there is no response during reception. If the method is invoked in other cases, the operation is not guaranteed.

After you invoke this method, do not reuse the `Client` object. If reused, the operation is not guaranteed.

Syntax

```
public void destroy()
```

Parameter

None.

Return value

None.

Notes

None.

getProperties() method

Description

This method acquires a changeable property map.

Syntax

```
public Map<String,Object> getProperties()
```

Parameter

None.

Return value

The method returns a property map.

Notes

- You can add or change properties in the returned property map. For details on the properties that you can add or change, see *25.1.1 Supported properties and features*. For details on the properties, see *11.4.3 Setting the properties and features*.

An example of using the `getProperties()` method is as follows:

```
// Generate the Client object
Client client = Client.create();
// Set the property read timeout value
client.getProperties().put(ClientConfig.PROPERTY_READ_TIMEOUT, 10000);

// Generate an HTTP request
```

```
ClientRequest cRequest = ClientRequest.create().build(new URI("http://test.com/"), "GET");
try{
 // Receive an HTTP response as the ClientResponse object
 ClientResponse cResponse = client.handle(cRequest);
} catch(ClientHandlerException e){
 // Execute the appropriate processing
}
```

In this example, initially the changeable property map is acquired with the `getProperties` method and the property read timeout is set to 10,000 milliseconds. Then, the `ClientRequest` object is created and the HTTP communication is performed by using the `handle` method of the `Client` class. The HTTP response is received as the `ClientResponse` object. If the read timeout occurs before the HTTP response is completely read, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps the `SocketTimeoutException` exception is thrown.

handle(ClientRequest request) method

Description

This method processes an HTTP request with the contents of the specified `ClientRequest` object and returns an HTTP response as the `ClientResponse` object.

Syntax

```
public ClientResponse handle(ClientRequest request)
throws ClientHandlerException
```

Parameter

`request`

This is an HTTP request.

Return value

The method returns an HTTP response.

Exception

`ClientHandlerException`

This exception is thrown if a problem occurs during the processing. The error message KDJJ18888-E is output to the log.

Notes

- If an exception is thrown during the processing of the HTTP request and HTTP response, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown. Exceptions are thrown under different conditions in different environments. Therefore, also see the notes for each of the following methods:

  `Client` class

  getProperties() method
  resource(String u) method
  resource(URI u) method
  setChunkedEncodingSize(Integer chunkSize) method
  setConnectTimeout(Integer interval) method
  setFollowRedirects(Boolean redirect) method
  setReadTimeout(Integer interval) method

`ClientRequest` class
getProperties() method
setEntity(Object entity) method
setMethod(String method) method
setURI(java.net.URI uri) method

`ClientRequest.Builder` class
accept(MediaType... types) method
accept(String... types) method
acceptLanguage(Locale... locales) method
acceptLanguage(String... locales) method
build(URI uri, String method) method
cookie(Cookie cookie) method
entity(Object entity) method
entity(Object entity, MediaType type) method
entity(Object entity, String type) method
header(String name, Object value) method
type(MediaType type) method
type(String type) method

- For an example of using the `handle(ClientRequest request)` method, see *getProperties() method*.

resource(String u) method

Description

This method generates the `WebResource` object from a client (`Client` object).

Syntax

```
public WebResource resource(String u)
```

Parameter

`u`
This parameter is the Web resource URI.

Return value

The method returns a `WebResource` object.

Notes

- Specify a valid non-null URI in the `u` parameter. If `null` is specified, the `NullPointerException` exception is thrown. If any other invalid value is specified, the operation is the same as that for the `create()` method of the `java.net.URI` class of Java SE. For details, see the *JDK documentation*.

resource(URI u) method

Description

This method generates a Web resource from a client (`Client` object).

Syntax

```
public WebResource resource(URI u)
```

Parameter

`u`

This parameter is the Web resource URI.

Return value

The method returns the `WebResource` object.

Notes

- Specify a valid URI in the `u` parameter. Also, use the characters defined in RFC 2396 in the URI. The `u` parameter is not automatically URI encoded. Specify a URI-encoded value in the `u` parameter as and when required.

setChunkedEncodingSize(Integer chunkSize) method

Description

This method specifies the settings to send an HTTP request entity by chunked transfer encoding in the specified chunk size.

This method functions in the same manner as when you set the `ClientConfig.PROPERTY_CHUNKED_ENCODING_SIZE` property in the property map acquired with the `getProperties()` method.

Syntax

```
public void setChunkedEncodingSize(Integer chunkSize)
```

Parameter

`chunkSize`

This parameter specifies the chunk size. If you specify `0` or less, the default value is applied.

Return value

None.

Notes

- The client operation based on the value specified in the `chunkSize` parameter is the same as that for the `setChunkedStreamingMode()` method of the `HttpURLConnection` class. For details, see the *JDK documentation*.
- The JAX-RS engine does not validate the value specified in the `chunkSize` parameter. Prior to the HTTP communication, the value will be copied as is to `HttpURLConnection`. To specify a value in the `chunkSize` parameter, follow the description of the `setChunkedStreamingMode()` method of the `HttpURLConnection` class of Java SE.
- If the client property map already contains the value specified in `ClientConfig.PROPERTY_CHUNKED_ENCODING_SIZE`, the value specified in the `chunkSize` parameter is overwritten.

- If an exception is thrown during the processing of an HTTP request or an HTTP response due to the value specified in the `chunkSize` parameter, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown.

setConnectTimeout(Integer interval) method

Description

This method sets the connection timeout in milliseconds.

This method functions in the same manner as when you set the `ClientConfig.PROPERTY_CONNECT_TIMEOUT` property in the property map acquired with the `getProperties()` method.

Syntax

```
public void setConnectTimeout(Integer interval)
```

Parameter

`interval`

This is the connection timeout value. If you specify `null` or `0`, the connection does not time out.

Return value

None.

Notes

- The client operation based on the value specified in the `interval` parameter is the same as that for the `setConnectTimeout()` method of the `HttpURLConnection` class. For details, see the *JDK documentation*.
- The JAX-RS engine does not validate the value specified in the `interval` parameter. Prior to the HTTP communication, the value will be copied as is to `HttpURLConnection`. To specify a value in the `interval` parameter, follow the description of the `setConnectTimeout()` method of the `HttpURLConnection` class of Java SE.
- If the client property map already contains the value specified in `ClientConfig.PROPERTY_CONNECT_TIMEOUT`, the value specified in the `interval` parameter is overwritten.
- If an exception is thrown during the processing of an HTTP request or an HTTP response due to the value specified in the `interval` parameter, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown.

setFollowRedirects(Boolean redirect) method

Description

This method specifies whether to automatically follow the HTTP redirector.

This method functions in the same manner as when you set the `ClientConfig.PROPERTY_FOLLOW_REDIRECTS` property in the property map acquired by using the `getProperties()` method.

Syntax

```
public void setFollowRedirects(Boolean redirect)
```

Parameter

`redirect`

If `true`, the client is automatically redirected to the coded URI of the HTTP response with the status code 300.

Return value

None.

Notes

- The client operation based on the value specified in the `redirect` parameter is the same as that for the `setInstanceFollowRedirects()` method of the `HttpURLConnection` class. For details, see the *JDK documentation*.
- The JAX-RS engine does not validate the value specified in the `redirect` parameter. Prior to the HTTP communication, the value will be copied as is to `HttpURLConnection`. To specify a value in the `redirect` parameter, follow the description of the `setInstanceFollowRedirects()` method of the `HttpURLConnection` class of Java SE.
- If the client property map already contains the value specified in `ClientConfig.PROPERTY_FOLLOW_REDIRECTS`, the value specified in the `redirect` parameter is overwritten.
- If an exception is thrown during the processing of an HTTP request or an HTTP response due to the value specified in the `redirect` parameter, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown.

setReadTimeout(Integer interval) method

Description

This method sets the read timeout in milliseconds.

This method functions in the same manner as when you set the `ClientConfig.PROPERTY_READ_TIMEOUT` property in the property map acquired by using the `getProperties()` method.

Syntax

```
public void setReadTimeout(Integer interval)
```

Parameter

`interval`

This is the timeout value for reading the property. If you specify `null` or `0`, the operation does not time out.

Return value

None.

Notes

- The client operation based on the value specified in the `interval` parameter is the same as that for the `setReadTimeout()` method of the `HttpURLConnection` class. For details, see the *JDK documentation*.

- The JAX-RS engine does not validate the value specified in the `interval` parameter. Prior to the HTTP communication, the value will be copied as is to `HttpURLConnection`. To specify a value in the `interval` parameter, follow the description of the `setReadTimeout()` method of the `HttpURLConnection` class of Java SE.
- If the client property map already contains the value specified in `ClientConfig.PROPERTY_READ_TIMEOUT`, the value specified in the `interval` parameter is overwritten.
- If an exception is thrown during the processing of an HTTP request or an HTTP response due to the value specified in the `interval` parameter, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown.

25.3 Method specifications and notes for the ClientHandlerException class

The `ClientHandlerException` class is a derived class of the `RuntimeException` class. Use the `ClientHandlerException` class within the scope of the `RuntimeException` class methods.

25.4 Method specifications and notes for the ClientRequest class

This section describes the specifications for the methods of the `ClientRequest` class and the notes on using the methods.

clone() method

Description

This method clones an HTTP request (`ClientRequest` object).

Syntax

```
public abstract ClientRequest clone()
```

Parameter

None.

Return value

The method returns the cloned HTTP request (`ClientRequest` object).

Notes

- This method clones the following entities of the `ClientRequest` object:
  - Web resource URI
  - Name of an HTTP method
  - HTTP request
  - HTTP header
- The property map of the `ClientRequest` object is not cloned. Add the property map to the cloned `ClientRequest` object as and when required.

create() method

Description

This method generates a builder (`ClientRequest.Builder` object) to set up the `ClientRequest` object.

Syntax

```
public static final ClientRequest.Builder create()
```

Parameter

None.

Return value

The method returns a builder (`ClientRequest.Builder` object).

Notes

- For an example of how to use the method, see *getProperties() method*.

getEntity() method

Description

This method acquires an HTTP request entity.

Syntax

```
public abstract Object getEntity()
```

Parameter

None.

Return value

The method returns an HTTP request entity.

Notes

- The value of the entity set with the `setEntity()` method of the `ClientRequest` class is returned.
- If no entity is set with the `setEntity()` method of the `ClientRequest` class or if you specify `null`, the method returns `null`.

getHeaders() method

Description

This method acquires an HTTP header map.

Syntax

```
public abstract MultivaluedMap<String,Object> getHeaders()
```

Parameter

None.

Return value

The method returns an HTTP header.map.

Notes

- The HTTP header set in the `ClientRequest` object is returned in the `MultivaluedMap<java.lang.String,java.lang.Object>` format. You can also set the HTTP header with the following methods of the `ClientRequest` object:`accept(MediaType... types)` method
  - `accept(String... types)` method
  - `acceptLanguage(Locale... locales)` method

- `acceptLanguage(String... locales)` method
- `cookie(Cookie cookie)` method
- `entity(Object entity, MediaType type)` method
- `entity(Object entity, String type)` method
- `header(String name, Object value)` method
- `type(MediaType type)` method
- `type(String type)` method

- If no HTTP header is set, a blank map is returned.

getHeaderValue(Object headerValue) method

Description

This method converts the specified value of an HTTP header to a character string.

Syntax

```
public static String getHeaderValue(Object headerValue)
```

Parameter

`headerValue`
This is the value of an HTTP header.

Return value

The method returns the converted character string.

Notes

- The return value is the value returned by the `toString()` method of the object specified in the parameter.

getMethod() method

Description

This method acquires an HTTP method.

Syntax

```
public abstract String getMethod()
```

Parameter

None.

Return value

The method returns an HTTP method.

Notes

- The name of the HTTP method set in the `ClientRequest` object is returned in the form of a character string. You can set an HTTP method with the following methods:
  - `setMethod(String method)` method of the `ClientRequest` class
  - `build(URI uri, String method)` method of the `ClientRequest.Builder` class
- If `null` is set in the HTTP method, the method returns `null`.

getProperties() method

Description

This method acquires a changeable property map.

Syntax

```
public abstract Map<String,Object> getProperties()
```

Parameter

None.

Return value

The method returns a property map.

Notes

- You can add or change the properties in the returned property map. For details on the properties that you can add or change, see *25.1.1 Supported properties and features*. For details on the properties, see *11.4.3 Setting the properties and features*.

An example of using the `getProperties()` method is as follows:

```
// Generate the Client object
Client client = Client.create();
// Generate an HTTP request
ClientRequest cRequest = ClientRequest.create().build(new URI("http://test.com/"), "GET");
// Set the property read time out value
cRequest.getProperties().put(ClientConfig.PROPERTY_READ_TIMEOUT, 10000);
try{
 // Receive an HTTP response as the ClientResponse object
 ClientResponse cResponse = client.handle(cRequest);
} catch(ClientHandlerException e){
 // Execute the appropriate processing
}
```

In this example, after the `ClientRequest` object is generated, the changeable property map is acquired with the `getProperties` method of the `ClientRequest` object and the property read timeout is set to 10,000 milliseconds. Then, the `handle` method of the `Client` class is invoked by specifying the `ClientRequest` object. The properties of the `ClientRequest` object are copied to the `Client` object. When an HTTP response is received as a `ClientResponse` object, if the read timeout occurs before the HTTP response is completely read, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps the `SocketTimeoutException` exception is thrown.

getPropertyAsFeature(String name) method

Description

This method acquires the Boolean property as a feature from the property map.

Syntax

```
public boolean getPropertyAsFeature(String name)
```

Parameter

`name`
This is the name of the property.

Return value

If the specified property is Boolean with the value `true`, `true` is returned, and in all other cases, `false` is returned.

Notes

None.

getPropertyAsFeature(String name, boolean defaultValue) method

Description

This method acquires a Boolean property as a feature from the property map.

Syntax

```
public boolean getPropertyAsFeature(String name,
boolean defaultValue)
```

Parameters

`name`
This is the name of the property.

`defaultValue`
This is the default value assumed when the specified property does not exist.

Return value

If the specified property is Boolean with the value `true`, `true` is returned, and in all other cases, the specified default value is returned.

Notes

None.

getURI() method

Description

This method acquires a URI of an HTTP request. A URI contains information that identifies the target resource of a request.

Syntax

```
public abstract URI getURI()
```

Parameter

None.

Return value

The method returns a URI of an HTTP request.

Notes

- The URI set in the `ClientRequest` object is returned. You can set a URI with the following methods:
  - `setURI(java.net.URI uri)` method of the `ClientRequest` class
  - `build(URI uri, String method)` method of the `ClientRequest.Builder` class
- If `null` is set in the URI, the method returns `null`.

setEntity(Object entity) method

Description

This method sets an entity.

Syntax

```
public abstract void setEntity(Object entity)
```

Parameter

`entity`
This is an entity.

Return value

None.

Notes

- For the Java types available for an HTTP request entity, see *25.15.1 Combination of Java types and MIME media types available for an HTTP request entity*.

setMethod(String method) method

Description

This method sets an HTTP method.

Syntax

```
public abstract void setMethod(String method)
```

Parameter

`method`
This is the name of an HTTP method.

Return value

None.

Notes

- Specify uppercase `GET`, `HEAD`, `POST`, `OPTIONS`, `PUT`, or `DELETE` in the `method` parameter.

setURI(java.net.URI uri) method

Description

This method sets a URI of an HTTP request. Correctly specify the information that identifies the target resource of a request in a URI.

Syntax

```
public abstract void setURI(URI uri)
```

Parameter

`uri`
This is the URI of an HTTP request.

Return value

None.

Notes

- Specify a valid URI in the `uri` parameter. Use the characters defined in RFC 2396 in the URI. A `uri` parameter is not automatically URI encoded. Specify a URI-encoded value in the `uri` parameter as and when required.

25.5 Method specifications and notes for the ClientRequest.Builder class

This section describes the specifications for the methods of the `ClientRequest.Builder` class and the notes on using the methods.

accept(MediaType... types) method

Description

This method adds a receivable MIME media type.

Syntax

```
public T accept(MediaType... types)
```

Parameter

`types`

This is an array of the receivable MIME media types.

Return value

The method returns a builder (`WebResource.Builder` object) to set up a `WebResource` object.

Notes

- A non-null value included in the `types` parameter is added in the `Accept HTTP` header. A null value is ignored and is not added in the `Accept HTTP` header.
- The value returned by the `toString()` method of the `MediaType` object is set in the `Accept HTTP` header. The JAX-RS engine does not validate the values returned by the `toString()` method. Specify a value according to the standard specifications.
- You can also use the following methods to add the value of the `Accept HTTP` header:
  - `ClientRequest.Builder` class
    `accept(String... types)` method
    `header(String name, Object value)` method
  - `WebResource` class
    `accept(MediaType... types)` method
    `accept(String... types)` method
    `header(String name, Object value)` method
  - `WebResource.Builder` class
    `accept(MediaType... types)` method
    `accept(String... types)` method
    `header(String name, Object value)` method

  The operation when no receivable MIME media type is added with these methods and the `accept(MediaType... types)` method is the same as the operation when the `Accept HTTP` header is not added in the `HttpURLConnection` object before performing HTTP communication.

accept(String... types) method

Description

This method adds a receivable MIME media type.

Syntax

```
public T accept(String... types)
```

Parameter

`types`

This is an array of the receivable MIME media types.

Return value

The method returns a builder (`WebResource.Builder` object) to set up a `WebResource` object.

Notes

- The non-null value included in the `types` parameter is added in the `Accept HTTP` header. A null value is ignored and is not added in the `Accept HTTP` header.
- The specified values are set in the `Accept HTTP` header as are. The JAX-RS engine does not validate the specified value. Specify a value according to the standard specifications.
- You can also use the following methods to add the value of the `Accept HTTP` header:
  - `ClientRequest.Builder` class
    `accept(MediaType... types)` method
    `header(String name, Object value)` method
  - `WebResource` class
    `accept(MediaType... types)` method
    `accept(String... types)` method
    `header(String name, Object value)` method
  - `WebResource.Builder` class
    `accept(MediaType... types)` method
    `accept(String... types)` method
    `header(String name, Object value)` method

  The operation when no receivable MIME media type is added with these methods and the `accept(String... types)` method is the same as the operation when the `Accept HTTP` header is not added in the `HttpURLConnection` object before performing HTTP communication.

acceptLanguage(Locale... locales) method

Description

This method adds a receivable language.

Syntax

```
public T acceptLanguage(Locale... locales)
```

Parameter

`locales`
This is an array of the receivable languages.

Return value

This method returns a builder (`WebResource.Builder` object) to set up a `WebResource` object.

Notes

- The non-null value in the `locales` parameter is added in the `Accept-Language HTTP` header. A null value is ignored and is not added in the `Accept-Language HTTP` header.
- The value returned by the `toString()` method of the `Locale` object is set in the `Accept-Language HTTP` header. The JAX-RS engine does not validate the values returned by the `toString()` method. Specify a value according to the standard specifications.
- You can also use the following methods to add the value of the `Accept-Language HTTP` header:
  - `ClientRequest.Builder` class
    `header(String name, Object value)` method
    `acceptLanguage(String... locales)` method
  - `WebResource` class
    `acceptLanguage(Locale... locales)` method
    `acceptLanguage(String... locales)` method
    `header(String name, Object value)` method
  - `WebResource.Builder` class
    `acceptLanguage(Locale... locales)` method
    `acceptLanguage(String... locales)` method
    `header(String name, Object value)` method

  The operation when no receivable language is added with these methods and the `acceptLanguage(Locale... locales)` method is the same as the operation when the `Accept-Language HTTP` header is not added in the `HttpURLConnection` object before performing HTTP communication.

acceptLanguage(String... locales) method

Description

This method adds a receivable language.

Syntax

```
public T acceptLanguage(String... locales)
```

Parameter

`locales`
This is an array of the receivable languages.

Return value

The method returns a builder (`WebResource.Builder` object) to set up a `WebResource` object.

Notes

- The non-null value in the `locales` parameter is added in the `Accept-Language HTTP` header. A null value is ignored and is not added in the `Accept-Language HTTP` header.
- The specified value is set in the `Accept-Language HTTP` header as is. The JAX-RS engine does not validate the specified value. Specify the value according to the standard specifications.
- You can also use the following methods to add the value of the `Accept-Language HTTP` header:
  - `ClientRequest.Builder` class
    `acceptLanguage(Locale... locales)` method
    `header(String name, Object value)` method
  - `WebResource` class
    `acceptLanguage(Locale... locales)` method
    `acceptLanguage(String... locales)` method
    `header(String name, Object value)` method
  - `WebResource.Builder` class
    `acceptLanguage(Locale... locales)` method
    `acceptLanguage(String... locales)` method
    `header(String name, Object value)` method

  The operation when no receivable language is added with these methods and the `acceptLanguage(String... locales)` method is the same as the operation when the `Accept-Language HTTP` header is not added in the `HttpURLConnection` object before performing HTTP communication.

build(URI uri, String method) method

Description

The method sets up the `ClientRequest` object.

Syntax

```
public ClientRequest build(URI uri,
String method)
```

Parameter

`uri`
This is the URI of an HTTP request.

`method`
This is the name of an HTTP method.

Return value

The method returns the `ClientRequest` object.

Notes

- Specify a valid URI in the `uri` parameter. Use the characters defined in RFC 2396 in the URI. A `uri` parameter is not automatically URI encoded. Specify a URI-encoded value in the `uri` parameter as and when required.
- Specify uppercase `GET`, `HEAD`, `POST`, `OPTIONS`, `PUT`, or `DELETE` in the `method` parameter.

cookie(Cookie cookie) method

Description

This method sets a Cookie.

Syntax

```
public T cookie(Cookie cookie)
```

Parameter

`cookie`
This is the Cookie to be set.

Return value

The method returns a builder (`WebResource.Builder` object) to set up the `WebResource` object.

Notes

- A non-null value is added to `Cookie HTTP` header. A null value is ignored and is not added in the `Cookie HTTP` header.
- The value returned by the `toString()` method of the `Cookie` object is set in the `Cookie HTTP` header. The JAX-RS engine does not validate the values returned by the `toString()` method. Specify the value according to the standard specifications.
- You can also use the following methods to add the value of the `Cookie HTTP` header:
  - `ClientRequest.Builder` class
    `header(String name, Object value)` method
  - `WebResource` class
    `cookie(Cookie cookie)` method
    `header(String name, Object value)` method
  - `WebResource.Builder` class
    `cookie(Cookie cookie)` method
    `header(String name, Object value)` method

  The operation when no `Cookie` is added with these methods and the `cookie(Cookie cookie)` method is the same as the operation when the `Cookie HTTP` header is not added in the `HttpURLConnection` object before performing HTTP communication.

entity(Object entity) method

Description

This method sets an HTTP request entity.

For the Java types available for an HTTP request entity, see *25.15.1 Combination of Java types and MIME media types available for an HTTP request entity*. You can use the `GenericEntity` object to specify generics in the entity.

Syntax

```
public T entity(Object entity)
```

Parameter

`entity`
This is an HTTP request entity.

Return value

The method returns a builder (`ClientRequest.Builder` object).

Notes

None.

entity(Object entity, MediaType type) method

Description

This method sets an HTTP request entity and the respective MIME media type.

For the Java types available for an HTTP request entity, see *25.15.1 Combination of Java types and MIME media types available for an HTTP request entity*. You can use the `GenericEntity` object to specify generics in the entity.

Syntax

```
public T entity(Object entity,
MediaType type)
```

Parameters

`entity`
This is an HTTP request entity.

`type`
This is a MIME media type.

Return value

The method returns a builder (`ClientRequest.Builder` object).

Notes

- If `null` is specified in the `type` parameter, the value is ignored and is not set in the `Content-Type HTTP` header.
- The value returned by the `toString()` method of the `MediaType` object is set in the `Content-Type HTTP` header. The JAX-RS engine does not validate the values returned by the `toString()` method. Specify the value according to the standard specifications.
- If the value of the `Content-Type HTTP` header is already set by using the following methods, the value is overwritten by the value of the `type` parameter:
  - `ClientRequest.Builder` class
    `entity(Object entity, String type)` method
    `header(String name, Object value)` method
    `type(MediaType type)` method
    `type(String type)` method
  - `WebResource` class

`entity(Object entity, MediaType type)` method
`entity(Object entity, String type)` method
`header(String name, Object value)` method
`type(MediaType type)` method
`type(String type)` method

- `WebResource.Builder` class
  `entity(Object entity, MediaType type)` method
  `entity(Object entity, String type)` method
  `header(String name, Object value)` method
  `type(MediaType type)` method
  `type(String type)` method

The operation when the MIME media type is not set with these methods and the `entity(Object entity, MediaType type)` method is the same as the operation when the `Content-Type HTTP` header is not set in the `HttpURLConnection` object before performing HTTP communication.

entity(Object entity, String type) method

Description

This method sets an HTTP request entity and the respective MIME media type.

For the Java types available for an HTTP request entity, see *25.15.1 Combination of Java types and MIME media types available for an HTTP request entity*. You can use the `GenericEntity` object to specify generics in the entity.

Syntax

```
public T entity(Object entity,
String type)
```

Parameters

`entity`
This is an HTTP request entity.

`type`
This is a MIME media type.

Return value

The method returns a builder (`ClientRequest.Builder` object).

Notes

- If `null` or an empty string is set in the `type` parameter, the `IllegalArgumentException` exception is thrown.
- The value returned by the `toString()` method of the `MediaType` object set up by specifying the `type` parameter in the parameter of the `valueOf(String)` static method of the `MediaType` class is set in the `Content-Type HTTP` header. The JAX-RS engine does not validate the specified parameter value. Specify the value according to the standard specifications.
- If the value of the `Content-Type HTTP` header is already set by using the following methods, the value is overwritten by the value of the `type` parameter:
  - `ClientRequest.Builder` class

`entity(Object entity, MediaType type)` method
`header(String name, Object value)` method
`type(MediaType type)` method
`type(String type)` method

- `WebResource` class
  `entity(Object entity, MediaType type)` method
  `entity(Object entity, String type)` method
  `header(String name, Object value)` method
  `type(MediaType type)` method
  `type(String type)` method
- `WebResource.Builder` class
  `entity(Object entity, MediaType type)` method
  `entity(Object entity, String type)` method
  `header(String name, Object value)` method
  `type(MediaType type)` method
  `type(String type)` method

The operation when the MIME media type is not set with these methods and the `entity(Object entity, String type)` method is the same as the operation when the `Content-Type HTTP` header is not set in the `HttpURLConnection` object before performing HTTP communication.

header(String name, Object value) method

Description

This method adds an HTTP header.

Syntax

```
public T header(String name,
Object value)
```

Parameters

`name`
This is the name of an HTTP header.

`value`
This is the value of an HTTP header.

Return value

The method returns a builder (`ClientRequest.Builder` object).

Notes

- If the value of both `name` and `value` parameters is `null`, the method is not invoked.
- Even if the value of the `name` parameter is not null, but if the value of the `value` parameter is `null`, the method is not invoked.
- If the value of the `name` parameter is `null` and the value of the `value` parameter is not null, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps the `NullPointerException` exception is thrown.

- You cannot set the HTTP headers `Content-Length`, `Connection`, or `Host` with this method. If you specify these headers in the `name` parameter, the method is not invoked even if the value of the `value` parameter is not null. Note that `HttpURLConnection` sets each of the HTTP headers.
- The value returned by the `toString()` method of a non-null object specified with the `value` parameter is set as the value of the HTTP header specified with the `name` parameter. The JAX-RS engine does not validate the values returned by the `toString()` method. Specify the value in the `value` parameter according to the standard specifications.
- You can also use the following methods to add the HTTP headers `accept`, `acceptLanguage`, and `cookie`:
  - `ClientRequest.Builder` class
    `accept(MediaType... types)` method
    `accept(String... types)` method
    `acceptLanguage(Locale... locales)` method
    `acceptLanguage(String... locales)` method
    `cookie(Cookie cookie)` method
  - `WebResource` class
    `accept(MediaType... types)` method
    `accept(String... types)` method
    `acceptLanguage(Locale... locales)` method
    `acceptLanguage(String... locales)` method
    `cookie(Cookie cookie)` method
  - `WebResource.Builder` class
    `accept(MediaType... types)` method
    `accept(String... types)` method
    `acceptLanguage(Locale... locales)` method
    `acceptLanguage(String... locales)` method
    `cookie(Cookie cookie)` method

  The operation when none of the HTTP headers is added with these methods and the `header(String name, Object value)` method is the same as the operation when none of the HTTP headers is added in the `HttpURLConnection` object.
- If the `Content-Type HTTP` header is already set by using the following methods, the value is overwritten by the value of the `value` parameter:
  - `ClientRequest.Builder` class
    `entity(Object entity, MediaType type)` method
    `entity(Object entity, String type)` method
    `type(MediaType type)` method
    `type(String type)` method
  - `WebResource` class
    `entity(Object entity, MediaType type)` method
    `entity(Object entity, String type)` method
    `header(String name, Object value)` method
    `type(MediaType type)` method
    `type(String type)` method
  - `WebResource.Builder` class
    `entity(Object entity, MediaType type)` method
    `entity(Object entity, String type)` method
    `header(String name, Object value)` method
    `type(MediaType type)` method
    `type(String type)` method

The operation when the MIME media type is not set with these methods and the `header(String name, Object value)` method is the same as the operation when the `Content-Type HTTP` header is not set in the `HttpURLConnection` object before performing HTTP communication.

type(MediaType type) method

Description

This method sets a MIME media type.

Syntax

```
public T type(MediaType type)
```

Parameter

`type`

This is a MIME media type.

Return value

The method returns a builder (`ClientRequest.Builder` object).

Notes

- The non-null value specified in the `type` parameter is set in the `Content-Type HTTP` header. A null value is ignored and is not set in the `Content-Type HTTP` header.
- The value returned by the `toString()` method of the `MediaType` object is set in the `Content-Type HTTP` header. The JAX-RS engine does not validate the values returned by the `toString()` method. Specify the value according to the standard specifications.
- If the `Content-Type HTTP` header is already set by using the following methods, the value is overwritten:
  - `ClientRequest.Builder` class
    `entity(Object entity, MediaType type)` method
    `entity(Object entity, String type)` method
    `header(String name, Object value)` method
    `type(String type)` method
  - `WebResource` class
    `entity(Object entity, MediaType type)` method
    `entity(Object entity, String type)` method
    `header(String name, Object value)` method
    `type(MediaType type)` method
    `type(String type)` method
  - `WebResource.Builder` class
    `entity(Object entity, MediaType type)` method
    `entity(Object entity, String type)` method
    `header(String name, Object value)` method
    `type(MediaType type)` method
    `type(String type)` method

  The operation when the MIME media type is not set with these methods and the `type(MediaType type)` method is the same as the operation when the `Content-Type HTTP` header is not set in the `HttpURLConnection` object before performing HTTP communication.

type(String type) method

Description

This method sets a MIME media type.

Syntax

```
public T type(String type)
```

Parameter

`type`

This is a MIME media type.

Return value

The method returns a builder (`ClientRequest.Builder` object).

Notes

- If `null` or an empty string is set in the `type` parameter, the `IllegalArgumentException` exception is thrown.
- The value returned by the `toString()` method of the `MediaType` object set up by specifying the `type` parameter in the parameter of the `valueOf(String)` static method of the `MediaType` class is set in the `Content-Type HTTP` header. The JAX-RS engine does not validate the values specified in the `type` parameter. Specify the value according to the standard specifications.
- If the `Content-Type HTTP` header is already set by using the following methods, the value is overwritten by the value of the `type` parameter:
  - `ClientRequest.Builder` class
    `entity(Object entity, MediaType type)` method
    `entity(Object entity, String type)` method
    `header(String name, Object value)` method
    `type(MediaType type)` method
  - `WebResource` class
    `entity(Object entity, MediaType type)` method
    `entity(Object entity, String type)` method
    `header(String name, Object value)` method
    `type(MediaType type)` method
    `type(String type)` method
  - `WebResource.Builder` class
    `entity(Object entity, MediaType type)` method
    `entity(Object entity, String type)` method
    `header(String name, Object value)` method
    `type(MediaType type)` method
    `type(String type)` method

  The operation when the MIME media type is not set with these methods and the `type(String type)` method is the same as the operation when the `Content-Type HTTP` header is not set in the `HttpURLConnection` object before performing HTTP communication.

25.6 Method specifications and notes for the ClientResponse class

This section describes the specifications for the methods of the `ClientResponse` class and the notes on using the methods.

bufferEntity() method

Description

This method buffers an entity. The entire data is read from the stream input in an HTTP response entity and stored in the memory. The input stream is closed after buffering.

Syntax

```
public void bufferEntity()
throws ClientHandlerException
```

Parameter

None.

Return value

`ClientHandlerException`

This exception is thrown if a problem occurs during processing. The error message KDJJ18888-E is output to the log.

Notes

None.

close() method

Description

This method ends an HTTP response. The input stream of an entity is closed.

Syntax

```
public void close()
throws ClientHandlerException
```

Parameter

None.

Return value

`ClientHandlerException`

This exception is thrown if a problem occurs during processing. The error message KDJJ18888-E is output to the log.

Notes

None.

getAllow() method

Description

This method acquires a set of HTTP methods supported by resources from the `Allow HTTP` header. Note that you can acquire the `Allow HTTP` header for an `HTTP OPTIONS` request.

Syntax

```
public Set<String> getAllow()
```

Parameter

None.

Return value

The method returns a set of HTTP methods supported by resources. All the methods are output in the upper case.

Notes

- This method returns an empty set if no `Allow HTTP` header exists in the HTTP response.

getClient() method

Description

This method acquires a client (`Client` object).

Syntax

```
public Client getClient()
```

Parameter

None.

Return value

The method returns a client (`Client` object).

Notes

- The `Client` object returned by the method is the `Client` object that generated the `ClientResponse` object.

getClientResponseStatus() method

Description

This method acquires a status code.

Syntax

```
public ClientResponse.Status getClientResponseStatus()
```

Parameter

None.

Return value

The method returns a status code.

If the status code of an HTTP response is compliant with the constant value defined in the `Response.Status` enumerated type, the method returns that `Response.Status` enumerated type constant value. If the status code is not compliant, the method returns `null`.

Notes

None.

getCookies() method

Description

This method acquires a list of Cookies. The method returns a list of the `NewCookie` objects when a value is set in the `Set-Cookie HTTP` header of an HTTP response.

Syntax

```
public List<NewCookie> getCookies()
```

Parameter

None.

Return value

The method returns a list of Cookies (`NewCookie` object).

Notes

- The method returns a blank list if no `Set-Cookie HTTP` header exists in the HTTP response.

getEntity(Class<T> c) method

Description

This method acquires an entity of an HTTP response. If the acquired entity is not an instance of the class that implemented the `Closeable` interface, the input stream is closed.

Syntax

```
public <T> T getEntity(Class<T> c)
throws ClientHandlerException,
UniformInterfaceException
```

Parameter

`c`

This is an entity type.

Return value

The method returns the object of the specified type.

Exception

`ClientHandlerException`

This exception is thrown if a problem occurs during processing. The error message KDJJ18888-E is output to the log.

`UniformInterfaceException`

This exception is thrown if the HTTP status code is 204 No Content. The error message KDJJ18888-E is output to the log.

Notes

- For the Java types available for an HTTP response entity, see *25.15.2 Combination of Java types and MIME media types available for an HTTP response entity*.
- You can invoke the method only once. The operation is not guaranteed if the method is invoked two or more times.

getEntity(GenericType<T> gt) method

Description

This method acquires an entity of an HTTP response. If the acquired entity is not an instance of the class that implemented the `Closeable` interface, the input stream is closed.

Syntax

```
public <T> T getEntity(GenericType<T> gt)
throws ClientHandlerException,
UniformInterfaceException
```

Parameter

`gt`

This is a `GenericType` object that represents the entity type.

Return value

The method returns the object of the type expressed by the specified `GenericType` object.

Exception

`ClientHandlerException`

This exception is thrown if a problem occurs during processing. The error message KDJJ18888-E is output to the log.

`UniformInterfaceException`

This exception is thrown if the HTTP status code is 204 No Content. The error message KDJJ18888-E is output to the log.

Notes

- For the Java types available for an HTTP response entity, see *25.15.2 Combination of Java types and MIME media types available for an HTTP response entity*.
- You can invoke the method only once. The operation is not guaranteed if the method is invoked two or more times.

getEntityInputStream() method

Description

This method acquires an input stream of an HTTP response.

Syntax

```
public InputStream getEntityInputStream()
```

Parameter

None.

Return value

The method returns an input stream of an HTTP response.

Notes

None.

getEntityTag() method

Description

This method acquires the value of an `ETag HTTP` header.

Syntax

```
public EntityTag getEntityTag()
```

Parameter

None.

Return value

The method returns the value of the `ETag HTTP` header.

Notes

- The method returns `null` if the HTTP response does not contain the `ETag HTTP` header or if `HttpURLConnection` returns `null` as the value of the `ETag HTTP` header.
- If the value of the `ETag HTTP` header is an empty string or if the value does not comply with the standard specifications, the `IllegalArgumentException` exception is thrown.

getHeaders() method

Description

This method acquires the HTTP headers map of an HTTP response.

Syntax

```
public MultivaluedMap<String,String> getHeaders()
```

Parameter

None.

Return value

The method returns the HTTP headers map of an HTTP response.

Notes

- The method returns a blank map if the HTTP response does not contain HTTP headers.

getLanguage() method

Description

This method acquires the value of the `Content-Language HTTP` header.

Syntax

```
public String getLanguage()
```

Parameter

None.

Return value

The method returns the value of the `Content-Language HTTP` header.

Notes

- The method returns `null` if the HTTP response does not contain the `Content-Language HTTP` header or if `HttpURLConnection` returns `null` as the value of the `Content-Language HTTP` header.

getLastModified() method

Description

This method acquires the value of the `Last-Modified HTTP` header.

Syntax

```
public Date getLastModified()
```

Parameter

None.

Return value

The method returns the value of the `Last-Modified HTTP` header.

Notes

- The method returns `null` if the HTTP response does not contain the `Last-Modified HTTP` header or if `HttpURLConnection` returns `null` as the value of the `Last-Modified HTTP` header.
- If the value of the `Last-Modified HTTP` header is an empty string or if the value does not comply with the following formats, the `IllegalArgumentException` exception is thrown.
  - EEE, dd MMM yyyy HH:mm:ss zzz
  - EEEE, dd-MMM-yy HH:mm:ss zzz
  - EEE MMM d HH:mm:ss yyyy

  For each pattern character in the Date format, especially the locale-dependant 'E' and 'M', see the information on the `SimpleDateFormat` class of the Java SE specifications.

getLength() method

Description

This method acquires the value of the `Content-Length HTTP` header.

Syntax

```
public int getLength()
```

Parameter

None.

Return value

The method returns an integer value when the value of the `Content-Length HTTP` header is a valid number.

Notes

- If the HTTP response does not contain the `Content-Length HTTP` header, if `HttpURLConnection` returns `null` as the value of the `Content-Length HTTP` header, or if the value of the `Content-Length HTTP` header is an invalid integer, the method returns `-1`.

getLocation() method

Description

This method acquires the value of the `Location HTTP` header.

Syntax

```
public URI getLocation()
```

Parameter

None.

Return value

The method returns the value of the `Location HTTP` header.

Notes

- The method returns `null` if the HTTP response does not contain the `Location HTTP` header or if `HttpURLConnection` returns `null` as the value of the `Location HTTP` header.
- If the value of the `Location HTTP` header is an invalid URI, the `IllegalArgumentException` exception is thrown.

getResponseDate() method

Description

This method acquires the value of the `Date HTTP` header.

Syntax

```
public Date getResponseDate()
```

Parameter

None.

Return value

The method returns the value of the `Date HTTP` header.

Notes

- The method returns `null` if the HTTP response does not contain the `Date HTTP` header or if `HttpURLConnection` returns `null` as the value of the `Date HTTP` header.

- If the value of the `Date HTTP` header is an empty string or if the value is not compliant with the following date formats, the `IllegalArgumentException` exception is thrown:
  - EEE, dd MMM yyyy HH:mm:ss zzz
  - EEEE, dd-MMM-yy HH:mm:ss zzz
  - EEE MMM d HH:mm:ss yyyy

  For each pattern character in the Date format, especially the locale-dependant 'E' and 'M', see the information on the `SimpleDateFormat` class of the Java SE specifications.

getStatus() method

Description

This method acquires the HTTP status code.

Syntax

```
public int getStatus()
```

Parameter

None.

Return value

This method returns an integer as the HTTP status code.

Notes

None.

getType() method

Description

This method acquires the value of the `Content-Type HTTP` header.

Syntax

```
public MediaType getType()
```

Parameter

None.

Return value

The method returns the value of the `Content-Type HTTP` header.

Notes

- The method returns `null` if the HTTP response does not contain the `Content-Type HTTP` header or if `HttpURLConnection` returns `null` as the value of the `Content-Type HTTP` header.

- If the `MediaType` object cannot be set up from the value of the `Content-Type HTTP` header, the `IllegalArgumentException` exception is thrown.

hasEntity() method

Description

This method checks whether an entity exists.

Syntax

```
public boolean hasEntity()
```

Parameter

None.

Return value

The method returns `true` if an entity exists in the HTTP response.

Notes

- The method returns `false` if an HTTP response is acquired with an HTTP HEAD request.
- The method returns `true` if `HttpURLConnection` returns a number greater than `0` or `-1` as the value of the `Content-Length HTTP` header. The method returns `false` if `HttpURLConnection` returns any other value.
- Generally, `HttpURLConnection` returns an integer greater than `0` if the encoding is not a chunked transfer encoding and an entity exists in the HTTP response, and `-1` if the encoding is a chunked transfer encoding. For details on chunked transfer encoding, see RFC 2616.
- In the following cases, the operations are not guaranteed because the HTTP response is invalid:
  - If the encoding is a chunked transfer encoding and the HTTP response contains the `Content-Length HTTP` header
  - If the encoding is not a chunked transfer encoding and the value of the `Content-Length HTTP` header is not an appropriate value based on the HTTP response entity.

25.7 Enumerated constants of the ClientResponse.Status class and specifications for the methods

This section describes the enumerated constants of the `ClientResponse.Status` class and specifications for the class methods.

Enumerated constants of the ClientResponse.Status class

The `ClientResponse.Status` class defines the following enumerated constants as the HTTP status codes. The following table lists and describes the syntax and HTTP status code of each constant.

Table 25-5: Enumerated constants defined with the ClientResponse.Status class

No.	Constant name	Syntax	HTTP status code
1	OK	public static final ClientResponse.Status OK	200 OK
2	CREATED	public static final ClientResponse.Status CREATED	201 Created
3	ACCEPTED	public static final ClientResponse.Status ACCEPTED	202 Accepted
4	NON_AUTHORITIVE_INFORMATION	public static final ClientResponse.Status NON_AUTHORITIVE_INFORMATION	203 Non-Authoritative Information
5	NO_CONTENT	public static final ClientResponse.Status NO_CONTENT	204 No Content
6	RESET_CONTENT	public static final ClientResponse.Status RESET_CONTENT	205 Reset Content
7	PARTIAL_CONTENT	public static final ClientResponse.Status PARTIAL_CONTENT	206 Partial Content
8	MOVED_PERMANENTLY	public static final ClientResponse.Status MOVED_PERMANENTLY	301 Moved Permantly
9	FOUND	public static final ClientResponse.Status FOUND	302 Found
10	SEE_OTHER	public static final ClientResponse.Status SEE_OTHER	303 See Other
11	NOT_MODIFIED	public static final ClientResponse.Status NOT_MODIFIED	304 Not Modified
12	USE_PROXY	public static final ClientResponse.Status USE_PROXY	305 Use Proxy
13	TEMPORARY_REDIRECT	public static final ClientResponse.Status TEMPORARY_REDIRECT	307 Temporary Redirect
14	BAD_REQUEST	public static final ClientResponse.Status BAD_REQUEST	400 Bad Request
15	UNAUTHORIZED	public static final ClientResponse.Status UNAUTHORIZED	401 Unauthorized
16	PAYMENT_REQUIRED	public static final ClientResponse.Status PAYMENT_REQUIRED	402 Payment Required

No.	Constant name	Syntax	HTTP status code
17	FORBIDDEN	public static final ClientResponse.Status FORBIDDEN	403 Forbidden
18	NOT_FOUND	public static final ClientResponse.Status NOT_FOUND	404 Not Found
19	METHOD_NOT_ALLOWED	public static final ClientResponse.Status METHOD_NOT_ALLOWED	405 Method Not Allowed
20	NOT_ACCEPTABLE	public static final ClientResponse.Status NOT_ACCEPTABLE	406 Not Acceptable
21	PROXY_AUTHENTICATION_REQUIRED	public static final ClientResponse.Status PROXY_AUTHENTICATION_REQUIRED	407 Proxy Authentication Required
22	REQUEST_TIMEOUT	public static final ClientResponse.Status REQUEST_TIMEOUT	408 Request Timeout
23	CONFLICT	public static final ClientResponse.Status CONFLICT	409 Conflict
24	GONE	public static final ClientResponse.Status GONE	410 Gone
25	LENGTH_REQUIRED	public static final ClientResponse.Status LENGTH_REQUIRED	411 Length Required
26	PRECONDITION_FAILED	public static final ClientResponse.Status PRECONDITION_FAILED	412 Precondition Failed
27	REQUEST_ENTITY_TOO_LARGE	public static final ClientResponse.Status REQUEST_ENTITY_TOO_LARGE	413 Request Entity Too Large
28	REQUEST_URI_TOO_LONG	public static final ClientResponse.Status REQUEST_URI_TOO_LONG	414 Request-URI Too Long
29	UNSUPPORTED_MEDIA_TYPE	public static final ClientResponse.Status UNSUPPORTED_MEDIA_TYPE	415 Unsupported Media Type
30	REQUESTED_RANGE_NOT_SATIFIABLE	public static final ClientResponse.Status REQUESTED_RANGE_NOT_SATIFIABLE	416 Requested Range Not Satisfiable
31	EXPECTATION_FAILED	public static final ClientResponse.Status EXPECTATION_FAILED	417 Expectation Failed
32	INTERNAL_SERVER_ERROR	public static final ClientResponse.Status INTERNAL_SERVER_ERROR	500 Internal Server Error
33	NOT_IMPLEMENTED	public static final ClientResponse.Status NOT_IMPLEMENTED	501 Not Implemented
34	BAD_GATEWAY	public static final ClientResponse.Status BAD_GATEWAY	502 Bad Gateway
35	SERVICE_UNAVAILABLE	public static final ClientResponse.Status SERVICE_UNAVAILABLE	503 Service Unavailable
36	GATEWAY_TIMEOUT	public static final ClientResponse.Status GATEWAY_TIMEOUT	504 Gateway Timeout
37	HTTP_VERSION_NOT_SUPPORTED	public static final ClientResponse.Status HTTP_VERSION_NOT_SUPPORTED	505 HTTP Version Not Supported

StatusCode(int statusCode) method

Description

This method converts the status codes expressed in numbers into the corresponding `ClientResponse.Status` object.

Syntax

```
public static ClientResponse.Status fromStatusCode(int statusCode)
```

Parameter

`statusCode`
This is the status code expressed with a numeric value.

Return value

The method returns a `ClientResponse.Status` object corresponding to the status code and returns `null` if the corresponding object does not exist.

Notes

None.

getFamily() method

Description

This method acquires a category of the status code.

Syntax

```
public Response.Status.Family getFamily()
```

Parameter

None.

Return value

This method returns the category of the status code.

Notes

None.

getReasonPhrase() method

Description

This method acquires a description string (`Reason Phrase`).

Syntax

```
public String getReasonPhrase()
```

Parameter

None.

Return value

The method returns the description string (`Reason Phrase`).

Notes

None.

getStatusCode() method

Description

This method acquires the corresponding status code.

Syntax

```
public int getStatusCode()
```

Parameter

None.

Return value

The method returns the status code.

Notes

None.

toString() method

Description

This method acquires a description string (`Reason Phrase`).

Syntax

```
public String toString()
```

Parameter

None.

Return value

The method returns the description string (`Reason Phrase`).

Notes

None.

valueOf(String name) method

Description

This method acquires the enumerated constant of the specified name.

Accurately specify an identifier for the constant declared with the enumerated type. Note that extra white spaces are not allowed.

Syntax

```
public static ClientResponse.Status valueOf(String name)
```

Parameter

`name`

This is the name of the enumerated constant to be acquired.

Return value

The method returns the enumerated constant of the specified name.

Exception

`IllegalArgumentException`

This exception is thrown if the enumerated constant of the specified name does not exist.

`NullPointerException`

This exception is thrown if `null` is specified in the `name` parameter.

Notes

None.

values() method

Description

This method returns an array of enumerated constants in the order in which they are declared with the enumerated type. You can use this method for the following type of repeated processing:

```
for (ClientResponse.Status c : ClientResponse.Status.values())
 System.out.println(c);
```

Syntax

```
public static ClientResponse.Status[] values()
```

Parameter

None.

Return value

The method returns an array of enumerated constants in the order in which they are declared with the enumerated type.

Notes

None.

25.8 Constructor and method specifications and notes for the GenericType class

This section describes the specifications for the constructors and methods of the `GenericType` class and the notes on using the constructors and methods. The `GenericType` class stores the type information of generics in which the `type` parameter is resolved.

GenericType() constructor

Description

This constructor sets up a new `GenericType` object. The type information of generics in which the target type parameter is resolved and the class declaring the generics are acquired from the type parameter `T`. The user must create a subclass (in most cases, an anonymous subclass is created) because the scope of the constructor is protected.

Syntax

```
protected GenericType()
```

Parameter

None.

Return value

None.

Notes

None.

GenericType(Type genericType) constructor

Description

This constructor sets up a new `GenericType` object. The types of generics in which the target type parameter is resolved and the class declaring the generics are acquired from the `genericType` parameter.

Syntax

```
public GenericType(Type genericType)
```

Parameter

`genericType`

This is the `java.lang.Type` object that expresses the generics in which the type parameter is resolved.

Return value

None.

Exception

`IllegalArgumentException`

This exception is thrown if the `genericType` parameter is `null`, or if the `genericType` parameter is neither the `java.lang.Class` object nor the `java.lang.reflect.ParameterizedType` object where the type declaring the parameter type is a class.

Notes

None.

getRawClass() method

Description

This method acquires the class that declares the generics in which the type parameter to be stored is resolved.

Syntax

```
public final Class<T> getRawClass()
```

Parameter

None.

Return value

The method returns the class declaring the generics in which the type parameter to be stored is resolved.

Notes

None.

getType() method

Description

This method acquires the generics type in which the type parameter to be stored is resolved.

Syntax

```
public final Type getType()
```

Parameter

None.

Return value

The method returns the generics type in which the type parameter to be stored is resolved.

Notes

None.

25.9 Method specifications and notes for the UniformInterfaceException class

This section describes the specifications for the methods of the `UniformInterfaceException` class and the notes on using the methods. Use the `UniformInterfaceException` class within the scope of the methods described in this section and the parent class methods.

getResponse() method

Description

This method acquires the `ClientResponse` object related to exceptions.

Syntax

```
public ClientResponse getResponse()
```

Parameter

None.

Return value

The method returns the `ClientResponse` object.

Notes

None.

25.10 Method specifications and notes for the WebResource class

This section describes the specifications for the methods of the `WebResource` class and the notes on using the methods.

accept(MediaType... types) method

Description

This method adds a receivable MIME media type.

Syntax

```
public WebResource.Builder accept(MediaType... types)
```

Parameter

`types`

This is an array of the receivable MIME media types.

Return value

The method returns a builder (`WebResource.Builder` object) to set up the `WebResource` object.

Notes

- The non-null value included in the `types` parameter is added in the `Accept HTTP` header. A null value is ignored and is not added in the `Accept HTTP` header.
- The value returned by the `toString()` method of the `MediaType` object is set in the `Accept HTTP` header The JAX-RS engine does not validate the values returned by the `toString()` method. Specify the value according to the standard specifications.
- You can also use the following methods to add the value of the `Accept HTTP` header:
  - `ClientRequest.Builder` class
    `accept(MediaType... types)` method
    `accept(String... types)` method
    `header(String name, Object value)` method
  - `WebResource` class
    `accept(String... types)` method
    `header(String name, Object value)` method
  - `WebResource.Builder` class
    `accept(MediaType... types)` method
    `accept(String... types)` method
    `header(String name, Object value)` method

  The operation when no receivable MIME media type is added with these methods and the `accept(MediaType... types)` method is the same as the operation when the `Accept HTTP` header is not added in the `HttpURLConnection` object before performing HTTP communication.

accept(String... types) method

Description

This method adds a receivable MIME media type.

Syntax

```
public WebResource.Builder accept(String... types)
```

Parameter

`types`

This is an array of the receivable MIME media types.

Return value

The method returns a builder (`WebResource.Builder` object) to set up the `WebResource` object.

Notes

- The non-null value included in the `types` parameter is added in the `Accept HTTP` header. A null value is ignored and is not added in the `Accept HTTP` header.
- The specified value is set as is in the `Accept HTTP` header. The JAX-RS engine does not validate the specified value. Specify the value according to the standard specifications.
- You can also use the following methods to add the value of the `Accept HTTP` header:
  - `ClientRequest.Builder` class
    `accept(MediaType... types)` method
    `accept(String... types)` method
    `header(String name, Object value)` method
  - `WebResource` class
    `accept(MediaType... types)` method
    `header(String name, Object value)` method
  - `WebResource.Builder` class
    `accept(MediaType... types)` method
    `accept(String... types)` method
    `header(String name, Object value)` method

  The operation when no receivable MIME media type is added with these methods and the `accept(String... types)` method is the same as the operation when the `Accept HTTP` header is not added in the `HttpURLConnection` object before performing HTTP communication.

acceptLanguage(Locale... locales) method

Description

This method adds a receivable language.

Syntax

```
public WebResource.Builder acceptLanguage(Locale... locales)
```

Parameter

`locales`
This is an array of the receivable languages.

Return value

The method returns a builder (`WebResource.Builder` object) to set up the `WebResource` object.

Notes

- The non-null value in the `locales` parameter is added in the `Accept-Language HTTP` header. A null value is ignored and is not added in the `Accept-Language HTTP` header.
- The value returned by the `toString()` method of the `Locale` object is set in the `Accept-Language HTTP` header. The JAX-RS engine does not validate the values returned by the `toString()` method. Specify the value according to the standard specifications.
- You can also use the following methods to change the value of the `Accept-Language HTTP` header:
  - `ClientRequest.Builder` class
    `acceptLanguage(Locale... locales)` method
    `acceptLanguage(String... locales)` method
    `header(String name, Object value)` method
  - `WebResource` class
    `acceptLanguage(String... locales)` method
    `header(String name, Object value)` method
  - `WebResource.Builder` class
    `acceptLanguage(Locale... locales)` method
    `acceptLanguage(String... locales)` method
    `header(String name, Object value)` method

  The operation when no receivable language is added with these methods and the `acceptLanguage(Locale... locales)` method is the same as the operation when the `Accept-Language HTTP` header is not set in the `HttpURLConnection` object before performing HTTP communication.

acceptLanguage(String... locales) method

Description

This method adds a receivable language.

Syntax

```
public WebResource.Builder acceptLanguage(String... locales)
```

Parameter

`locales`
This is an array of the receivable languages.

Return value

The method returns a builder (`WebResource.Builder` object) to set up the `WebResource` object.

Notes

- The non-null value in the `locales` parameter is added in the `Accept-Language HTTP` header. A null value is ignored and is not added in the `Accept-Language HTTP` header.
- The specified value is set as is in the `Accept-Language HTTP` header. The JAX-RS engine does not validate the specified value. Specify the value according to the standard specifications.
- You can also use the following methods to add the value of the `Accept-Language HTTP` header:
  - `ClientRequest.Builder` class
    `acceptLanguage(Locale... locales)` method
    `acceptLanguage(String... locales)` method
    `header(String name, Object value)` method
  - `WebResource` class
    `acceptLanguage(Locale... locales)` method
    `header(String name, Object value)` method
  - `WebResource.Builder` class
    `acceptLanguage(Locale... locales)` method
    `acceptLanguage(String... locales)` method
    `header(String name, Object value)` method

  The operation when no receivable language is added with these methods and the `acceptLanguage(String... locales)` method is the same as the operation when the `Accept-Language HTTP` header is not added in the `HttpURLConnection` object before performing HTTP communication.

cookie(Cookie cookie) method

Description

This method sets a Cookie.

Syntax

```
public WebResource.Builder cookie(Cookie cookie)
```

Parameter

`cookie`
This is the Cookie to be specified.

Return value

The method returns a builder (`WebResource.Builder` object) to set up the `WebResource` object.

Notes

- A non-null value is added in the `Cookie HTTP` header. A null value is ignored and is not added in the `Cookie HTTP` header.
- The value returned by the `toString()` method of the `Cookie` object is set in the `Cookie HTTP` header. The JAX-RS engine does not validate the values returned by the `toString()` method. Specify the value according to the standard specifications.
- You can also use the following methods to add the value of the `Cookie HTTP` header:
  - `ClientRequest.Builder` class

`cookie(Cookie cookie)` method
`header(String name, Object value)` method

- `WebResource` class
  `header(String name, Object value)` method
- `WebResource.Builder` class
  `cookie(Cookie cookie)` method
  `header(String name, Object value)` method

The operation when no Cookie is added with these methods and the `cookie(Cookie cookie)` method is the same as the operation when the `Cookie HTTP` header is not added in the `HttpURLConnection` object before performing HTTP communication.

delete() method

Description

This method invokes the `HTTP DELETE` method. The `delete()` method does not include any entities in the request and does not receive any response entities.

An entity is ignored if the status code is below 300 and the entity is included in a response.

Syntax

```
public void delete()
throws UniformInterfaceException
```

Parameter

None.

Return value

None.

Exception

`UniformInterfaceException`
This exception is thrown if the HTTP response status code is 300 or more. The error message KDJJ18888-E is output to the log.

Notes

- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown.

delete(Class<T> c) method

Description

This method invokes the `HTTP DELETE` method. The `delete(Class<T> c)` method does not include any entities in the request. The `delete(Class<T> c)` method, however, receives response entities.

Syntax

```
public <T> T delete(Class<T> c)
throws UniformInterfaceException
```

Parameter

`c`

This is the type of an HTTP response entity.

Return value

The `delete(Class<T> c)` method returns the object of the specified type.

Exception

`UniformInterfaceException`

This exception is thrown if any of the following conditions is fulfilled. The error message KDJJ18888-E is output to the log.

- If the status code of the HTTP response is 204
- If the status code of the HTTP response is 300 or more and the `c` parameter is not of the `ClientResponse` type

Notes

- For the Java types available for an HTTP response entity, see *25.15.2 Combination of Java types and MIME media types available for an HTTP response entity*.
- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown.

delete(Class<T> c, Object requestEntity) method

Description

This method invokes the `HTTP DELETE` method. The `delete(Class<T> c, Object requestEntity)` method includes an entity in the request and also receives a response entity.

Syntax

```
public <T> T delete(Class<T> c,
Object requestEntity)
throws UniformInterfaceException
```

Parameters

`c`

This is the type of an HTTP response entity.

`requestEntity`

This is an HTTP request entity.

Return value

The `delete(Class<T> c, Object requestEntity)` method returns the object of the specified type.

Exception

`UniformInterfaceException`

This exception is thrown if any of the following conditions is fulfilled. The error message KDJJ18888-E is output to the log.

- If the status code of the HTTP response is 204
- If the status code of the HTTP response is 300 or more and the `c` parameter is not of the `ClientResponse` type

Notes

- For the Java types available for an HTTP request entity, see *25.15.1 Combination of Java types and MIME media types available for an HTTP request entity*.
- For the Java types available for an HTTP response entity, see *25.15.2 Combination of Java types and MIME media types available for an HTTP response entity*.
- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown.

delete(GenericType<T> gt) method

Description

This method invokes the `HTTP DELETE` method. The `delete(GenericType<T> gt)` method does not include any entities in the request. The `delete(GenericType<T> gt)` method, however, receives a response entity.

Syntax

```
public <T> T delete(GenericType<T> gt)
throws UniformInterfaceException
```

Parameter

`gt`

This is the `GenericType` object that expresses the type of an HTTP response entity.

Return value

The `delete(GenericType<T> gt)` method returns the object of the type expressed by the specified `GenericType` object.

Exception

`UniformInterfaceException`

This exception is thrown if any of the following conditions is fulfilled. The error message KDJJ18888-E is output to the log.

- If the status code of the HTTP response is 204
- If the status code of the HTTP response is 300 or more and the `gt` parameter does not express the `ClientResponse` type

Notes

- For the Java types available for an HTTP response entity, see *25.15.2 Combination of Java types and MIME media types available for an HTTP response entity*.

- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown.

delete(GenericType<T> gt, Object requestEntity) method

Description

This method invokes the `HTTP DELETE` method. The `delete(GenericType<T> gt, Object requestEntity)` method includes an entity in the request and also receives a response entity.

Syntax

```
public <T> T delete(GenericType<T> gt,
Object requestEntity)
throws UniformInterfaceException
```

Parameter

`gt`

This is the `GenericType` object that expresses the type of an HTTP response entity.

`requestEntity`

This is the HTTP request entity.

Return value

The `delete(GenericType<T> gt, Object requestEntity)` method returns the object of the type expressed by the specified `GenericType` object.

Exception

`UniformInterfaceException`

This exception is thrown if any of the following conditions is fulfilled. The error message KDJJ18888-E is output to the log.

- If the status code of the HTTP response is 204
- If the status code of the HTTP response is 300 or more and the `gt` parameter does not express the `ClientResponse` type

Notes

- For the Java types available for an HTTP request entity, see *25.15.1 Combination of Java types and MIME media types available for an HTTP request entity*.
- For the Java types available for an HTTP response entity, see *25.15.2 Combination of Java types and MIME media types available for an HTTP response entity*.
- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown.

delete(Object requestEntity) method

Description

This method invokes the `HTTP DELETE` method. The `delete(Object requestEntity)` method includes an entity in the request, but does not receive a response entity.

An entity is ignored if the status code is below 300 and the entity is included in a response.

Syntax

```
public void delete(Object requestEntity)
throws UniformInterfaceException
```

Parameter

`requestEntity`
This is an HTTP request entity.

Return value

None.

Exception

`UniformInterfaceException`
This exception is thrown if the HTTP response status code is 300 or more. The error message KDJJ18888-E is output to the log.

Notes

- For the Java types available for an HTTP request entity, see *25.15.1 Combination of Java types and MIME media types available for an HTTP request entity*.
- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown.

entity(Object entity) method

Description

This method sets an HTTP request entity.

Syntax

```
public WebResource.Builder entity(Object entity)
```

Parameter

`entity`
This is an HTTP request entity.

Return value

The method returns a builder (`WebResource.Builder` object).

Notes

- For the Java types available for an HTTP request entity, see *25.15.1 Combination of Java types and MIME media types available for an HTTP request entity*.

entity(Object entity, MediaType type) method

Description

This method sets the MIME media type of an HTTP request entity.

For the Java types available for an HTTP request entity, see *25.15.1 Combination of Java types and MIME media types available for an HTTP request entity*. You can use the `GenericEntity` object to specify generics in the entity.

Syntax

```
public WebResource.Builder entity(Object entity,
MediaType type)
```

Parameters

`entity`
This is an HTTP request entity.

`type`
This is a MIME media type.

Return value

The method returns a builder (`WebResource.Builder` object).

Notes

- The non-null value specified in the `type` parameter is set in the `Content-Type HTTP` header. A null value is ignored and is not set in the `Content-Type HTTP` header.
- The value returned by the `toString()` method of the `MediaType` object is set in the `Content-Type HTTP` header. The JAX-RS engine does not validate the values returned by the `toString()` method. Specify the value according to the standard specifications.
- You can also use the following methods to change the value of the `Content-Type HTTP` header:
  - `ClientRequest.Builder` class
    `entity(Object entity, MediaType type)` method
    `entity(Object entity, String type)` method
    `header(String name, Object value)` method
    `type(MediaType type)` method
    `type(String type)` method
  - `WebResource` class
    `entity(Object entity, String type)` method
    `header(String name, Object value)` method
    `type(MediaType type)` method
    `type(String type)` method
  - `WebResource.Builder` class
    `entity(Object entity, MediaType type)` method
    `entity(Object entity, String type)` method
    `header(String name, Object value)` method
    `type(MediaType type)` method
    `type(String type)` method

The operation when the MIME media type is not set with these methods and the `entity(Object entity, MediaType type)` method is the same as the operation when the `Content-Type HTTP` header is not set in the `HttpURLConnection` object before performing HTTP communication.

entity(Object entity, String type) method

Description

This method sets the MIME media type of an HTTP request entity.

For the Java types available for an HTTP request entity, see *25.15.1 Combination of Java types and MIME media types available for an HTTP request entity*. You can use the `GenericEntity` object to specify generics in the entity.

Syntax

```
public WebResource.Builder entity(Object entity,
String type)
```

Parameters

`entity`
This is an HTTP request entity.

`type`
This is a MIME media type.

Return value

The method returns a builder (`WebResource.Builder` object).

Notes

- If `null` or an empty string is set in the `type` parameter, the `IllegalArgumentException` exception is thrown.
- The non-null value specified in the `type` parameter is set in the `Content-Type HTTP` header.
- The value returned by the `toString()` method of the `MediaType` object is set in the `Content-Type HTTP` header. The JAX-RS engine does not validate the values returned by the `toString()` method. Specify the value according to the standard specifications.
- You can also use the following methods to change the value of the `Content-Type HTTP` header:
  - `ClientRequest.Builder` class
    `entity(Object entity, MediaType type)` method
    `entity(Object entity, String type)` method
    `header(String name, Object value)` method
    `type(MediaType type)` method
    `type(String type)` method
  - `WebResource` class
    `entity(Object entity, MediaType type)` method
    `header(String name, Object value)` method
    `type(MediaType type)` method
    `type(String type)` method
  - `WebResource.Builder` class
    `entity(Object entity, MediaType type)` method

`entity(Object entity, String type)` method
`header(String name, Object value)` method
`type(MediaType type)` method
`type(String type)` method

The operation when the MIME media type is not set with these methods and the `entity(Object entity, String type)` method is the same as the operation when the `Content-Type HTTP` header is not set in the `HttpURLConnection` object before performing HTTP communication.

get(Class<T> c) method

Description

This method invokes the `HTTP GET` method.

Syntax

```
public <T> T get(Class<T> c)
throws UniformInterfaceException
```

Parameter

`c`

This is the type of an HTTP response entity.

Return value

The `get(Class<T> c)` method returns the object of the specified type.

Exception

`UniformInterfaceException`

This exception is thrown if any of the following conditions is fulfilled. The error message KDJJ18888-E is output to the log.

- If the status code of the HTTP response is 204
- If the status code of the HTTP response is 300 or more and the `c` parameter is not of the `ClientResponse` type

Notes

- For the Java types available for an HTTP response entity, see *25.15.2 Combination of Java types and MIME media types available for an HTTP response entity*.
- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown.

get(GenericType<T> gt) method

Description

This method invokes the `HTTP GET` method.

Syntax

```
public <T> T get(GenericType<T> gt)
throws UniformInterfaceException
```

Parameter

`gt`

This is the `GenericType` object that expresses the type of an HTTP response entity.

Return value

The `get(GenericType<T> gt)` method returns the object of the type expressed by the specified `GenericType` object.

Exception

`UniformInterfaceException`

This exception is thrown if any of the following conditions is fulfilled. The error message KDJJ18888-E is output to the log.

- If the status code of the HTTP response is 204
- If the status code of the HTTP response is 300 or more and the `gt` parameter does not express the `ClientResponse` type

Notes

- For the Java types available for an HTTP response entity, see *25.15.2 Combination of Java types and MIME media types available for an HTTP response entity*.
- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown.

getRequestBuilder() method

Description

This method acquires a builder (`WebResource.Builder` object).

Syntax

```
public WebResource.Builder getRequestBuilder()
```

Parameter

None.

Return value

The method returns the builder (`WebResource.Builder` object).

Notes

None.

getURI() method

Description

This method acquires a Web resource URI. A URI contains information that identifies the target resource of a request.

Syntax

```
public URI getURI()
```

Parameter

None.

Return value

This method returns a Web resource URI.

Notes

None.

getUriBuilder() method

Description

This method acquires a URI builder (`URIBuilder` object).

Syntax

```
public UriBuilder getUriBuilder()
```

Parameter

None.

Return value

This method returns the URI builder (`URIBuilder` object).

Notes

- The URI builder (`URIBuilder` object) created from the Web resource URI stored by this `WebResource` object is returned.

head() method

Description

This method invokes the `HTTP HEAD` method.

Syntax

```
public ClientResponse head()
```

Parameter

None.

Return value

The `head()` method returns an HTTP response.

Notes

- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown.

header(String name, Object value) method

Description

This method adds an HTTP header.

Syntax

```
public WebResource.Builder header(String name,
Object value)
```

Parameters

`name`
This is the name of an HTTP header.

`value`
This is the value of an HTTP header.

Return value

The method returns a builder (`WebResource.Builder` object).

Notes

- If the value of both `name` and `value` parameters is `null`, the method is not invoked.
- Even if the value of the `name` parameter is not null, but if the value of the `value` parameter is `null`, the method is not invoked.
- If the value of the `name` parameter is `null` and the value of the `value` parameter is not null, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps the `NullPointerException` exception is thrown.
- You cannot set the HTTP headers `Content-Length`, `Connection`, or `Host` with this method. If you specify these headers in the `name` parameter, the method is not invoked even if the value of the `value` parameter is not null. Note that `HttpURLConnection` sets each of the HTTP headers.
- The value returned by the `toString()` method of a non-null object specified with the `value` parameter is set as the value of the HTTP header specified with the `name` parameter. The JAX-RS engine does not validate the values returned by the `toString()` method. Specify the value in the `value` parameter according to the standard specifications.
- You can also use the following methods to add the HTTP headers `accept`, `acceptLanguage`, and `cookie`:
  - `ClientRequest.Builder` class
    `accept(MediaType... types)` method

`accept(String... types)` method
`acceptLanguage(Locale... locales)` method
`acceptLanguage(String... locales)` method
`cookie(Cookie cookie)` method

- `WebResource` class
  `accept(MediaType... types)` method
  `accept(String... types)` method
  `acceptLanguage(Locale... locales)` method
  `acceptLanguage(String... locales)` method
  `cookie(Cookie cookie)` method
- `WebResource.Builder` class
  `accept(MediaType... types)` method
  `accept(String... types)` method
  `acceptLanguage(Locale... locales)` method
  `acceptLanguage(String... locales)` method
  `cookie(Cookie cookie)` method

The operation when none of the HTTP headers is added with these methods and the `header(String name, Object value)` method is the same as the operation when none of the HTTP headers is added in the `HttpURLConnection` object.

- If the `Content-Type HTTP` header is already set by using the following methods, the value is overwritten by the value of the `value` parameter:
  - `ClientRequest.Builder` class
    `entity(Object entity, MediaType type)` method
    `entity(Object entity, String type)` method
    `header(String name, Object value)` method
    `type(MediaType type)` method
    `type(String type)` method
  - `WebResource` class
    `entity(Object entity, MediaType type)` method
    `entity(Object entity, String type)` method
    `type(MediaType type)` method
    `type(String type)` method
  - `WebResource.Builder` class
    `entity(Object entity, MediaType type)` method
    `entity(Object entity, String type)` method
    `header(String name, Object value)` method
    `type(MediaType type)` method
    `type(String type)` method

  The operation when the MIME media type is not set with these methods and the `header(String name, Object value)` method is the same as the operation when the `Content-Type HTTP` header is not set in the `HttpURLConnection` object before performing HTTP communication.

method(String method) method

Description

This method invokes an HTTP method. The `method(String method)` method does not include any entity in the request and does not receive any response entity.

An entity is ignored if the status code is below 300 and the entity is included in a response.

Syntax

```
public void method(String method)
throws UniformInterfaceException
```

Parameter

`method`
This is the name of an HTTP method.

Return value

None.

Exception

`UniformInterfaceException`
This exception is thrown if the HTTP response status code is 300 or more. The error message KDJJ18888-E is output to the log.

Notes

- Specify uppercase `GET`, `HEAD`, `POST`, `OPTIONS`, `PUT`, or `DELETE` in the `method` parameter.
- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown.

method(String method, Class<T> c) method

Description

This method invokes an HTTP method. The `method(String method, Class<T> c)` method does not include any entity in the request, but receives a response entity.

Syntax

```
public <T> T method(String method,
Class<T> c)
throws UniformInterfaceException
```

Parameters

`method`
This is an HTTP method.

`c`
This is the type of an HTTP response entity.

Return value

The method returns the object of the specified type.

Exception

`UniformInterfaceException`

This exception is thrown if any of the following conditions is fulfilled. The error message KDJJ18888-E is output to the log.

- If the status code of the HTTP response is 204
- If the status code of the HTTP response is 300 or more and the `c` parameter is not of the `ClientResponse` type

Notes

- Specify uppercase `GET`, `HEAD`, `POST`, `OPTIONS`, `PUT`, or `DELETE` in the `method` parameter.
- For the Java types available for an HTTP response entity, see *25.15.2 Combination of Java types and MIME media types available for an HTTP response entity*.
- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown.

method(String method, Class<T> c, Object requestEntity) method

Description

This method invokes an HTTP method. The `method(String method, Class<T> c, Object requestEntity)` method does not include any entity in the request, but receives a response entity.

Syntax

```
public <T> T method(String method,
Class<T> c,
Object requestEntity)
throws UniformInterfaceException
```

Parameters

`method`
This is an HTTP method.

`c`
This is the type of an HTTP response entity.

`requestEntity`
This is an HTTP request entity.

Return value

The method returns the object of the specified type.

Exception

`UniformInterfaceException`
This exception is thrown if any of the following conditions is fulfilled. The error message KDJJ18888-E is output to the log.

- If the status code of the HTTP response is 204
- If the status code of the HTTP response is 300 or more and the `c` parameter is not of the `ClientResponse` type

Notes

- Specify uppercase `GET`, `HEAD`, `POST`, `OPTIONS`, `PUT`, or `DELETE` in the `method` parameter.
- For the Java types available for an HTTP request entity, see *25.15.1 Combination of Java types and MIME media types available for an HTTP request entity*.
- For the Java types available for an HTTP response entity, see *25.15.2 Combination of Java types and MIME media types available for an HTTP response entity*.
- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown.

method(String method, GenericType<T> gt) method

Description

This method invokes an HTTP method. The `method (String method, GenericType<T> gt)` method does not include any entity in the request, but receives a response entity.

Syntax

```
public <T> T method(String method,
GenericType<T> gt)
throws UniformInterfaceException
```

Parameters

`method`

This is an HTTP method.

`gt`

This is the `GenericType` object that expresses the type of an HTTP response entity.

Return value

The method returns the object of the type expressed by the specified `GenericType` object.

Exception

`UniformInterfaceException`

This exception is thrown if any of the following conditions is fulfilled. The error message KDJJ18888-E is output to the log.

- If the status code of the HTTP response is 204
- If the status code of the HTTP response is 300 or more and the `gt` parameter does not express the `ClientResponse` type

Notes

- Specify uppercase `GET`, `HEAD`, `POST`, `OPTIONS`, `PUT`, or `DELETE` in the `method` parameter.
- For the Java types available for an HTTP response entity, see *25.15.2 Combination of Java types and MIME media types available for an HTTP response entity*.
- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown.

method(String method, GenericType<T> gt, Object requestEntity) method

Description

This method invokes an HTTP method. The `method(String method, GenericType<T>gt, Object requestEntity)` method does not include any entity in the request, but receives a response entity.

Syntax

```
public <T> T method(String method,
GenericType<T> gt,
Object requestEntity)
throws UniformInterfaceException
```

Parameters

`method`
This is an HTTP method.

`gt`
This is the `GenericType` object that expresses the type of an HTTP response entity.

`requestEntity`
This is an HTTP request entity.

Return value

The method returns the object of the type expressed by the specified `GenericType` object.

Exception

`UniformInterfaceException`
This exception is thrown if any of the following conditions is fulfilled. The error message KDJJ18888-E is output to the log.

- If the status code of the HTTP response is 204
- If the status code of the HTTP response is 300 or more and the `gt` parameter does not express the `ClientResponse` type

Notes

- Specify uppercase `GET`, `HEAD`, `POST`, `OPTIONS`, `PUT`, or `DELETE` in the `method` parameter.
- For the Java types available for an HTTP request entity, see *25.15.1 Combination of Java types and MIME media types available for an HTTP request entity*.
- For the Java types available for an HTTP response entity, see *25.15.2 Combination of Java types and MIME media types available for an HTTP response entity*.
- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown.

method(String method, Object requestEntity) method

Description

This method invokes an HTTP method. The `method(String method, Object request Entity)` method does not include any entity in the request, but receives a response entity.

An entity is ignored if the status code is below 300 and the entity is included in a response.

Syntax

```
public void method(String method,
Object requestEntity)
throws UniformInterfaceException
```

Parameters

`method`

This is an HTTP method.

`requestEntity`

This is an HTTP request entity.

Return value

None.

Exception

`UniformInterfaceException`

This exception is thrown if the HTTP response status code is 300 or more. The error message KDJJ18888-E is output to the log.

Notes

- Specify uppercase `GET`, `HEAD`, `POST`, `OPTIONS`, `PUT`, or `DELETE` in the `method` parameter.
- For the Java types available for an HTTP request entity, see *25.15.1 Combination of Java types and MIME media types available for an HTTP request entity*.
- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown.

options(Class<T> c) method

Description

This method invokes the `HTTP OPTIONS` method.

Syntax

```
public <T> T options(Class<T> c)
throws UniformInterfaceException
```

Parameter

`c`

This is the type of an HTTP response entity.

Return value

The `options(Class<T> c)` method returns the object of the specified type.

Exception

`UniformInterfaceException`

This exception is thrown if any of the following conditions is fulfilled. The error message KDJJ18888-E is output to the log.

- If the status code of the HTTP response is 204
- If the status code of the HTTP response is 300 or more and the `c` parameter is not of the `ClientResponse` type

Notes

- For the Java types available for an HTTP response entity, see *25.15.2 Combination of Java types and MIME media types available for an HTTP response entity*.
- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown.

options(GenericType<T> gt) method

Description

This method invokes the `HTTP OPTIONS` method.

Syntax

```
public <T> T options(GenericType<T> gt)
throws UniformInterfaceException
```

Parameter

`gt`

This is the `GenericType` object that expresses the type of an HTTP response entity.

Return value

The `options(GenericType<T> gt)` method returns the object of the type expressed by the specified `GenericType` object.

Exception

`UniformInterfaceException`

This exception is thrown if any of the following conditions is fulfilled. The error message KDJJ18888-E is output to the log.

- If the status code of the HTTP response is 204

- If the status code of the HTTP response is 300 or more and the `gt` parameter does not express the `ClientResponse` type

Notes

- For the Java types available for an HTTP response entity, see *25.15.2 Combination of Java types and MIME media types available for an HTTP response entity*.
- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown.

path(String path) method

Description

This method generates a new `WebResource` object from this object (`WebResource` object). The Web resource URI that is stored in the generated object is the URI for which the path specified by the parameter is added to the Web resource URI stored in this object.

Syntax

```
public WebResource path(String path)
```

Parameter

`path`

This is the path to be added.

Return value

The method returns the generated `WebResource` object.

Notes

- If `null` is set in the `path` parameter, the `IllegalArgumentException` exception is thrown.
- Specify a non-null and correct path in the `path` parameter. The invalid characters are automatically URL encoded according to the standard specifications.
- Note that a query string is not a part of the path; therefore, considering that a question mark is an invalid character, the question mark (?) is automatically URL encoded. The following characters are not encoded:

  `! $ & ' ( ) * + - / ; = @ _ ~ . ,`

  Single-byte alphanumeric characters (0 to 9, A to Z, and a to z)

  The already URL-encoded tokens are correctly recognized and hence are not URL-encoded again.
- As and when required, a forward slash (/) is automatically inserted between the Web resource URI that is stored in this object (`WebResource` object) and the path specified with the `path` parameter. Also, if more than one forward slash (/) is inserted, the extra ones are automatically corrected.
- The entity of this object (`WebResource` object) is not copied to the `WebResource` object that is generated with this method.

post() method

Description

This method invokes the `HTTP POST` method. The `post()` method does not include any entity in the request and does not receive any response entity.

An entity is ignored if the status code is below 300 and the entity is included in a response.

Syntax

```
public void post()
throws UniformInterfaceException
```

Parameter

None.

Return value

None.

Exception

`UniformInterfaceException`

This exception is thrown if the HTTP response status code is 300 or more. The error message KDJJ18888-E is output to the log.

Notes

- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown.

post(Class<T> c) method

Description

This method invokes the `HTTP POST` method. The `post(Class<T> c)` method does not include any entity in the request, but receives a response entity.

Syntax

```
public <T> T post(Class<T> c)
throws UniformInterfaceException
```

Parameter

`c`

This is the type of an HTTP response entity.

Return value

The `post(Class<T> c)` method returns the object of the specified type.

Exception

`UniformInterfaceException`

This exception is thrown if any of the following conditions is fulfilled. The error message KDJJ18888-E is output to the log.

- If the status code of the HTTP response is 204
- If the status code of the HTTP response is 300 or more and the `c` parameter is not of the `ClientResponse` type

Notes

- For the Java types available for an HTTP response entity, see *25.15.2 Combination of Java types and MIME media types available for an HTTP response entity*.
- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown.

post(Class<T> c, Object requestEntity) method

Description

This method invokes the `HTTP POST` method. The `post(Class<T> c, Object requestEntity)` method includes an entity in the request and also receives a response entity.

Syntax

```
public <T> T post(Class<T> c,
Object requestEntity)
throws UniformInterfaceException
```

Parameters

`c`

This is the type of an HTTP response entity.

`requestEntity`

This is an HTTP request entity.

Return value

The `post(Class<T> c, Object requestEntity)` method returns the object of the specified type.

Exception

`UniformInterfaceException`

This exception is thrown if any of the following conditions is fulfilled. The error message KDJJ18888-E is output to the log.

- If the status code of the HTTP response is 204
- If the status code of the HTTP response is 300 or more and the `c` parameter is not of the `ClientResponse` type

Notes

- For the Java types available for an HTTP request entity, see *25.15.1 Combination of Java types and MIME media types available for an HTTP request entity*.

- For the Java types available for an HTTP response entity, see *25.15.2 Combination of Java types and MIME media types available for an HTTP response entity*.
- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown.

post(GenericType<T> gt) method

Description

This method invokes the `HTTP POST` method. The `post(GenericType<T> gt)` method includes an entity in the request and also receives a response entity.

Syntax

```
public <T> T post(GenericType<T> gt)
throws UniformInterfaceException
```

Parameter

`gt`

This is the `GenericType` object that expresses the type of an HTTP response entity.

Return value

The `post(GenericType<T> gt)` method returns the object of the type expressed by the specified `GenericType` object.

Exception

`UniformInterfaceException`

This exception is thrown if any of the following conditions is fulfilled. The error message KDJJ18888-E is output to the log.

- If the status code of the HTTP response is 204
- If the status code of the HTTP response is 300 or more and the `gt` parameter does not express the `ClientResponse` type

Notes

- For the Java types available for an HTTP response entity, see *25.15.2 Combination of Java types and MIME media types available for an HTTP response entity*.
- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown.

post(GenericType<T> gt, Object requestEntity) method

Description

This method invokes the `HTTP POST` method. The `post(GenericType<T> gt, Object requestEntity)` method includes an entity in the request and also receives a response entity.

Syntax

```
public <T> T post(GenericType<T> gt,
```

```
Object requestEntity)
throws UniformInterfaceException
```

Parameters

`gt`
This is the `GenericType` object that expresses the type of an HTTP response entity.

`requestEntity`
This is an HTTP request entity.

Return value

The `post(GenericType<T> gt, Object requestEntity)` method returns the object of the type expressed by the specified `GenericType` object.

Exception

`UniformInterfaceException`
This exception is thrown if any of the following conditions is fulfilled. The error message KDJJ18888-E is output to the log.

- If the status code of the HTTP response is 204
- If the status code of the HTTP response is 300 or more and the `gt` parameter does not express the `ClientResponse` type

Notes

- For the Java types available for an HTTP request entity, see *25.15.1 Combination of Java types and MIME media types available for an HTTP request entity*.
- For the Java types available for an HTTP response entity, see *25.15.2 Combination of Java types and MIME media types available for an HTTP response entity*.
- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown.

post(Object requestEntity) method

Description

This method invokes the `HTTP POST` method. The `post(Object requestEntity)` method includes an entity in the request, but does not receive a response entity.

An entity is ignored if the status code is below 300 and the entity is included in a response.

Syntax

```
public void post(Object requestEntity)
throws UniformInterfaceException
```

Parameter

`requestEntity`
This is an HTTP request entity.

Return value

None.

Exception

`UniformInterfaceException`

This exception is thrown if the HTTP response status code is 300 or more. The error message KDJJ18888-E is output to the log.

Notes

- For the Java types available for an HTTP request entity, see *25.15.1 Combination of Java types and MIME media types available for an HTTP request entity*.
- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown.

put() method

Description

This method invokes the `HTTP PUT` method. The `put()` method does not include any entity in the request and does not receive any response entity.

An entity is ignored if the status code is below 300 and the entity is included in a response.

Syntax

```
public void put()
throws UniformInterfaceException
```

Parameter

None.

Return value

None.

Exception

`UniformInterfaceException`

This exception is thrown if the HTTP response status code is 300 or more. The error message KDJJ18888-E is output to the log.

Notes

- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown.

put(Class<T> c) method

Description

This method invokes the `HTTP PUT` method. The `put(Class<T> c)` method includes an entity in the request and also receives a response entity.

Syntax

```
public <T> T put(Class<T> c)
throws UniformInterfaceException
```

Parameter

`c`

This is the type of an HTTP response entity.

Return value

The `put(Class<T> c)` method returns the object of the specified type.

Exception

`UniformInterfaceException`

This exception is thrown if any of the following conditions is fulfilled. The error message KDJJ18888-E is output to the log.

- If the status code of the HTTP response is 204
- If the status code of the HTTP response is 300 or more and the `c` parameter is not of the `ClientResponse` type

Notes

- For the Java types available for an HTTP response entity, see *25.15.2 Combination of Java types and MIME media types available for an HTTP response entity*.
- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown.

put(Class<T> c, Object requestEntity) method

Description

This method invokes the `HTTP PUT` method. The `put(Class<T> c, Object requestEntity)` method includes an entity in the request and also receives a response entity.

Syntax

```
public <T> T put(Class<T> c,
Object requestEntity)
throws UniformInterfaceException
```

Parameters

`c`

This is the type of an HTTP response entity.

`requestEntity`

This is an HTTP request entity.

Return value

The `put(Class<T> c, Object requestEntity)` method returns the object of the specified type.

Exception

This exception is thrown if any of the following conditions is fulfilled. The error message KDJJ18888-E is output to the log.

- If the status code of the HTTP response is 204
- If the status code of the HTTP response is 300 or more and the `c` parameter is not of the `ClientResponse` type

Notes

- For the Java types available for an HTTP request entity, see *25.15.1 Combination of Java types and MIME media types available for an HTTP request entity*.
- For the Java types available for an HTTP response entity, see *25.15.2 Combination of Java types and MIME media types available for an HTTP response entity*.
- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown.

put(GenericType<T> gt) method

Description

This method invokes the `HTTP PUT` method. The `put(GenericType<T> gt)` method includes an entity in the request and also receives a response entity.

Syntax

```
public <T> T put(GenericType<T> gt)
throws UniformInterfaceException
```

Parameter

`gt`

This is the `GenericType` object that expresses the type of an HTTP response entity.

Return value

The `put(GenericType<T> gt)` method returns the object of the type expressed by the specified `GenericType` object.

Exception

`UniformInterfaceException`

This exception is thrown if any of the following conditions is fulfilled. The error message KDJJ18888-E is output to the log.

- If the status code of the HTTP response is 204
- If the status code of the HTTP response is 300 or more and the `gt` parameter does not express the `ClientResponse` type

Notes

- For the Java types available for an HTTP response entity, see *25.15.2 Combination of Java types and MIME media types available for an HTTP response entity*.
- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown.

put(GenericType<T> gt, Object requestEntity) method

Description

This method invokes the `HTTP PUT` method. The `put(GenericType<T> gt, Object requestEntity)` method includes an entity in the request and also receives a response entity.

Syntax

```
public <T> T put(GenericType<T> gt,
Object requestEntity)
throws UniformInterfaceException
```

Parameters

`gt`

This is the `GenericType` object that expresses the type of an HTTP response entity.

`requestEntity`

This is an HTTP request entity.

Return value

The `put(GenericType<T> gt, Object requestEntity)` method returns the object of the type expressed by the specified `GenericType` object.

Exception

`UniformInterfaceException`

This exception is thrown if any of the following conditions is fulfilled. The error message KDJJ18888-E is output to the log.

- If the status code of the HTTP response is 204
- If the status code of the HTTP response is 300 or more and the `gt` parameter does not express the `ClientResponse` type

Notes

- For the Java types available for an HTTP request entity, see *25.15.1 Combination of Java types and MIME media types available for an HTTP request entity*.
- For the Java types available for an HTTP response entity, see *25.15.2 Combination of Java types and MIME media types available for an HTTP response entity*.
- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown.

put(Object requestEntity) method

Description

This method invokes the `HTTP PUT` method. The `put(Object requestEntity)` method includes an entity in the request and also receives a response entity.

An entity is ignored if the status code is below 300 and the entity is included in a response.

Syntax

```
public void put(Object requestEntity)
throws UniformInterfaceException
```

Parameter

`requestEntity`
This is an HTTP request entity.

Return value

The `put(Object requestEntity)` method returns the object of the type expressed by the specified `GenericType` object.

Exception

`UniformInterfaceException`
This exception is thrown if the HTTP response status code is 300 or more. The error message KDJJ18888-E is output to the log.

Notes

- For the Java types available for an HTTP request entity, see *25.15.1 Combination of Java types and MIME media types available for an HTTP request entity*.
- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown.

queryParam(String key, String value) method

Description

This method generates a new `WebResource` object from this object (`WebResource` object). The Web resource URI that is stored in the generated object is the URI for which the query parameter specified in the parameter is added to the Web resource URI stored in this object.

Syntax

```
public WebResource queryParam(String key,
String value)
```

Parameters

`key`
This is the name of a query parameter.

`value`
This is the value of a query parameter.

Return value

The method returns the generated `WebResource` object.

Notes

- If `null` is specified in the `key` parameter or the `value` parameter, the `IllegalArgumentException` exception is thrown.
- Specify the correct query parameter name and value in the `key` and `value` parameter. The invalid characters are automatically URL encoded according to the standard specifications.
  - The following characters are not encoded:
    `! $ ' ( ) * - / ; ? @ _ ~ . ,`
    Single-byte alphanumeric characters (0 to 9, A to Z, and a to z)
  - The already URL-encoded tokens are correctly recognized and hence are not URL-encoded again.
  - Spaces are encoded as `+`.
  - The query parameter of the generated `WebResource` object is the one for which the query parameter specified in the parameter is added in the query parameter of this object (`WebResource` object).
- The entity of this object (`WebResource` object) is not copied to the `WebResource` object generated with this method.

queryParams(MultivaluedMap<String, String> params) method

Description

This method generates a new `WebResource` object from this object (`WebResource` object). The Web resource URI that is stored in the generated object is the URI for which the query parameter specified in the parameter is added to the Web resource URI stored in this object.

Syntax

```
public WebResource queryParams(MultivaluedMap<String,String> params)
```

Parameter

`params`
This is the map of query parameters.

Return value

The method returns the generated `WebResource` object.

Notes

- The `NullPointerException` exception is thrown if `null` is specified in the `params` parameter.
- If a `null` key is included in the query parameter map specified in the `params` parameter, the `IllegalArgumentException` exception is thrown.
- From the combination of keys and values in the query parameter map specified in the `params` parameter, if the key is non-null and the value is `null`, only the key is added as a query parameter of the generated `WebResource` object.
- Specify the correct query parameter in the `params` parameter. The invalid characters are automatically URL encoded according to the standard specifications.
  - The following characters are not encoded:
    `! $ ' ( ) * - / ; ? @ _ ~ . ,`
    Single-byte alphanumeric characters (0 to 9, A to Z, and a to z)
  - The already URL-encoded tokens are correctly recognized and hence are not URL-encoded again.
  - Spaces are encoded as +.
  - The query parameter of the generated `WebResource` object is the one for which the query parameter specified in the parameter is added in the query parameter of this object (`WebResource` object). Note that this is applicable even when a query parameter of the same name exists in this object.
- The entity of this object (`WebResource` object) is not copied to the `WebResource` object generated with this method.

type(MediaType type) method

Description

This method sets a MIME media type.

Syntax

```
public WebResource.Builder type(MediaType type)
```

Parameter

`type`
This is a MIME media type.

Return value

The method returns a builder (`WebResource.Builder` object).

Notes

- The non-null value specified in the `type` parameter is set in the `Content-Type HTTP header`. A null value is ignored and is not set to the `Content-Type HTTP` header.
- The value returned by the `toString()` method of the `MediaType` object is set in the `Content-Type HTTP` header. The JAX-RS engine does not validate the values returned by the `toString()` method. Specify the value according to the standard specifications.
- If the `Content-Type HTTP` header is already set by using the following methods, the value is overwritten:
  - `ClientRequest.Builder` class
    `entity(Object entity, MediaType type)` method
    `entity(Object entity, String type)` method

`header(String name, Object value)` method
`type(MediaType type)` method
`type(String type)` method

- `WebResource` class
  `entity(Object entity, MediaType type)` method
  `entity(Object entity, String type)` method
  `header(String name, Object value)` method
  `type(String type)` method
- `WebResource.Builder` class
  `entity(Object entity, MediaType type)` method
  `entity(Object entity, String type)` method
  `header(String name, Object value)` method
  `type(MediaType type)` method
  `type(String type)` method

The operation when the MIME media type is not set with these methods and the `type(MediaType type)` method is the same as the operation when the `Content-Type HTTP` header is not set in the `HttpURLConnection` object before performing HTTP communication.

type(String type) method

Description

This method sets a MIME media type.

Syntax

```
public WebResource.Builder type(String type)
```

Parameter

`type`
This is a MIME media type.

Return value

The method returns a builder (`WebResource.Builder` object).

Notes

- If `null` or an empty string is set in the `type` parameter, the `IllegalArgumentException` exception is thrown.
- The value returned by the `toString()` method of the `MediaType` object set up by specifying the `type` parameter in the parameter of the `valueOf(String)` static method of the `MediaType` class is set in the `Content-Type HTTP` header. The JAX-RS engine does not validate the values specified in the `type` parameter. Specify the value according to the standard specifications.
- If the `Content-Type HTTP` header is already set by using the following methods, the value is overwritten by the value of the `type` parameter:
  - `ClientRequest.Builder` class
    `entity(Object entity, MediaType type)` method
    `entity(Object entity, String type)` method
    `header(String name, Object value)` method

`type(MediaType type)` method
`type(String type)` method

- `WebResource` class
  `entity(Object entity, MediaType type)` method
  `entity(Object entity, String type)` method
  `header(String name, Object value)` method
  `type(MediaType type)` method
- `WebResource.Builder` class
  `entity(Object entity, MediaType type)` method
  `entity(Object entity, String type)` method
  `header(String name, Object value)` method
  `type(MediaType type)` method
  `type(String type)` method

The operation when the MIME media type is not set with these methods and the `type(String type)` method is the same as the operation when the `Content-Type HTTP` header is not set in the `HttpURLConnection` object before performing HTTP communication.

uri(java.net.URI uri) method

Description

This method generates a new `WebResource` object from this object (`WebResource` object).

If the URI specified in the parameter contains a path component beginning with a forward slash (/), the path of the Web resource URI stored in the generated `WebResource` object is replaced. If a path component beginning with a forward slash (/) does not exist, the path of the Web resource URI stored in the generated `WebResource` object is the path where that path component is added in the path of this object (`WebResource` object).

If the URI specified in the parameter contains a query parameter, this query parameter becomes the query parameter for the Web resource URI stored in the generated `WebResource` object. If the Web resource URI stored in this object contains a query parameter, the query parameter is replaced.

Syntax

```
public WebResource uri(URI uri)
```

Parameter

`uri`
This is the Web resource URI.

Return value

The method returns the generated `WebResource` object.

Notes

- Specify the correct URI in the `uri` parameter. The `uri` parameter is not automatically URL encoded. Specify the URL-encoded value in the `uri` parameter as and when required.
- The entity of this object (`WebResource` object) is not copied to the `WebResource` object generated with this method.
- The `NullPointerException` exception is thrown if `null` is specified in the `uri` parameter.

25.11 Method specifications and notes for the WebResource.Builder class

This section describes the specifications for the methods of the `WebResource.Builder` class and the notes on using the methods.

accept(MediaType... types) method

Description

This method adds a receivable MIME media type.

Syntax

```
public T accept(MediaType... types)
```

Parameter

`types`

This is an array of the receivable MIME media types.

Return value

The method returns a builder (`WebResource.Builder` object) to set up the `WebResource` object.

Notes

- The non-null value included in the `types` parameter is added in the `Accept HTTP` header. A null value is ignored and is not added in the `Accept HTTP` header.
- The value returned by the `toString()` method of the `MediaType` object is set in the `Accept HTTP` header. The JAX-RS engine does not validate the values returned by the `toString()` method. Specify the value according to the standard specifications.
- You can also use the following methods to add the value of the `Accept HTTP` header:
  - `ClientRequest.Builder` class
    `accept(MediaType... types)` method
    `accept(String... types)` method
    `header(String name, Object value)` method
  - `WebResource` class
    `accept(MediaType... types)` method
    `accept(String... types)` method
    `header(String name, Object value)` method
  - `WebResource.Builder` class
    `accept(String... types)` method
    `header(String name, Object value)` method

  The operation when no receivable MIME media type is added with these methods and the `accept(MediaType... types)` method is the same as the operation when the `Accept HTTP` header is not added in the `HttpURLConnection` object before performing HTTP communication.

accept(String... types) method

Description

This method adds a receivable MIME media type.

Syntax

```
public T accept(String... types)
```

Parameter

`types`

This is an array of the receivable MIME media types.

Return value

The method returns a builder (`WebResource.Builder` object) to set up the `WebResource` object.

Notes

- The non-null value included in the `types` parameter is added in the `Accept HTTP` header. A null value is ignored and is not added in the `Accept HTTP` header.
- The specified value is set as is in the `Accept HTTP` header. The JAX-RS engine does not validate the specified value. Specify the value according to the standard specifications.
- You can also use the following methods to add a value of the `Accept HTTP` header:
  - `ClientRequest.Builder` class
    `accept(MediaType... types)` method
    `accept(String... types)` method
    `header(String name, Object value)` method
  - `WebResource` class
    `accept(MediaType... types)` method
    `accept(String... types)` method
    `header(String name, Object value)` method
  - `WebResource.Builder` class
    `accept(MediaType... types)` method
    `header(String name, Object value)` method

  The operation when no receivable MIME media type is added with these methods and the `accept(String... types)` method is the same as the operation when the `Accept HTTP` header is not added in the `HttpURLConnection` object before performing HTTP communication.

acceptLanguage(Locale... locales) method

Description

This method adds a receivable language.

Syntax

```
public T acceptLanguage(Locale... locales)
```

Parameter

`locales`
This is an array of the receivable languages.

Return value

The method returns a builder (`WebResource.Builder` object) to set up the `WebResource` object.

Notes

- The non-null value in the `locales` parameter is added in the `Accept-Language HTTP` header. A null value is ignored and is not added in the `Accept-Language HTTP` header.
- The value returned by the `toString()` method of the `Locale` object is set in the `Accept-Language HTTP` header. The JAX-RS engine does not validate the values returned by the `toString()` method. Specify the value according to the standard specifications.
- You can also use the following methods to change the value of the `Accept-Language HTTP` header:
  - `ClientRequest.Builder` class
    `acceptLanguage(Locale... locales)` method
    `acceptLanguage(String... locales)` method
    `header(String name, Object value)` method
  - `WebResource` class
    `acceptLanguage(Locale... locales)` method
    `acceptLanguage(String... locales)` method
    `header(String name, Object value)` method
  - `WebResource.Builder` class
    `acceptLanguage(String... locales)` method
    `header(String name, Object value)` method

  The operation when no receivable language is added with these methods and the `acceptLanguage(Locale... locales)` method is the same as the operation when the `Accept-Language HTTP` header is not set in the `HttpURLConnection` object before performing HTTP communication.

acceptLanguage(String... locales) method

Description

This method adds a receivable language.

Syntax

```
public T acceptLanguage(String... locales)
```

Parameter

`locales`
This is an array of the receivable languages.

Return value

The method returns a builder (`WebResource.Builder` object) to set up the `WebResource` object.

Notes

- The non-null value in the `locales` parameter is added in the `Accept-Language HTTP` header. A `null` value is ignored and is not added in the `Accept-Language HTTP` header.
- The specified value is set as is in the `Accept-Language HTTP` header. The JAX-RS engine does not validate the specified value. Specify the value according to the standard specifications.
- You can also use the following methods to add the value of the `Accept-Language HTTP` header:
  - `ClientRequest.Builder` class
    `acceptLanguage(Locale... locales)` method
    `acceptLanguage(String... locales)` method
    `header(String name, Object value)` method
  - `WebResource` class
    `acceptLanguage(Locale... locales)` method
    `acceptLanguage(String... locales)` method
    `header(String name, Object value)` method
  - `WebResource.Builder` class
    `acceptLanguage(Locale... locales)` method
    `header(String name, Object value)` method

  The operation when no receivable language is added with these methods and the `acceptLanguage(String... locales)` method is the same as the operation when the `Accept-Language HTTP` header is not added in the `HttpURLConnection` object before performing HTTP communication.

cookie(Cookie cookie) method

Description

This method sets a Cookie.

Syntax

```
public T cookie(Cookie cookie)
```

Parameter

`cookie`
This is the Cookie to be set up.

Return value

The method returns a builder (`WebResource.Builder` object) to set up the `WebResource` object.

Notes

- A non-null value is added to `Cookie HTTP` header. A `null` value is ignored and is not added in the `Cookie HTTP` header.
- The value returned by the `toString()` method of the `Cookie` object is set in the `Cookie HTTP` header. The JAX-RS engine does not validate the values returned by the `toString()` method. Specify the value according to the standard specifications.
- You can also use the following methods to add the value of the `Cookie HTTP` header:
  - `ClientRequest.Builder` class

`cookie(Cookie cookie)` method
`header(String name, Object value)` method

- `WebResource` class
  `cookie(Cookie cookie)` method
  `header(String name, Object value)` method
- `WebResource.Builder` class
  `header(String name, Object value)` method

The operation when no Cookie is added with these methods and the `cookie(Cookie cookie)` method is the same as the operation when the `Cookie HTTP` header is not added in the `HttpURLConnection` object before performing HTTP communication.

delete() method

Description

This method invokes the `HTTP DELETE` method. The `delete()` method does not include any entity in the request and does not receive any response entity.

An entity is ignored if the status code is below 300 and the entity is included in a response.

Syntax

```
public void delete()
throws UniformInterfaceException
```

Parameter

None.

Return value

None.

Exception

`UniformInterfaceException`
This exception is thrown if the HTTP response status code is 300 or more. The error message KDJJ18888-E is output to the log.

Notes

- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown.

delete(Class<T> c) method

Description

This method invokes the `HTTP DELETE` method. The `delete(Class<T> c)` method does not include any entity in the request, but receives a response entity.

Syntax

```
public <T> T delete(Class<T> c)
throws UniformInterfaceException
```

Parameter

`c`

This is the type of an HTTP response entity.

Return value

The `delete(Class<T> c)` method returns the object of the specified type.

Exception

`UniformInterfaceException`

This exception is thrown if any of the following conditions is fulfilled. The error message KDJJ18888-E is output to the log.

- If the status code of the HTTP response is 204
- If the status code of the HTTP response is 300 or more and the `c` parameter is not of the `ClientResponse` type

Notes

- For the Java types available for an HTTP response entity, see *25.15.2 Combination of Java types and MIME media types available for an HTTP response entity*.
- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown.

delete(Class<T> c, Object requestEntity) method

Description

This method invokes the `HTTP DELETE` method. The `delete(Class<T>c, Object requestEntity)` method includes an entity in the request and also receives a response entity.

Syntax

```
public <T> T delete(Class<T> c,
Object requestEntity)
throws UniformInterfaceException
```

Parameter

`c`

This is the type of an HTTP response entity.

`requestEntity`

This is an HTTP request entity.

Return value

The `delete(Class<T>c, Object requestEntity)` method returns the object of the specified type.

Exception

`UniformInterfaceException`

This exception is thrown if any of the following conditions is fulfilled. The error message KDJJ18888-E is output to the log.

- If the status code of the HTTP response is 204
- If the status code of the HTTP response is 300 or more and the `c` parameter is not of the `ClientResponse` type

Notes

- For the Java types available for an HTTP request entity, see *25.15.1 Combination of Java types and MIME media types available for an HTTP request entity*.
- For the Java types available for an HTTP response entity, see *25.15.2 Combination of Java types and MIME media types available for an HTTP response entity*.
- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown.

delete(GenericType<T> gt) method

Description

This method invokes the `HTTP DELETE` method. The `delete(GenericType<T> gt)` method does not include any entity in the request, but receives a response entity.

Syntax

```
public <T> T delete(GenericType<T> gt)
throws UniformInterfaceException
```

Parameter

`gt`

This is the `GenericType` object that expresses the type of an HTTP response entity.

Return value

The `delete(GenericType<T> gt)` method returns the object of the type expressed by the specified `GenericType` object.

Exception

UniformInterfaceException

This exception is thrown if any of the following conditions is fulfilled. The error message KDJJ18888-E is output to the log.

- If the status code of the HTTP response is 204
- If the status code of the HTTP response is 300 or more and the `gt` parameter does not express the `ClientResponse` type

Notes

- For the Java types available for an HTTP response entity, see *25.15.2 Combination of Java types and MIME media types available for an HTTP response entity*.

- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown.

delete(GenericType<T> gt, Object requestEntity) method

Description

This method invokes the `HTTP DELETE` method. The `delete(GenericType<T> gt, Object requestEntity)` method includes an entity in the request and also receives a response entity.

Syntax

```
public <T> T delete(GenericType<T> gt,
Object requestEntity)
throws UniformInterfaceException
```

Parameters

`gt`

This is the `GenericType` object that expresses the type of an HTTP response entity.

`requestEntity`

This is an HTTP request entity.

Return value

The `delete(GenericType<T> gt, Object requestEntity)` method returns the object of the type expressed by the specified `GenericType` object.

Exception

`UniformInterfaceException`

This exception is thrown if any of the following conditions is fulfilled. The error message KDJJ18888-E is output to the log.

- If the status code of the HTTP response is 204
- If the status code of the HTTP response is 300 or more and the `gt` parameter does not express the `ClientResponse` type

Notes

- For the Java types available for an HTTP request entity, see *25.15.1 Combination of Java types and MIME media types available for an HTTP request entity*.
- For the Java types available for an HTTP response entity, see *25.15.2 Combination of Java types and MIME media types available for an HTTP response entity*.
- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown.

delete(Object requestEntity) method

Description

This method invokes the `HTTP DELETE` method. The `delete(Object requestEntity)` method includes an entity in the request, but does not receive a response entity.

An entity is ignored if the status code is below 300 and the entity is included in a response.

Syntax

```
public void delete(Object requestEntity)
throws UniformInterfaceException
```

Parameter

`requestEntity`
This is an HTTP request entity.

Return value

None.

Exception

`UniformInterfaceException`
This exception is thrown if the HTTP response status code is 300 or more. The error message KDJJ18888-E is output to the log.

Notes

- For the Java types available for an HTTP request entity, see *25.15.1 Combination of Java types and MIME media types available for an HTTP request entity*.
- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown.

entity(Object entity) method

Description

This method sets an HTTP request entity.

Syntax

```
public T entity(Object entity)
```

Parameter

`entity`
This is an HTTP request entity.

Return value

The method returns a builder (`WebResource.Builder` object).

Notes

- For the Java types available for an entity parameter, see *25.15.1 Combination of Java types and MIME media types available for an HTTP request entity*.

entity(Object entity, MediaType type) method

Description

This method sets the MIME media type of an HTTP request entity.

For the Java types available for an HTTP request entity, see *25.15.1 Combination of Java types and MIME media types available for an HTTP request entity*. You can use the `GenericEntity` object to specify generics in the entity.

Syntax

```
public T entity(Object entity,
MediaType type)
```

Parameters

`entity`
This is an HTTP request entity.

`type`
This is a MIME media type.

Return value

The method returns a builder (`WebResource.Builder` object).

Notes

- A non-null value specified in the `type` parameter is set in the `Content-Type HTTP` header. A null value is ignored and is not set in the `Content-Type HTTP` header.
- The value returned by the `toString()` method of the `MediaType` object is set in the `Content-Type HTTP` header. The JAX-RS engine does not validate the values returned by the `toString()` method. Specify the value according to the standard specifications.
- You can also use the following methods to change the value of the `Content-Type HTTP` header:
  - `ClientRequest.Builder` class
    `entity(Object entity, MediaType type)` method
    `entity(Object entity, String type)` method
    `header(String name, Object value)` method
    `type(MediaType type)` method
    `type(String type)` method
  - `WebResource` class
    `entity(Object entity, MediaType type)` method
    `entity(Object entity, String type)` method
    `header(String name, Object value)` method
    `type(MediaType type)` method
    `type(String type)` method
  - `WebResource.Builder` class
    `entity(Object entity, String type)` method
    `header(String name, Object value)` method
    `type(MediaType type)` method
    `type(String type)` method

The operation when the MIME media type is not set with these methods and the `entity(Object entity, MediaType type)` method is the same as the operation when the `Content-Type HTTP` header is not set in the `HttpURLConnection` object before performing HTTP communication.

entity(Object entity, String type) method

Description

This method sets the MIME media type of an HTTP request entity.

For the Java types available for an HTTP request entity, see *25.15.1 Combination of Java types and MIME media types available for an HTTP request entity*. You can use the `GenericEntity` object to specify generics in the entity.

Syntax

```
public T entity(Object entity,
String type)
```

Parameters

`entity`
This is an HTTP request entity.

`type`
This is a MIME media type.

Return value

The method returns a builder (`WebResource.Builder` object).

Notes

- If `null` or an empty string is specified in the `type` parameter, the `IllegalArgumentException` exception is thrown.
- The non-null value specified in the `type` parameter is set in the `Content-Type HTTP` header.
- The value returned by the `toString()` method of the `MediaType` object is set in the `Content-Type HTTP` header. The JAX-RS engine does not validate the values returned by the `toString()` method. Specify the value according to the standard specifications.
- You can also use the following methods to change the value of the `Content-Type HTTP` header:
  - `ClientRequest.Builder` class
    `entity(Object entity, MediaType type)` method
    `entity(Object entity, String type)` method
    `header(String name, Object value)` method
    `type(MediaType type)` method
    `type(String type)` method
  - `WebResource` class
    `entity(Object entity, MediaType type)` method
    `entity(Object entity, String type)` method
    `header(String name, Object value)` method
    `type(MediaType type)` method
    `type(String type)` method
  - `WebResource.Builder` class

`entity(Object entity, MediaType type)` method
`header(String name, Object value)` method
`type(MediaType type)` method
`type(String type)` method

The operation when the MIME media type is not set with these methods and the `entity(Object entity, StringType)` method is the same as the operation when the `Content-Type HTTP` header is not set in the `HttpURLConnection` object before performing HTTP communication.

get(Class<T> c) method

Description

This method invokes the `HTTP GET` method.

Syntax

```
public <T> T get(Class<T> c)
throws UniformInterfaceException
```

Parameter

`c`

This is the type of an HTTP response entity.

Return value

The `get(Class<T> c)` method returns the object of the specified type.

Exception

`UniformInterfaceException`

This exception is thrown if any of the following conditions is fulfilled. The error message KDJJ18888-E is output to the log.

- If the status code of the HTTP response is 204
- If the status code of the HTTP response is 300 or more and the `c` parameter is not of the `ClientResponse` type

Notes

- For the Java types available for an HTTP response entity, see *25.15.2 Combination of Java types and MIME media types available for an HTTP response entity*.
- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown.

get(GenericType<T> gt) method

Description

This method invokes the `HTTP GET` method.

Syntax

```
public <T> T get(GenericType<T> gt)
throws UniformInterfaceException
```

Parameter

`gt`

This is the `GenericType` object that expresses the type of an HTTP response entity.

Return value

The `get(GenericType<T> gt)` method returns the object of the type expressed by the specified `GenericType` object.

Exception

`UniformInterfaceException`

This exception is thrown if any of the following conditions is fulfilled. The error message KDJJ18888-E is output to the log.

- If the status code of the HTTP response is 204
- If the status code of the HTTP response is 300 or more and the `gt` parameter does not express the `ClientResponse` type

Notes

- For the Java types available for an HTTP response entity, see *25.15.2 Combination of Java types and MIME media types available for an HTTP response entity*.
- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown.

head() method

Description

This method invokes the `HTTP HEAD` method.

Syntax

```
public ClientResponse head()
```

Parameter

None.

Return value

The `head()` method returns an HTTP response.

Notes

- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown.

header(String name, Object value) method

Description

This method adds an HTTP header.

Syntax

```
public T header(String name,
Object value)
```

Parameters

`name`
This is the name of an HTTP header.

`value`
This is the value of an HTTP header.

Return value

The method returns the builder (`WebResource.Builder` object).

Notes

- If the value of both `name` and `value` parameters is `null`, the method is not invoked.
- Even if the value of the `name` parameter is not null, but if the value of the `value` parameter is `null`, the method is not invoked.
- If the value of the `name` parameter is `null` and the value of the `value` parameter is not null, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps the `NullPointerException` exception is thrown.
- You cannot set the HTTP headers `Content-Length`, `Connection`, or `Host` with this method. If you specify these headers in the `name` parameter, the method is not invoked even if the value of the `value` parameter is not null. Note that `HttpURLConnection` sets each of the HTTP headers.
- The value returned by the `toString()` method of a non-null object specified with the `value` parameter is set as the value of the HTTP header specified with the `name` parameter. The JAX-RS engine does not validate the values returned by the `toString()` method. Specify the value in the `value` parameter according to the standard specifications.
- You can also use the following methods to add the HTTP headers `accept`, `acceptLanguage`, and `cookie`:
  - `ClientRequest.Builder` class
    `accept(MediaType... types)` method
    `accept(String... types)` method
    `acceptLanguage(Locale... locales)` method
    `acceptLanguage(String... locales)` method
    `cookie(Cookie cookie)` method
  - `WebResource` class
    `accept(MediaType... types)` method
    `accept(String... types)` method
    `acceptLanguage(Locale... locales)` method
    `acceptLanguage(String... locales)` method
    `cookie(Cookie cookie)` method
  - `WebResource.Builder` class

`accept(MediaType... types)` method
`accept(String... types)` method
`acceptLanguage(Locale... locales)` method
`acceptLanguage(String... locales)` method
`cookie(Cookie cookie)` method

The operation when none of the HTTP headers is added with these methods and the `header(String name, Object value)` method is the same as the operation when none of the HTTP headers is added in the `HttpURLConnection` object.

- If the `Content-Type HTTP` header is already set by using the following methods, the value is overwritten by the value of the `value` parameter:
  - `ClientRequest.Builder` class
    `entity(Object entity, MediaType type)` method
    `entity(Object entity, String type)` method
    `header(String name, Object value)` method
    `type(MediaType type)` method
    `type(String type)` method
  - `WebResource` class
    `entity(Object entity, MediaType type)` method
    `entity(Object entity, String type)` method
    `header(String name, Object value)` method
    `type(MediaType type)` method
    `type(String type)` method
  - `WebResource.Builder` class
    `entity(Object entity, MediaType type)` method
    `entity(Object entity, String type)` method
    `type(MediaType type)` method
    `type(String type)` method

  The operation when the MIME media type is not set with these methods and the `header(String name, Object value)` method is the same as the operation when the `Content-Type HTTP` header is not set in the `HttpURLConnection` object before performing HTTP communication.

method(String method) method

Description

This method invokes the HTTP method. The `method(String method)` method does not include any entity in the request and does not receive any response entity.

An entity is ignored if the status code is below 300 and the entity is included in a response.

Syntax

```
public void method(String method)
throws UniformInterfaceException
```

Parameter

`method`
This is the name of an HTTP method.

Return value

None.

Exception

`UniformInterfaceException`

This exception is thrown if the HTTP response status code is 300 or more. The error message KDJJ18888-E is output to the log.

Notes

- Specify uppercase `GET`, `HEAD`, `POST`, `OPTIONS`, `PUT`, or `DELETE` in the `method` parameter.
- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown.

method(String method, Class<T> c) method

Description

This method invokes the HTTP method. The `method(String method, Class<T> c)` method does not include any entity in the request, but receives a response entity.

Syntax

```
public <T> T method(String method,
Class<T> c)
throws UniformInterfaceException
```

Parameters

`method`

This is an HTTP method.

`c`

This is the type of an HTTP response entity.

Return value

The `method(String method, Class<T> c)` method returns the object of the specified type.

Exception

`UniformInterfaceException`

This exception is thrown if any of the following conditions is fulfilled. The error message KDJJ18888-E is output to the log.

- If the status code of the HTTP response is 204
- If the status code of the HTTP response is 300 or more and the `c` parameter is not of the `ClientResponse` type

Notes

- Specify uppercase `GET`, `HEAD`, `POST`, `OPTIONS`, `PUT`, or `DELETE` in the `method` parameter.

- For the Java types available for an HTTP response entity, see *25.15.2 Combination of Java types and MIME media types available for an HTTP response entity*.
- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown.

method(String method, Class<T> c, Object requestEntity) method

Description

This method invokes the HTTP method. The `method(String method, Class<T> c, Object requestEntity)` method does not include any entities in the request, but receives a response entity.

Syntax

```
public <T> T method(String method,
Class<T> c,
Object requestEntity)
throws UniformInterfaceException
```

Parameters

`method`
This is an HTTP method.

`c`
This is the type of an HTTP response entity.

`requestEntity`
This is an HTTP request entity.

Return value

The `method(String method, Class<T> c, Object requestEntity)` method returns the object of the specified type.

Exception

`UniformInterfaceException`
This exception is thrown if any of the following conditions is fulfilled. The error message KDJJ18888-E is output to the log.

- If the status code of the HTTP response is 204
- If the status code of the HTTP response is 300 or more and the `c` parameter is not of the `ClientResponse` type

Notes

- Specify uppercase `GET`, `HEAD`, `POST`, `OPTIONS`, `PUT`, or `DELETE` in the `method` parameter.
- For the Java types available for an HTTP request entity, see *25.15.1 Combination of Java types and MIME media types available for an HTTP request entity*.
- For the Java types available for an HTTP response entity, see *25.15.2 Combination of Java types and MIME media types available for an HTTP response entity*.
- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown.

method(String method, GenericType<T> gt) method

Description

This method invokes an `HTTP` method. The `method(String method, GenericType<T> gt)` method does not include any entity in the request, but receives a response entity.

Syntax

```
public <T> T method(String method,
GenericType<T> gt)
throws UniformInterfaceException
```

Parameters

`method`

This is an HTTP method.

`gt`

This is the `GenericType` object that expresses the type of an HTTP response entity.

Return value

The `method(String method, GenericType<T> gt)` method returns the object of the type expressed by the specified `GenericType` object.

Exception

`UniformInterfaceException`

This exception is thrown if any of the following conditions is fulfilled. The error message KDJJ18888-E is output to the log.

- If the status code of the HTTP response is 204
- If the status code of the HTTP response is 300 or more and the `gt` parameter does not express the `ClientResponse` type

Notes

- Specify uppercase `GET`, `HEAD`, `POST`, `OPTIONS`, `PUT`, or `DELETE` in the `method` parameter.
- For the Java types available for an HTTP response entity, see *25.15.2 Combination of Java types and MIME media types available for an HTTP response entity*.
- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown.

method(String method, GenericType<T> gt, Object requestEntity) method

Description

This method invokes an HTTP method. The `method(String method, GenericType<T> gt, Object requestEntity)` method does not include any entity in the request, but receives a response entity.

Syntax

```
public <T> T method(String method,
```

```
GenericType<T> gt,
Object requestEntity)
throws UniformInterfaceException
```

Parameters

`method`
This is an HTTP method.

`gt`
This is the `GenericType` object that expresses the type of an HTTP response entity.

`requestEntity`
This is an HTTP request entity.

Return value

The `method(String method, GenericType<T> gt, Object requestEntity)` method returns the object of the type expressed by the specified `GenericType` object.

Exception

`UniformInterfaceException`
This exception is thrown if any of the following conditions is fulfilled. The error message KDJJ18888-E is output to the log.

- If the status code of the HTTP response is 204
- If the status code of the HTTP response is 300 or more and the `gt` parameter does not express the `ClientResponse` type

Notes

- Specify uppercase `GET`, `HEAD`, `POST`, `OPTIONS`, `PUT`, or `DELETE` in the `method` parameter.
- For the Java types available for an HTTP request entity, see *25.15.1 Combination of Java types and MIME media types available for an HTTP request entity*.
- For the Java types available for an HTTP response entity, see *25.15.2 Combination of Java types and MIME media types available for an HTTP response entity*.
- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown.

method(String method, Object requestEntity) method

Description

This method invokes the `HTTP` method. The `method(String method, Object requestEntity)` method does not include any entities in the request, but receives a response entity.

An entity is ignored if the status code is below 300 and the entity is included in a response.

Syntax

```
public void method(String method,
Object requestEntity)
throws UniformInterfaceException
```

Parameters

`method`

This is an HTTP method.

`requestEntity`

This is an HTTP request entity.

Return value

None.

Exception

`UniformInterfaceException`

This exception is thrown if the HTTP response status code is 300 or more. The error message KDJJ18888-E is output to the log.

Notes

- Specify uppercase `GET`, `HEAD`, `POST`, `OPTIONS`, `PUT`, or `DELETE` in the `method` parameter.
- For the Java types available for an HTTP request entity, see *25.15.1 Combination of Java types and MIME media types available for an HTTP request entity*.
- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown.

options(Class<T> c) method

Description

This method invokes the `HTTP OPTIONS` method.

Syntax

```
public <T> T options(Class<T> c)
throws UniformInterfaceException
```

Parameter

`c`

This is the type of an HTTP response entity.

Return value

The `options(Class<T> c)` method returns the object of the specified type.

Exception

`UniformInterfaceException`

This exception is thrown if any of the following conditions is fulfilled. The error message KDJJ18888-E is output to the log.

- If the status code of the HTTP response is 204
- If the status code of the HTTP response is 300 or more and the `c` parameter is not of the `ClientResponse` type

Notes

- For the Java types available for an HTTP response entity, see *25.15.2 Combination of Java types and MIME media types available for an HTTP response entity*.
- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown.

options(GenericType<T> gt) method

Description

This method invokes the `HTTP OPTIONS` method.

Syntax

```
public <T> T options(GenericType<T> gt)
throws UniformInterfaceException
```

Parameter

`gt`

This is the `GenericType` object that expresses the type of an HTTP response entity.

Return value

The `options(GenericType<T> gt)` method returns the object of the type expressed by the specified `GenericType` object.

Exception

`UniformInterfaceException`

This exception is thrown if any of the following conditions is fulfilled. The error message KDJJ18888-E is output to the log.

- If the status code of the HTTP response is 204
- If the status code of the HTTP response is 300 or more and the `gt` parameter does not express the `ClientResponse` type

Notes

- For the Java types available for an HTTP response entity, see *25.15.2 Combination of Java types and MIME media types available for an HTTP response entity*.
- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown.

post() method

Description

This method invokes the `HTTP POST` method. The `post()` method does not include any entity in the request and does not receive any response entity.

An entity is ignored if the status code is below 300 and the entity is included in a response.

Syntax

```
public void post()
throws UniformInterfaceException
```

Parameter

None.

Return value

None.

Exception

`UniformInterfaceException`

This exception is thrown if the HTTP response status code is 300 or more. The error message KDJJ18888-E is output to the log.

Notes

- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown.

post(Class<T> c) method

Description

This method invokes the `HTTP POST` method. The `post(Class<T> c)` method does not include any entity in the request, but receives a response entity.

Syntax

```
public <T> T post(Class<T> c)
throws UniformInterfaceException
```

Parameter

`c`

This is the type of an HTTP response entity.

Return value

The `post(Class<T> c)` method returns the object of the specified type.

Exception

`UniformInterfaceException`

This exception is thrown if any of the following conditions is fulfilled. The error message KDJJ18888-E is output to the log.

- If the status code of the HTTP response is 204
- If the status code of the HTTP response is 300 or more and the `c` parameter is not of the `ClientResponse` type

Notes

- For the Java types available for an HTTP response entity, see *25.15.2 Combination of Java types and MIME media types available for an HTTP response entity*.
- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and `ClientHandlerException` exception that wraps this exception is thrown.

post(Class<T> c, Object requestEntity) method

Description

This method invokes the `HTTP POST` method. The `post(Class<T> c, Object requestEntity)` method includes an entity in the request and also receives a response entity.

Syntax

```
public <T> T post(Class<T> c,
Object requestEntity)
throws UniformInterfaceException
```

Parameters

`c`

This is the type of an HTTP response entity.

`requestEntity`

This is an HTTP request entity.

Return value

The `post(Class<T> c, Object requestEntity)` method returns the object of the specified type.

Exception

`UniformInterfaceException`

This exception is thrown if any of the following conditions is fulfilled. The error message KDJJ18888-E is output to the log.

- If the status code of the HTTP response is 204
- If the status code of the HTTP response is 300 or more and the `c` parameter is not of the `ClientResponse` type

Notes

- For the Java types available for an HTTP request entity, see *25.15.1 Combination of Java types and MIME media types available for an HTTP request entity*.
- For the Java types available for an HTTP response entity, see *25.15.2 Combination of Java types and MIME media types available for an HTTP response entity*.
- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown.

post(GenericType<T> gt) method

Description

This method calls the `HTTP POST` method. The `post(GenericType<T> gt)` method includes an entity in the request and also receives a response entity.

Syntax

```
public <T> T post(GenericType<T> gt)
throws UniformInterfaceException
```

Parameter

`gt`

This is the `GenericType` object that expresses the type of an HTTP response entity.

Return value

The `post(GenericType<T> gt)` method returns the object of the type expressed by the specified `GenericType` object.

Exception

`UniformInterfaceException`

This exception is thrown if any of the following conditions is fulfilled. The error message KDJJ18888-E is output to the log.

- If the status code of the HTTP response is 204
- If the status code of the HTTP response is 300 or more and the `gt` parameter does not express the `ClientResponse` type

Notes

- For the Java types available for an HTTP response entity, see *25.15.2 Combination of Java types and MIME media types available for an HTTP response entity*.
- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown.

post(GenericType<T> gt, Object requestEntity) method

Description

This method invokes the `HTTP POST` method. The `post(GenericType<T> gt,Object requestEntity)` method includes an entity in the request and also receives a response entity.

Syntax

```
public <T> T post(GenericType<T> gt,
Object requestEntity)
throws UniformInterfaceException
```

Parameters

`gt`

This is the `GenericType` object that expresses the type of an HTTP response entity.

`requestEntity`

This is an HTTP request entity.

Return value

The `post(GenericType<T> gt,Object requestEntity)` method returns the object of the type expressed by the specified `GenericType` object.

Exception

`UniformInterfaceException`

This exception is thrown if any of the following conditions is fulfilled. The error message KDJJ18888-E is output to the log.

- If the status code of the HTTP response is 204
- If the status code of the HTTP response is 300 or more and the `gt` parameter does not express the `ClientResponse` type

Notes

- For the Java types available for an HTTP request entity, see *25.15.1 Combination of Java types and MIME media types available for an HTTP request entity*.
- For the Java types available for an HTTP response entity, see *25.15.2 Combination of Java types and MIME media types available for an HTTP response entity*.
- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and `ClientHandlerException` exception that wraps this exception is thrown.

post(Object requestEntity) method

Description

This method invokes the `HTTP POST` method. The `post(Object requestEntity)` method includes an entity in the request, but does not receive a response entity.

An entity is ignored if the status code is below 300 and the entity is included in a response.

Syntax

```
public void post(Object requestEntity)
throws UniformInterfaceException
```

Parameter

`requestEntity`

This is an HTTP request entity.

Return value

None.

Exception

`UniformInterfaceException`

This exception is thrown if the HTTP response status code is 300 or more. The error message KDJJ18888-E is output to the log.

Notes

- For the Java types available for an HTTP request entity, see *25.15.1 Combination of Java types and MIME media types available for an HTTP request entity*.
- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and `ClientHandlerException` exception that wraps this exception is thrown.

put() method

Description

This method invokes the `HTTP PUT` method. The `put()` method does not include any entity in the request and does not receive any response entity.

An entity is ignored if the status code is below 300 and the entity is included in a response.

Syntax

```
public void put()
throws UniformInterfaceException
```

Parameter

None.

Return value

None.

Exception

`UniformInterfaceException`

This exception is thrown if the HTTP response status code is 300 or more. The error message KDJJ18888-E is output to the log.

Notes

- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown.

put(Class<T> c) method

Description

This method invokes the `HTTP PUT` method. The `put(Class<T> c)` method includes an entity in the request and also receives a response entity.

Syntax

```
public <T> T put(Class<T> c)
throws UniformInterfaceException
```

Parameter

`c`

This is the type of an HTTP response entity.

Return value

The `put(Class<T> c)` method returns the object of the specified type.

Exception

`UniformInterfaceException`

This exception is thrown if any of the following conditions is fulfilled. The error message KDJJ18888-E is output to the log.

- If the status code of the HTTP response is 204
- If the status code of the HTTP response is 300 or more and the `c` parameter is not of the `ClientResponse` type

Notes

- For the Java types available for an HTTP response entity, see *25.15.2 Combination of Java types and MIME media types available for an HTTP response entity*.
- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and `ClientHandlerException` exception that wraps this exception is thrown.

put(Class<T> c, Object requestEntity) method

Description

This method invokes the `HTTP PUT` method. The `put(Class<T> c, Object requestEntity)` method includes an entity in the request and also receives a response entity.

Syntax

```
public <T> T put(Class<T> c,
Object requestEntity)
throws UniformInterfaceException
```

Parameters

`c`

This is the type of an HTTP response entity.

`requestEntity`

This is an HTTP request entity.

Return value

The `put(Class<T> c, Object requestEntity)` method returns the object of the specified type.

Exception

`UniformInterfaceException`

This exception is thrown if any of the following conditions is fulfilled. The error message KDJJ18888-E is output to the log.

- If the status code of the HTTP response is 204
- If the status code of the HTTP response is 300 or more and the `c` parameter is not of the `ClientResponse` type

Notes

- For the Java types available for an HTTP request entity, see *25.15.1 Combination of Java types and MIME media types available for an HTTP request entity*.
- For the Java types available for an HTTP response entity, see *25.15.2 Combination of Java types and MIME media types available for an HTTP response entity*.
- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and `ClientHandlerException` exception that wraps this exception is thrown.

put(GenericType<T> gt) method

Description

This method invokes the `HTTP PUT` method. The `put(GenericType<T> gt)` method includes an entity in the request and also receives a response entity.

Syntax

```
public <T> T put(GenericType<T> gt)
throws UniformInterfaceException
```

Parameter

`gt`

This is the `GenericType` object that expresses the type of an HTTP response entity.

Return value

The `put(GenericType<T> gt)` method returns the object of the type expressed by the specified `GenericType` object.

Exception

`UniformInterfaceException`

This exception is thrown if any of the following conditions is fulfilled. The error message KDJJ18888-E is output to the log.

- If the status code of the HTTP response is 204
- If the status code of the HTTP response is 300 or more and the `gt` parameter does not express the `ClientResponse` type

Notes

- For the Java types available for an HTTP response entity, see *25.15.2 Combination of Java types and MIME media types available for an HTTP response entity*.

- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and `ClientHandlerException` exception that wraps this exception is thrown.

put(GenericType<T> gt, Object requestEntity) method

Description

This method invokes the `HTTP PUT` method. The `put(GenericType<T> gt, Object requestEntity)` method includes an entity in the request and also receives a response entity.

Syntax

```
public <T> T put(GenericType<T> gt,
Object requestEntity)
throws UniformInterfaceException
```

Parameters

`gt`

This is a `GenericType` object that expresses the type of an HTTP response entity.

`requestEntity`

This is an HTTP request entity.

Return value

The `put(GenericType<T> gt, Object requestEntity)` method returns the object of the type expressed by the specified `GenericType` object.

Exception

`UniformInterfaceException`

This exception is thrown if any of the following conditions is fulfilled. The error message KDJJ18888-E is output to the log.

- If the status code of the HTTP response is 204
- If the status code of the HTTP response is 300 or more and the `gt` parameter does not express the `ClientResponse` type

Notes

- For the Java types available for an HTTP request entity, see *25.15.1 Combination of Java types and MIME media types available for an HTTP request entity*.
- For the Java types available for an HTTP response entity, see *25.15.2 Combination of Java types and MIME media types available for an HTTP response entity*.
- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and `ClientHandlerException` exception that wraps this exception is thrown.

put(Object requestEntity) method

Description

This method invokes the `HTTP PUT` method. The `put(Object requestEntity)` method includes an entity in a request and also receives a response entity.

An entity is ignored if the status code is below 300 and the entity is included in a response.

Syntax

```
public void put(Object requestEntity)
throws UniformInterfaceException
```

Parameter

`requestEntity`
This is an HTTP request entity.

Return value

The `put(Object requestEntity)` method returns the object of the type expressed by the specified `GenericType` object.

Exception

`UniformInterfaceException`
This exception is thrown if the HTTP response status code is 300 or more. The error message KDJJ18888-E is output to the log.

Notes

- For the Java types available for an HTTP request entity, see *25.15.1 Combination of Java types and MIME media types available for an HTTP request entity*.
- If an exception is thrown during the processing of an HTTP request or an HTTP response, an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps this exception is thrown.

type(MediaType type) method

Description

This method sets a MIME media type.

Syntax

```
public T type(MediaType type)
```

Parameter

`type`
This is a MIME media type.

Return value

The method returns a builder (`WebResource.Builder` object).

Notes

- The non-null value specified in the `type` parameter is set in the `Content-Type HTTP` header. A null value is ignored and is not set in the `Content-Type HTTP` header.

- The value returned by the `toString()` method of the `MediaType` object is set in the `Content-Type HTTP` header. The JAX-RS engine does not validate the values returned by the `toString()` method. Specify the value according to the standard specifications.
- If the `Content-Type HTTP` header is already set by using the following methods, the value is overwritten:
  - `ClientRequest.Builder` class
    `entity(Object entity, MediaType type)` method
    `entity(Object entity, String type)` method
    `header(String name, Object value)` method
    `type(MediaType type)` method
    `type(String type)` method
  - `WebResource` class
    `entity(Object entity, MediaType type)` method
    `entity(Object entity, String type)` method
    `header(String name, Object value)` method
    `type(MediaType type)` method
    `type(String type)` method
  - `WebResource.Builder` class
    `entity(Object entity, MediaType type)` method
    `entity(Object entity, String type)` method
    `header(String name, Object value)` method
    `type(String type)` method

  The operation when the MIME media type is not set with these methods and the `type(MediaType type)` method is the same as the operation when the `Content-Type HTTP` header is not set in the `HttpURLConnection` object before performing HTTP communication.

type(String type) method

Description

This method sets a MIME media type.

Syntax

```
public T type(String type)
```

Parameter

`type`
This is a MIME media type.

Return value

The method returns a builder (`WebResource.Builder` object).

Notes

- If `null` or an empty string is set in the `type` parameter, the `IllegalArgumentException` exception is thrown.
- The value returned by the `toString()` method of the `MediaType` object set up by specifying the `type` parameter in the parameter of the `valueOf(String)` static method of the `MediaType` class is set in the

`Content-Type HTTP` header. The JAX-RS engine does not validate the values specified in the `type` parameter. Specify the value according to the standard specifications.

- If the `Content-Type HTTP` header is already set by using the following methods, the value is overwritten by the value of the `type` parameter:
  - `ClientRequest.Builder` class
    `entity(Object entity, MediaType type)` method
    `entity(Object entity, String type)` method
    `header(String name, Object value)` method
    `type(MediaType type)` method
    `type(String type)` method
  - `WebResource` class
    `entity(Object entity, MediaType type)` method
    `entity(Object entity, String type)` method
    `header(String name, Object value)` method
    `type(MediaType type)` method
    `type(String type)` method
  - `WebResource.Builder` class
    `entity(Object entity, MediaType type)` method
    `entity(Object entity, String type)` method
    `header(String name, Object value)` method
    `type(MediaType type)` method

  The operation when the MIME media type is not set with these methods and the `type(String type)` method is the same as the operation when the `Content-Type HTTP` header is not set in the `HttpURLConnection` object before performing HTTP communication.

25.12 Constant and method specifications and notes for the DefaultClientConfig class

This section describes the specifications for the constants and methods of the `DefaultClientConfig` class and the notes on using the constants and methods.

PROPERTY_BUFFER_RESPONSE_ENTITY_ON_EXCEPTION constant

Description

This property is used to set the buffering of the response entity for an exception.

Syntax

```
static final java.lang.String PROPERTY_BUFFER_RESPONSE_ENTITY_ON_EXCEPTION
```

Notes

- For details on the settings for the buffering of a response entity when an exception is thrown, see *13.1.2 Settings of a common definition file*. For details on the properties, see *25.1.1 Supported properties and features*.

PROPERTY_CHUNKED_ENCODING_SIZEconstant

Description

This property is used to set the chunked transfer encoding.

Syntax

```
static final java.lang.String PROPERTY_CHUNKED_ENCODING_SIZE
```

Notes

- For details on the chunked transfer encoding, see *13.1.2 Settings of a common definition file*. For details on the properties, see *25.1.1 Supported properties and features*.

PROPERTY_CONNECT_TIMEOUT constant

Description

This property is used to set the timeout value of a client socket connection.

Syntax

```
static final java.lang.String PROPERTY_CONNECT_TIMEOUT
```

Notes

- For details on the settings for the client socket connection timeout, see *13.1.2 Settings of a common definition file*. For details on the properties, see *25.1.1 Supported properties and features*.

PROPERTY_FOLLOW_REDIRECTS constant

Description

This property is used to specify the automatic redirect settings.

Syntax

```
static final java.lang.String PROPERTY_FOLLOW_REDIRECTS
```

Notes

- For details on the automatic redirect settings, see *13.1.2 Settings of a common definition file*. For details on the properties, see *25.1.1 Supported properties and features*.

PROPERTY_READ_TIMEOUT constant

Description

This property is used to set the read timeout value of the client socket.

Syntax

```
static final java.lang.String PROPERTY_READ_TIMEOUT
```

Notes

- For details on the settings for the client socket connection timeout, see *13.1.2 Settings of a common definition file*. For details on the properties, see *25.1.1 Supported properties and features*.

getPropertyAsFeature(String featureName) method

Description

This method acquires the Boolean type property as a feature from the property map.

Syntax

```
public boolean getPropertyAsFeature(String name)
```

Parameter

`featurename`
This is the property name (feature name) to be acquired.

Return value

The method returns `true` when a Boolean type property of the name specified in the property map exists and is `true`, and in other cases, the method returns `false`.

Notes

None.

getFeatures() method

Description

This method acquires a map containing all the features.

Syntax

```
public Map<String,Boolean> getFeatures()
```

Parameter

None.

Return value

The method returns a feature map. The method does not return `null`.

Notes

An example of using the `getFeatures()` method is as follows:

```
// Generate a ClientConfig object
ClientConfig cc = new DefaultClientConfig();
// Add a feature to enable JSON POJO mapping
// in the ClientConfig object
cc.getFeatures().put(JSONConfiguration.FEATURE_POJO_MAPPING, true);
// Specify the ClientConfig object generated above and
// create the Client object
Client client = Client.create(cc);
// Set the property read timeout
client.getProperties().put(ClientConfig.PROPERTY_READ_TIMEOUT, 10000);

// Generate a POJO object that maps to the JSON format
Pojo pojo = new Pojo();

// Generate an HTTP request and send the mapped POJO object
ClientRequest cRequest = ClientRequest.create().entity(pojo).type("application/
json").build(new URI("http://example.com/example"), "POST");
try{
 // Receive the HTTP response as the ClientResponse object
 ClientResponse cResponse = client.handle(cRequest);
} catch(ClientHandlerException e){
 // Execute the appropriate processing
}
```

In this example, initially a changeable property map is acquired with the `getFeatures` method and then, the feature to enable the JSON POJO mapping is added. Then, the `ClientRequest` object is created, the `handle()` method of the `Client` class is used to perform the HTTP communication, and the HTTP response is received as the `ClientResponse` object. The POJO object that was sent as the HTTP request entity is mapped in the JSON format according to the JSON POJO mapping. This is achieved with the features added for enabling the JSON POJO mapping.

getFeature(String featureName) method

Description

This method acquires the value of a feature.

Syntax

```
public boolean getFeature(String featureName)
```

Parameter

`featureName`
This is the name of a feature.

Return value

If the specified feature name exists and the value is `true`, the method returns `true`, and in the other cases, the method returns `false`.

Notes

- The method returns `false` if `null` or empty string is set in the `featureName` parameter.

getProperties() method

Description

This method acquires all the feature maps related to the client.

Syntax

```
public Map<String,Object> getProperties()
```

Parameter

None.

Return value

The method returns the feature map. The method does not return `null`.

Notes

An example of using the `getProperties()` method is as follows:

```
// Set up the client
ClientConfig cc = new DefaultClientConfig();

// Set the property read timeout value
cc.getProperties().put(ClientConfig.PROPERTY_READ_TIMEOUT, 10000);

// Generate the Client object by using client settings
Client client = Client.create(cc);

// Generate an HTTP request
ClientRequest cRequest= ClientRequest.create().build(new URI ("http://example.com/
example"), "GET");
try{
 //Receive the HTTP response as the ClientResponse object
 ClientResponse cResponse = client.handle(cRequest);
} catch(ClientHandlerException e){
 // Execute the appropriate processing
}
```

In this example, initially a changeable property map is acquired with the `getProperties` method and the read timeout value is set to 10,000 milliseconds. A client instance is created by using these client settings with the `create(ClientConfig cc)` method of the `Client` class. Then, the `ClientRequest` object is created, the HTTP communication is performed by using the `handle` method of the `Client` class, and the HTTP response is received as the `ClientResponse` object. If the read timeout occurs before an HTTP response is completely read,

an error (KDJJ18888-E) occurs and the `ClientHandlerException` exception that wraps the `SocketTimeoutException` exception is thrown.

getProperty(String propertyName) method

Description

This method acquires the value of the property.

Syntax

```
public Object getProperty(String propertyName)
```

Parameter

`propertyName`

This is a property name.

Return value

The method returns the value of the property. The method returns `null` if the parameter of the name specified in the `propertyName` parameter does not exist.

Notes

- The method returns `null` if `null` or an empty string is specified in the `propertyName` parameter.

25.13 Specifications for the constant, constructors, and methods and the notes for the HTTPSProperties class

This section describes the specifications for the constants, constructors, and methods of the `HTTPSProperties` class and the notes on using the constants, constructors, and methods.

PROPERTY_HTTPS_PROPERTIES constant

Description

This property is used to set the `HTTPSProperties` object. Use this property to add the `HTTPSProperties` object in the changeable property map. For details on the properties, see *25.1.1 Supported properties and features*.

Syntax

```
public static final java.lang.String PROPERTY_HTTPS_PROPERTIES
```

HTTPSProperties() constructor

Description

This method sets up the `HTTPSProperties` object from the `SSLContext` object that is generated by specifying `SSL` in the argument of the `getInstance()` method of the `javax.net.ssl.SSLContext` class. The set up `HTTPSProperties` object does not include the `javax.net.ssl.HostnameVerifier` object.

Syntax

```
public HTTPSProperties()
throws java.security.NoSuchAlgorithmException
```

Parameter

None.

Return value

None.

Exception

`NoSuchAlgorithmException`
This exception is thrown if the `SSLContext` object cannot be generated.

Notes

None.

HTTPSProperties(Hostname Verifier hv) constructor

Description

This method sets up an `HTTPSProperties` object from the `javax.net.ssl.HostnameVerifier` object specified in the argument and the `SSLContext` object that is generated by specifying `SSL` in the argument of the `getInstance()` method of the `javax.net.ssl.SSLContext` class.

Syntax

```
public HTTPSProperties(HostnameVerifier hv)
throws NoSuchAlgorithmException
```

Parameter

`hv`

This is the `HostnameVerifier` object to be specified in the `HTTPSProperties` object to be set up.

Return value

None.

Exception

`NoSuchAlgorithmException`

This exception is thrown if the `SSLContext` object cannot be generated.

Notes

- The operation when a null value is set in the `hv` parameter is the same as the operation when the `HostnameVerifier` object is not set in the `HttpsURLConnection` class before generating an HTTP request.
- The operation when a non-null value is specified in the `hv` parameter is the same as the operation when the same value is specified in the `setHostnameVerifier()` method of the `HttpsURLConnection` class. Note that the JAX-RS engine does not validate the value specified in the parameter. Specify the parameter value according to the standard specifications.

HTTPSProperties(Hostname Verifier hv, SSLContext c) constructor

Description

This method sets up the `HTTPSProperties` object from the `javax.net.ssl.HostnameVerifier` and `javax.net.ssl.SSLContext` objects specified in the argument.

Syntax

```
public HTTPSProperties(HostnameVerifier hv,
SSLContext c)
```

Parameters

`hv`

This is the `HostnameVerifier` object to be specified in the `HTTPSProperties` object to be set up.

`c`

This is `SSLContext` to be specified in the `HTTPSProperties` object to be set up. You cannot specify `null`.

Return value

None.

Notes

- If `null` is set in the `c` parameter, the `IllegalArgumentException` exception is thrown.
- The operation when `null` is set in the `hv` parameter is the same as the operation when a `HostnameVerifier` object is not set in the `HttpsURLConnection` class before generating an HTTP request.
- The operation when a non-null value is specified in the `hv` parameter is the same as the operation when the same value is specified in the `setHostnameVerifier()` method of the `HttpsURLConnection` class.
- The operation when a non-null value is specified in the `c` parameter is the same as the operation when the value acquired by invoking the `getSocketFactory()` method for the `c` parameter is specified in the `setSSLSocketFactory()` method of the `HttpsURLConnection` class.
- The JAX-RS engine does not validate the value specified in the parameter. Specify the parameter value according to the standard specifications.

getHostnameVerifier() method

Description

This method acquires the `HostnameVerifier` object.

Syntax

```
public HostnameVerifier getHostnameVerifier()
```

Parameter

None.

Return value

The method returns the `HostnameVerifier` object. The method returns `null` if the `HostnameVerifier` object is not set in this object (`HTTPSProperties` object).

Notes

- You can set the `HostnameVerifier` object with the `HTTPSProperties (HostnameVerifier hv)` constructor and the `HTTPSProperties (HostnameVerifier hv, SSLContext c)` constructor.

getSSLContext() method

Description

This method acquires the `SSLContext` object.

Syntax

```
public SSLContext getSSLContext()
```

Parameter

None.

Return value

The method returns the `SSLContext` object. The method returns `null` when the `SSLContext` object is not set in this object (`HTTPSProperties` object).

Notes

- You can set the `SSLContext` object with the `HTTPSProperties (HostnameVerifier hv, SSLContext)` constructor.

25.14 Constructor and method specifications and notes for the MultivaluedMapImpl class

The `MultivaluedMapImpl` class is an implementation class of the `javax.ws.rs.core.MultivaluedMap` interface. For details on the specifications for the methods of the `MultivaluedMapImpl` class, see the JAX-RS API documentation.

This section describes the notes on using the constructors and methods of the `MultivaluedMapImpl` class.

- The `MultivaluedMapImpl` class is not thread safe.
- The `MultivaluedMapImpl` class is a simple implementation class and only provides minimum functions. You can only use the `add` method, `getFirst` method, and `putSingle` method of the `javax.ws.rs.core.MultivaluedMap` interface.

  You cannot invoke the other methods of the `java.util.Map` interface (such as the `put` method and `remove` method). If the methods other than the `add` method, `getFirst` method, and `putSingle` method are used, the operations of the JAX-RS engine are not guaranteed.
- If the functionality of the `MultivaluedMapImpl` class is not available, separately create a class where the `javax.ws.rs.core.MultivaluedMap` interface is implemented.
- If `null` is specified in the `value` parameter for the `add(String key, String value)` method and `putSingle(String key, String value)` method, an empty string ("") is added as a value instead of `null`.

25.15 Combinations of available Java types and MIME media types

This section describes the combinations of Java types and MIME Media types that can be used as entity parameters and return values of the client APIs for RESTful Web Services. Do not use the annotations compliant with the JAXB specifications in POJO. If such annotations are used, the actual operation might differ from the description.

25.15.1 Combination of Java types and MIME media types available for an HTTP request entity

The following table lists the combination of Java types and MIME media types available for the entity body of an HTTP request. Note that the operation of the JAX-RS engine is not guaranteed for other combinations.

Table 25-6: Combination of Java types and MIME media types available for the entity body

No.	Java Type	charset[#1]	Content-Type HTTP header	Content-Type HTTP header[#2] (JAX-RS engine)
1	`byte[]`	N	Any (`*/*`)	`application/octet-stream`
2	`java.lang.String`	Y	Any (`*/*`)	`text/plain`
3	`java.io.InputStream`	N	Any (`*/*`)	`application/octet-stream`
4	`java.io.Reader`	Y	Any (`*/*`)	`text/plain`
5	`java.io.File`	N	Any (`*/*`)	`application/octet-stream`
6	`javax.activation.DataSource`	N	Any (`*/*`)	`application/octet-stream`
7	`javax.xml.transform.Source`[#3]	N	Any (`*/*`)	`application/xml`
8	`javax.xml.bind.JAXBElement<String>`[#4]	Y	`ext/xml`, `application/xml`, `application/*+xml`	`application/xml`
9	`JAXB` class[#4] annotated with the `XmlRootElement` annotation	Y	`text/xml`, `application/xml`, `application/*+xml`	`application/xml`
10	`javax.ws.rs.core.MultivaluedMap<String,String>`	Y	`application/x-www-form-urlencoded`	`application/x-www-form-urlencoded`
11	`javax.ws.rs.core.StreamingOutput`	N	Any (`*/*`)	`application/octet-stream`
12	`org.w3c.dom.Document`	N	Any (`*/*`)	`application/xml`
13	`java.awt.image.RenderedImage`	N	`image/jpeg`	`application/octet-stream`
14	`javax.ws.rs.core.GenericEntity<T>`[#5]	D	MIME media type similar to the Type specified in T.	MIME media type similar to the Type specified in T.
15	`POJO`[#6]	N[#7]	`application/json`	`application/octet-stream`

Legend:

Any (*/*): Indicates that all the MIME media types are supported.

#1

Indicates whether the charset parameter information, if included in the Content-Type HTTP header, is considered during conversion to an HTTP request.

Y: Considered. When the Content-Type HTTP header does not include the charset parameter, UTF-8 is assumed.

D: Depends on the type specified in T.

N: Not considered.

#2

If the Content-Type HTTP header is not specified in the ClientRequest object or the WebResource object and when the entity is non-null, specify the Content-Type HTTP header of an HTTP request by considering the Content-Type specified in this column.

The operation when the entity is null is the same as the operation when the Content-Type HTTP header is not set in the HttpURLConnection object before making an HTTP request.

#3

You can use the following implementation classes:

- javax.xml.transform.stream.StreamSource
- javax.xml.transform.sax.SAXSource
- javax.xml.transform.dom.DOMSource

#4

When the MIME Media type is application/fastinfoset or application/json, the operation ends normally without any error.

#5

In T, you can specify the types described in No.1 to No.13, and No. 15 in the table.

#6

Enable JSON POJO mapping. The operation when JSON POJO mapping is disabled is the same as the operation when an unsupported Java type is specified in the entity parameter. For details on how to enable JSON POJO mapping, see *18. Mapping JSON and POJO*.

#7

Do not add the charset parameter in the Content-Type HTTP header.

If the specified Java type is an unsupported type and if the value is not null, an error occurs (KDJJ10032-E and KDJJ18888-E) and the ClientHandlerException exception is thrown. However, if the specified Java type is javax.mail.internet.MimeMultipart and the MIME media type is multipart/*, the operation ends normally without any error.

With the following Java types, if there is a MIME media type that an HTTP request entity body cannot use, an error occurs (KDJJ10032-E and KDJJ18888-E) and the ClientHandlerException exception is thrown:

1. javax.xml.bind.JAXBElement <String>
2. JAXB class annotated with the XmlRootElement annotation
3. javax.ws.rs.core.MultivaluedMap<String, String>
4. java.awt.image.RenderedImage
5. POJO

However, if the MIME media type of an HTTP request entity body in 1 or 2 is application/fastinfoset or application/json, the operation ends normally without any errors.

If an exception is thrown during the processing of an HTTP request and HTTP response, an error (KDJJ18888-E) occurs and the ClientHandlerException exception that wraps this exception is thrown.

Notes for the types of HTTP methods are as follows:

- If the HTTP method is DELETE, HEAD, or OPTIONS, do not specify a non-null value in the entity body of an HTTP request. If you specify a non-null value, an error occurs (KDJJ18888-E) and the ClientHandlerException exception is thrown.
- If the HTTP method is GET, do not specify a non-null value in the entity body of an HTTP request. If specified, the non-null value is ignored.

- If the HTTP method is `POST` or `PUT`, and `null` is specified in the entity body of an HTTP request, the entity body is not included in the sent HTTP request.

25.15.2 Combination of Java types and MIME media types available for an HTTP response entity

The following table lists the combination of Java types and MIME media types that can be used in the return values.

Table 25-7: Combination of Java types and MIME media types that can be used in the return value of an entity body

No.	Java Type	charset[#1]	MIME Media Type
1	`byte[]`	N	Any (`*/*`)
2	`java.lang.String`	Y	Any (`*/*`)
3	`java.io.InputStream`	N	Any (`*/*`)
4	`java.io.Reader`	Y	Any (`*/*`)
5	`java.io.File`[#2]	N	Any (`*/*`)
6	`javax.activation.DataSource`	N	Any (`*/*`)
7	`javax.xml.transform.Source`[#3]	N	Any (`*/*`)
8	`javax.xml.bind.JAXBElement<String>`[#4, #5]	N	`text/xml`, `application/xml`, `application/*+xml`
9	`JAXB` class annotated with the `XmlRootElement` annotation and/or the `XmlType` annotation[#5]	N	`text/xml`, `application/xml`, `application/*+xml`
10	`javax.ws.rs.core.MultivaluedMap<String ,String>`	Y	`application/x-www-form-urlencoded`
11	`org.w3c.dom.Document`	N	Any (`*/*`)
12	`java.awt.image.RenderedImage`	N	`application/octet-stream`, `image/jpeg`
13	`com.cosminexus.jersey.api.client.GenericType<T>`[#6]	D	MIME media type similar to the type specified in T.
14	`com.cosminexus.jersey.api.client.GenericType<T>`[#7]	D	MIME media type similar to the type specified in T.
15	`POJO` [#8]	Y	`application/json`

Legend:

Any (`*/*`): Indicates that all the MIME media types are supported.

#1

Indicates whether the `charset` parameter information, if included in the `Content-Type HTTP` header, is considered during injection to an HTTP response.

Y: Considered. When the `Content-Type HTTP` header does not include the `charset` parameter, UTF-8 is assumed.

D: Depends on the type specified in T.

N: Not considered.

#2

The JAX-RS engine creates a `temp` directory on the local computer and saves the temporary file.

#3

You can use the following implementation classes:

- `javax.xml.transform.stream.StreamSource`
- `javax.xml.transform.sax.SAXSource`
- `javax.xml.transform.dom.DOMSource`

Note that when you use `javax.xml.transform.stream.StreamSource` or `javax.xml.transform.sax.SAXSource`, receive the HTTP response with the generic type (`ClientResponse`) and obtain the entity object by using the `getEntity()` method of the `ClientResponse` class. At this time, make sure you invoke the `bufferEntity()` method before invoking the `getEntity()` method. For details on how to receive an HTTP response by using a generic type (`ClientResponse`), see *11.4.1 Use case of a Web resource client*.

#4

Used as T in No. 14 in the table. Operation is not guaranteed for any other usage.

#5

If the MIME media type is `application/fastinfoset` or `application/json`, the operation ends normally without any errors.

#6

Used as T in No.14 in this table. In T, you can specify the types described in No.2 to No.12, and No.15 in the table. Operation is not guaranteed for any other usage.

#7

In T, you can specify the types described from No. 2 to No. 13, and No. 15 in the table.

#8

Enable the JSON POJO mapping. The operation when the JSON POJO mapping is disabled is the same as the operation when an unsupported Java type is specified in the entity parameter. For details on how to enable the JSON POJO mapping, see *18. Mapping JSON and POJO*.

If the specified Java type is an unsupported type, an error occurs (KDJJ10031-E and KDJJ18888-E) and the `ClientHandlerException` exception is thrown.

If the return value is the following Java type and the MIME media type is a type that cannot be used by the entity body of an HTTP response, an error occurs (KDJJ10031-E or KDJJ18888-E) and the `ClientHandlerException` exception is thrown.

- `javax.xml.bind.JAXBElement <String>`
- `JAXB` class annotated with the `XmlRootElement` annotation and/or the `XmlType` annotation
- `javax.ws.rs.core.MultivaluedMap<String,String>`
- `java.awt.image.RenderedImage`
- `POJO`

If the `IOException` exception is thrown during conversion from the HTTP response entity body, an error occurs (KDJJ18888-E) and the `ClientHandlerException` exception is thrown.

If an exception other than the `IOException` exception is thrown during conversion from the HTTP response entity body, the corresponding exception is thrown. To check the log, use the J2EE server log files instead of the JAX-RS functionality log files.

With the entity parameter `java.awt.image.RenderedImage`, if the `Content-Type HTTP` header is `image/*`, an error occurs (KDJJ18888-E) and the `ClientHandlerException` exception that wraps the `java.io.IOException` exception is thrown.

With the entity parameter `com.cosminexus.jersey.api.client.GenericType<T>`, if an HTTP response contains an entity body, and if T is of the type described in No.1 in the table, an error occurs (KDJJ10031-E and KDJJ18888-E) and the `ClientHandlerException` exception is thrown.

With the entity parameter `com.cosminexus.jersey.core.provider.EntityHolder<T>`, if an HTTP response contains an entity body, an error (KDJJ10003-E) occurs and the `javax.ws.rs.WebApplicationException` exception is thrown in one of the following cases:

- When T is an unsupported type (No. 1, No.13, and No.14 of the table)
- When T is a supported type (from No. 2 to No.12, and No. 15 of the table), but the MIME media type is a type that the HTTP response entity body cannot use.

If the client APIs operate on the J2EE server, the entity body contains maximum 10,000 form parameters by default. If the entity parameter type is `javax.ws.rs.core.MultivaluedMap<String, String>`, or `com.cosminexus.jersey.core.provider.GenericType<EntityHolder<javax.ws.rs.core.MultivaluedMap<String, String>>>` and if the number of response parameters exceed the specified value, the `RuntimeException` exception is thrown. Change the value in the `webserver.connector.limit.max_parameter_count` property of the user property file for J2EE servers (`usrconf.properties`) as and when required. To check the log, use the J2EE server log files instead of the JAX-RS functionality log files.

If the `Content-Type HTTP` header does not exist in the HTTP response, the MIME media type is considered to be the `application/octet-stream`.

25.16 Thread safety of the client APIs for RESTful Web Services

The following table describes the thread safety of the client APIs for RESTful Web Services.

Table 25-8: Thread safety of the client APIs for RESTful Web Services

No.	Interface or class	Constructor/ Method/ Field	Thread Safety[#1]
`com.cosminexus.jersey.api.client` package			
1	`Client`	`create()`	Y
2	`ApplicationPath`	`create(ClientConfig cc)`	Y[#2]
3	`Consumes`	`destroy()`	N
4	`CookieParam`	`getProperties()`	N
5	`DefaultValue`	`handle(ClientRequest request)`	Y[#2]
6		`resource(String u)`	Y
7		`resource(URI u)`	Y
8		`setChunkedEncodingSize(Integer chunkSize)`	N
9		`setConnectTimeout(Integer interval)`	N
10		`setFollowRedirects(Boolean redirect)`	N
11		`setReadTimeout(Integer interval)`	N
12	`ClientRequest`	All methods	N
13	`ClientRequest.Builder`	All methods	N
14	`ClientResponse`	All methods	N
15	`GenericType`	All methods	Y
16	`WebResource`	`accept(MediaType... types)`	Y
17		`accept(String... types)`	Y
18		`acceptLanguage(Locale... locales)`	Y
19		`acceptLanguage(String... locales)`	Y
20		`cookie(Cookie cookie)`	Y
21		`delete()`	Y
22		`delete(Class<T> c)`	Y
23		`delete(Class<T> c, Object requestEntity)`	C
24		`delete(GenericType<T> gt)`	Y
25		`delete(GenericType<T> gt, Object requestEntity)`	C
26		`delete(Object requestEntity)`	C
27		`entity(Object entity)`	Y

No.	Interface or class	Constructor/ Method/ Field	Thread Safety[#1]
28	WebResource	entity(Object entity, MediaType type)	Y
29		entity(Object entity, String type)	Y
30		get(Class<T> c)	Y
31		get(GenericType<T> gt)	Y
32		getRequestBuilder()	Y
33		getURI()	Y
34		getUriBuilder()	Y
35		head()	Y
36		header(String name, Object value)	Y
37		method(String method)	Y
38		method(String method, Class<T> c)	Y
39		method(String method, Class<T> c, Object requestEntity)	C
40		method(String method, GenericType<T> gt)	Y
41		method(String method, GenericType<T> gt, Object requestEntity)	C
42		method(String method, Object requestEntity)	C
43		options(Class<T> c)	Y
44		options(GenericType<T> gt)	Y
45		path(String path)	Y
46		post()	Y
47		post(Class<T> c)	Y
48		post(Class<T> c, Object requestEntity)	C
49		post(GenericType<T> gt)	Y
50		post(GenericType<T> gt, Object requestEntity)	C
51		post(Object requestEntity)	C
52		put()	Y
53		put(Class<T> c)	Y
54		put(Class<T> c, Object requestEntity)	C
55		put(GenericType<T> gt)	Y
56		put(GenericType<T> gt, Object requestEntity)	C
57		put(Object requestEntity)	C

No.	Interface or class	Constructor/ Method/ Field	Thread Safety#1
58	`WebResource`	`queryParam(String key, String value)`	Y
59		`queryParams(MultivaluedMap<String,String>params)`	Y
60		`type(MediaType type)`	Y
61		`type(String type)`	Y
62		`uri(URI uri)`	Y
63	`WebResource.Builder`	All methods	N
`com.cosminexus.jersey.api.client.config` package			
64	`DefaultClientConfig`	All methods	N
65	`Provider`	--	Y

Legend:

Y: Thread safe.

C: Thread safe when the instance specified in the request entity (`requestEntity` parameter) is thread safe.

N: Not thread safe.

--: No method available.

#1

If the operation is not thread safe, do not invoke this method of the same object from multiple threads.

#2

The class specified in the argument is not thread-safe. Generate the instances of the classes to be specified in the argument with the respective threads.

26 WSDL Import Functionality

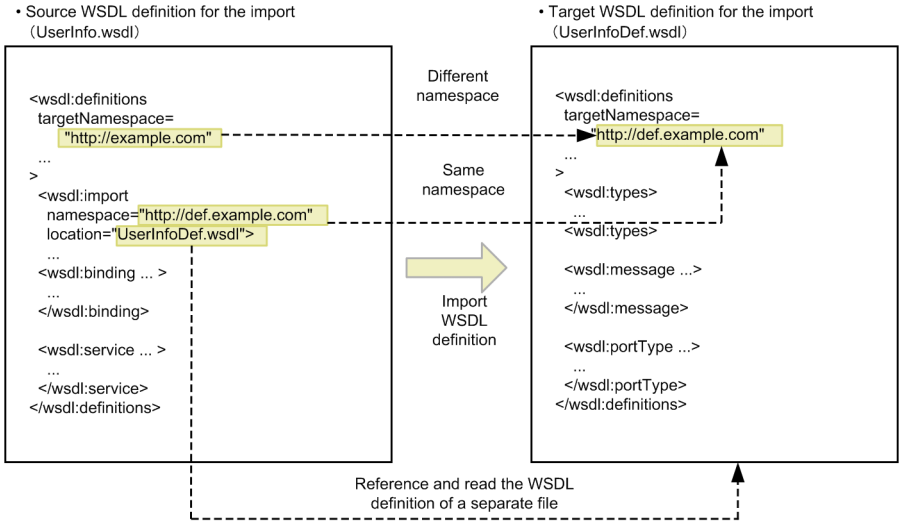
This chapter describes the WSDL import functionality.

26.1 What is the WSDL import functionality

The *WSDL import functionality* is the functionality that uses the `wsdl:import` element to read WSDL definitions in another file as a common component.

The following figure shows the image of the WSDL import functionality:

Figure 26-1: Image of the WSDL import functionality



26.2 WSDL definitions that can be imported

This section describes the conditions for the WSDL definitions to be imported and the points to remember when you import the WSDL definition.

(1) Conditions for the WSDL definitions to be imported

The WSDL definitions to be imported must fulfill the following conditions:

- `wsdl:definitions` is specified in the root element.
  If a file with a format other than `wsdl:definitions` is specified in the root element, an error message is output in the standard error output and log and the processing ends (KDJW51200-E).
- The user has access permission to the WSDL definition.
  If you specify a WSDL definition without the access permission, a JDK error occurs and the processing ends.

The extension of the WSDL definition to be imported is optional.

(2) Importing multiple WSDL definitions

You can combine and import multiple WSDL definitions. However, the `wsdl:service` element must be defined in the import source WSDL definition that configures the initial starting point. In the subsequent layers, you can import the WSDL definition by combining elements other than the `wsdl:service` element. For details about operations when the `wsdl:service` element is not defined, see *20.1.16 wsdl:service element*.

The following figures separately show the examples when the method of *hierarchical importing of WSDL definitions* is correct and when the method is wrong:

Figure 26–2: Hierarchical importing of WSDL definitions (Example of a correct method)

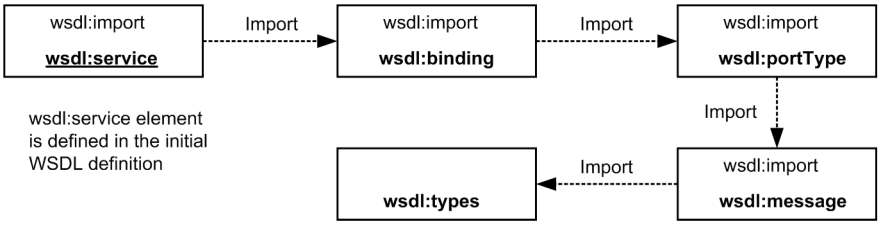
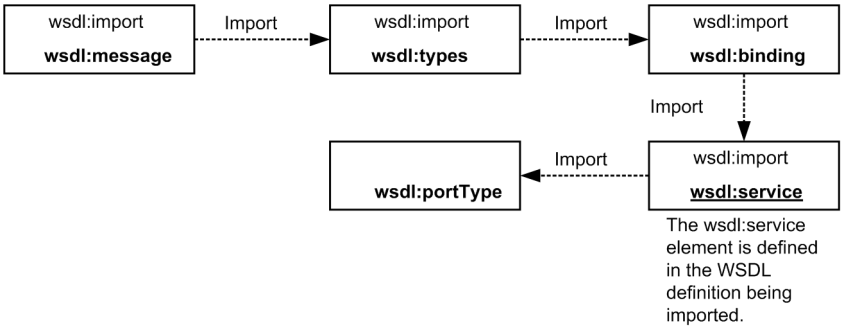


Figure 26–3: Hierarchical importing of WSDL definitions (Example of a wrong method)

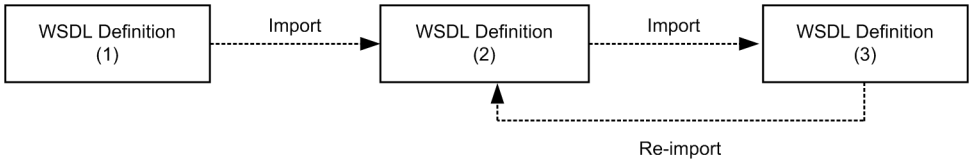


(3) Recursive importing of WSDL definitions

In the WSDL import functionality, you cannot import the WSDL definition recursively. Even if you attempt to import the definition recursively, the WSDL definition will not be read, and therefore, the `wsdl:import` element will be ignored.

The following figure shows an example of the recursive importing of WSDL definitions:

Figure 26-4: Example of the recursive importing of WSDL definitions



26.3 Format of the wsdl:import element

The following is an example of the format of the `wsdl:import` element:

```
<wsdl:import namespace="namespace-name-of-the-WSDL-definition-to-be-imported"
location="location-of-the-WSDL-definition-to-be-imported"/>
```

The following points describe the attributes of the `wsdl:import` element:

(1) namespace attribute (wsdl:import element)

Specifies the namespace name of the Import target WSDL definition.

In the `namespace` attribute of the `wsdl:import` element coded in the import source WSDL definition, specify the same namespace name as the namespace name of the Import target WSDL definition (`targetNamespace` attribute of the `wsdl:definitions` element).

The following table describes the relationship between the namespace of the import source WSDL definition and the namespace of the import target WSDL definition:

Table 26-1: Relationship of the namespaces for the WSDL definition (Import source/ import target)

No.	Import source WSDL definition	Import target WSDL definition	Conditions	Runtime operations
1	`namespace` attribute of the `wsdl:import` element	`targetNamespace` attribute of the `wsdl:definitions` element	Matching	Terminates normally.
2			Not matching	A warning message is output in the standard error output and log and the processing continues (KDJW51191-W). The elements of the import target WSDL definition and import source WSDL definition belong to the respective namespaces.
3	`targetNamespace` attribute of the `wsdl:definitions` element		Matching	A warning message is output in the standard error output and log and the processing continues (KDJW51192-W). The elements of the import target WSDL definition and import source WSDL definition belong to the same namespace.
4			Not matching	Terminates normally.

(2) location attribute (wsdl:import element)

Specifies the location of the WSDL definition to be imported.

The conditions for the strings specified in the `location` attribute are as follows:

- You can specify the WSDL file available in the relative path, remote, or local using the URL format.
- The WSDL file included in the WAR file is issued as the meta data (except when a new WSDL file is generated). However, you need to take precautions when issuing the meta data. For details about the precautions for issuing the meta data, see *10.6(6) Notes for importing/including WSDL definition or XML Schema*.
- Use characters defined in RFC 2396 and characters fulfilling `xsd:anyURI`. However, you cannot use RFC 2732 (IPv6).
- The strings are not case sensitive.
- The maximum length of the specifiable sting is not limited. However, an error occurs when you exceed the limit set for the OS.

The following table describes an example of coding of the `location` attribute:

Table 26-2: Example of specification of the location attribute (WSDL import functionality)

No.	Specified contents	Example of specification
1	Specifying the local WSDL file using the relative path[#1]	`./wsdl/input.wsdl`
2	Specifying the local WSDL file using the absolute path expressed with a URL (`file://~`)[#1, 2]	`file:///C:/tmp/wsdl/input.wsdl`
3	Specifying the remote WSDL file using a URL (`http:~`)[#3]	`http://example.com:8080/fromjava/test?wsdl`

#1

When you specify the WSDL file using the relative path or absolute path, specify the correct path. If the path is wrong and the WSDL file is not found, an error message is output in the standard error output and log and the processing ends (KDJW51197-E or KDJW51198-E).

#2

You cannot specify an absolute path string beginning with a drive such as "`C:/~`". If such a string is specified, an error message will be output in the standard error output and log and the processing will end (KDJW51199-E).

#3

When you specify a remote WSDL file using a URL, specify the correct URL. If the path is wrong and the WSDL file is not found, an error message will be output in the standard error output and log and the processing will end (KDJW51197-E or KDJW51198-E).

27 Catalog Functionality

This chapter describes the catalog functionality.

27.1 What is the catalog functionality

The catalog functionality, based on the XML Catalogs 1.1 specifications, executes mapping. You can execute mapping in the following two ways:

- Mapping the namespace URIs of XML schemas specified by using the `namespace` attribute of the `xsd:import` element and URI references that point to the XML schema locations
- Mapping URI references that point to the location of the WSDL or XML schemas and URI references that point to the different location of the WSDL or XML schemas

To map by using the catalog functionality, you must allocate a catalog file that defines the mapping information. You can use the catalog functionality when developing and starting a Web Services client. However, using the catalog functionality, when publishing metadata, is not supported.

For details on catalog files, see *27.6 Catalog file*.

27.2 Using the catalog functionality (when developing a Web Services client)

Specify a catalog file by using the `-catalog` option of the `cjwsimport` command to use the catalog functionality when developing a stub based Web Services client. For details on the `-catalog` option, see *14.1(2) List of options*.

According to the mapping information of a catalog file, the `cjwsimport` command maps the WSDL specified in an argument and the namespace or the location of an XML schema that this WSDL references, to a different WSDL or XML schema location. The `cjwsimport` command also scans the WSDL and XML schema in the mapped location and generates a Java code.

(1) Mapping targets

Table 27-1 lists mapping targets of the catalog functionality in the `cjwsimport` command and Table 27-2 lists mapping targets in the code generated by a WSDL specified in the arguments of the `cjwsimport` command.

Table 27-1: Mapping targets in the `cjwsimport` command

No.	Mapping target	Supported
1	Location of the WSDL specified in the arguments of the `cjwsimport` command	N

Legend:
N: Not supported.

Table 27-2: Mapping targets in the code generated by a WSDL specified in the arguments of the `cjwsimport` command

No.	Element	Attribute	Support
1	`wsdl:import`	namespace	N
2		location	Y
3	`xsd:import`	namespace	Y#
4		schemaLocation	Y
5	`xsd:include`	schemaLocation	Y

Legend:
Y: Supported.
N: Not supported.

#
The `namespace` attribute is mapped only when the `xsd:import` element has only the `namespace` attribute. If the `xsd:import` element contains both the `namespace` and `schemaLocation` attributes, the `namespace` attribute is not considered a mapping target.

(2) Notes

Do not change the values of the following locations and elements when using the catalog functionality when developing a Web Services client.

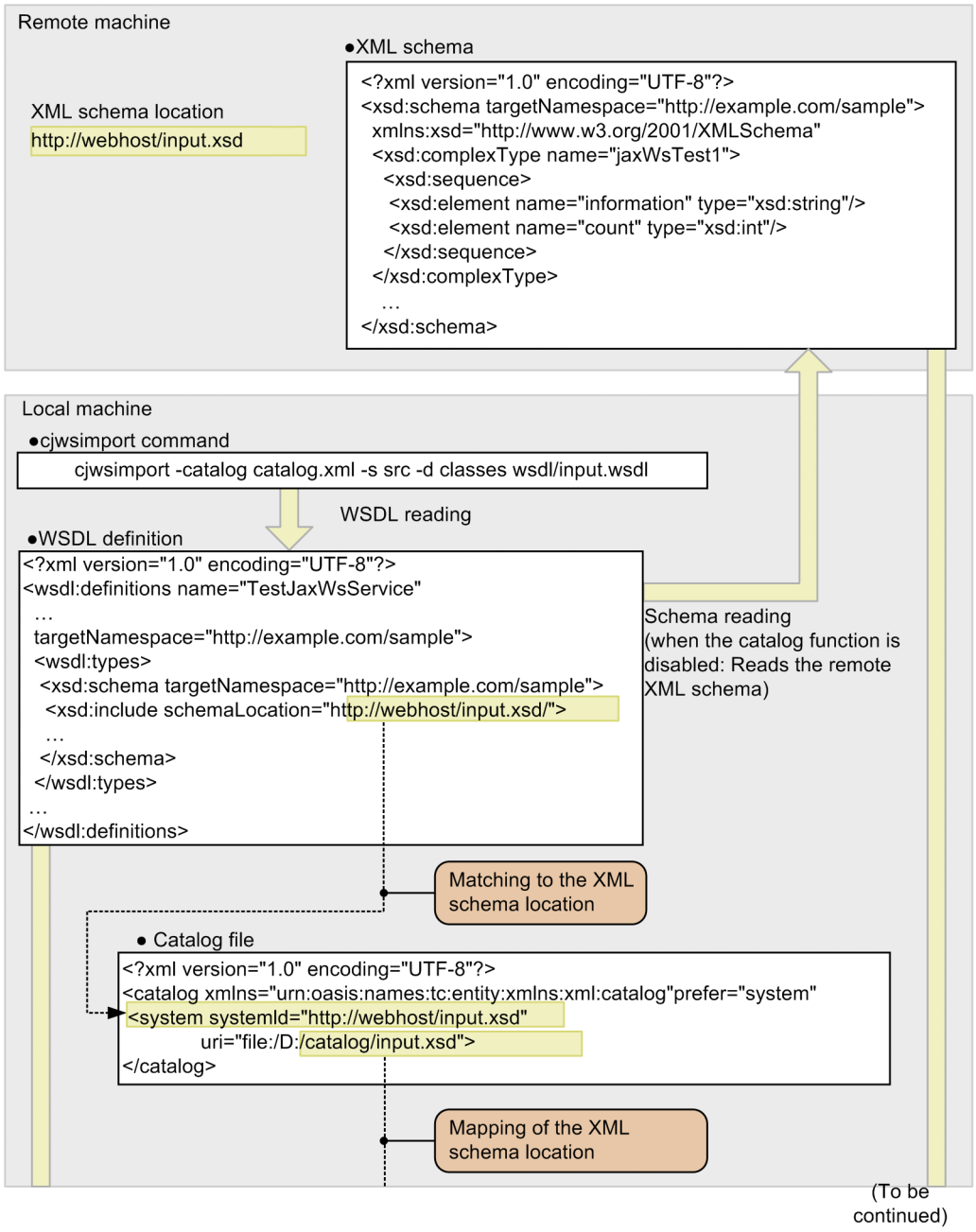
- The default WSDL location of the service class that the `cjwsimport` command generates (WSDL location that the constructor that does not have a WSDL location specified as an argument uses)
- The `wsdlLocation` element in the `javax.xml.ws.WebServiceClient` annotation to be assigned to a service class

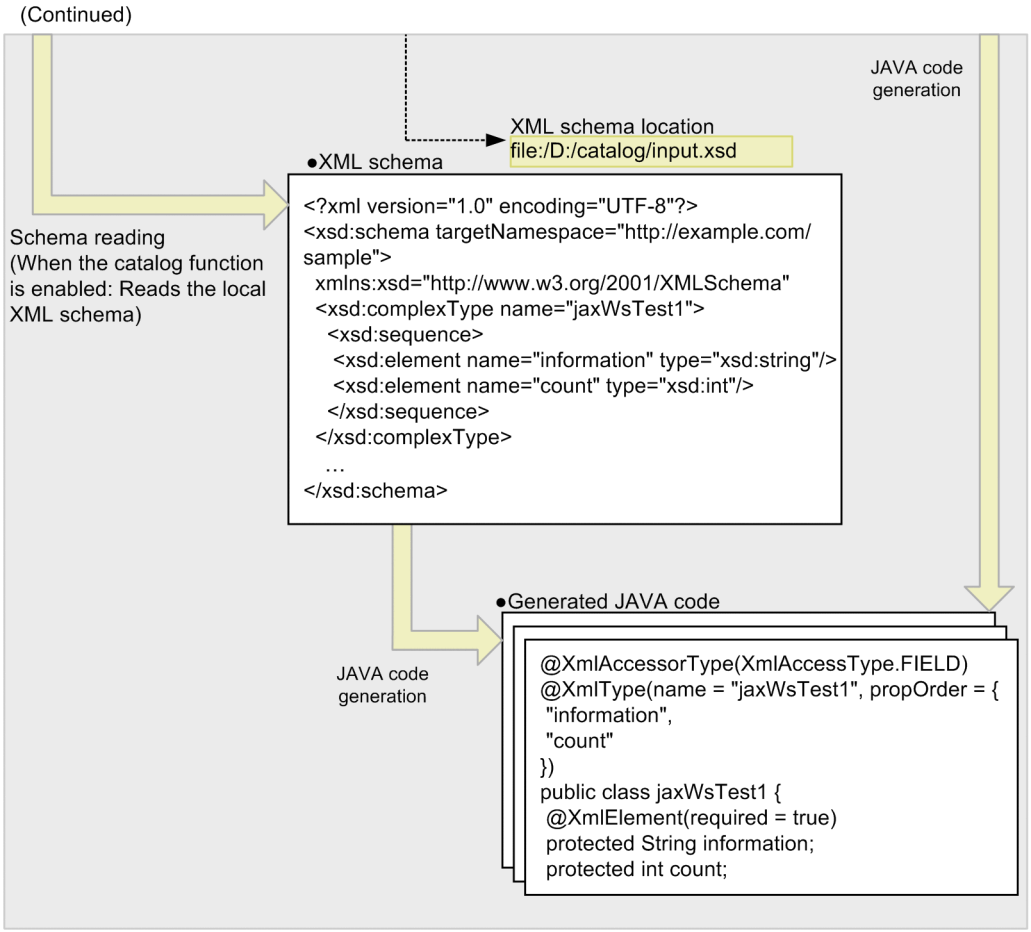
Use the `-wsdllocation` option of the `cjwsimport` command to change the values of the aforementioned locations or elements. For details, see *14.1 cjwsimport command*.

(3) Example of mapping

The following figure shows an example of mapping an XML schema location that a WSDL refers by using the catalog functionality when developing a Web Services client.

Figure 27－1: Example of mapping an XML schema location that a WSDL refers





In this figure, a remote XML schema location is specified in the `schemaLocation` attribute of the `xsd:include` element of the WSDL. The figure also shows procedures when the catalog functionality is disabled and enabled.

When the catalog functionality is disabled

The `cjwsimport` command generates a Java code by reading the remote XML schema.

When the catalog functionality is enabled

As per the information on mapping the catalog file, map the remote XML schema location specified in the `schemaLocation` attribute of the `xsd:include` element to a local XML schema location. The `cjwsimport` command reads the local XML schema and generates a Java code.

27.3 Using the catalog functionality (when starting a Web Services client)

When starting a Web Services client, you can use the catalog functionality just by storing the catalog file. You need not change the WSDL, specify the `-catalog` option by using the `cjwsimport` command, and recreate a developed Web Services client. For details on allocating the catalog file, see *27.6.2 Storing the catalog file*.

As per the information on mapping the catalog file, map the WSDL location from the service class of WSDL and the WSDL location specified in the argument of the constructor of the service class to different WSDL locations and generate a service class.

(1) Mapping targets

Table 27-3 lists the mapping targets of the catalog functionality when starting a Web Services client and Table 27-4lists the mapping targets in a WSDL code to be used when starting a Web Services client.

Table 27－3: Mapping targets of the catalog functionality when executing a Web Services client

No.	Mapping target	Supported
1	Default WSDL location of a service class[#1]	Y
2	The WSDL location specified in the arguments of the constructor of a service class[#1]	Y
3	The WSDL location specified in the arguments of the API of the `javax.xml.ws.Service` class[#2]	Y
4	The WSDL location specified in the `wsdlLocation` element of the `javax.xml.ws.WebServiceRef` annotation[#3]	Y

Legend:

Y: Supported

#1

For details on the WSDL location of a service class, see *Table 15-17The methods included in a service class* in *15.1.9 Mapping the service and port to the service class*.

#2

For details on the `javax.xml.ws.Service` class WSDL location, see *19.2.2(4) javax.xml.ws.Service class*.

#3

For details on the `javax.xml.ws.WebServiceRef` annotation, see *19.3 Support range of annotations*.

Table 27－4: 4Mapping targets in a WSDL code to be used when executing a Web Services client

No.	Element	Attribute	Supported
1	wsdl:import	namespace	N
2		location	Y

Legend:

Y: Supported.

N: Not supported.

(2) Notes

The catalog functionality, when starting the Web Services client, does not execute the following mappings because the functionality does not need XML schemas when generating a service class.

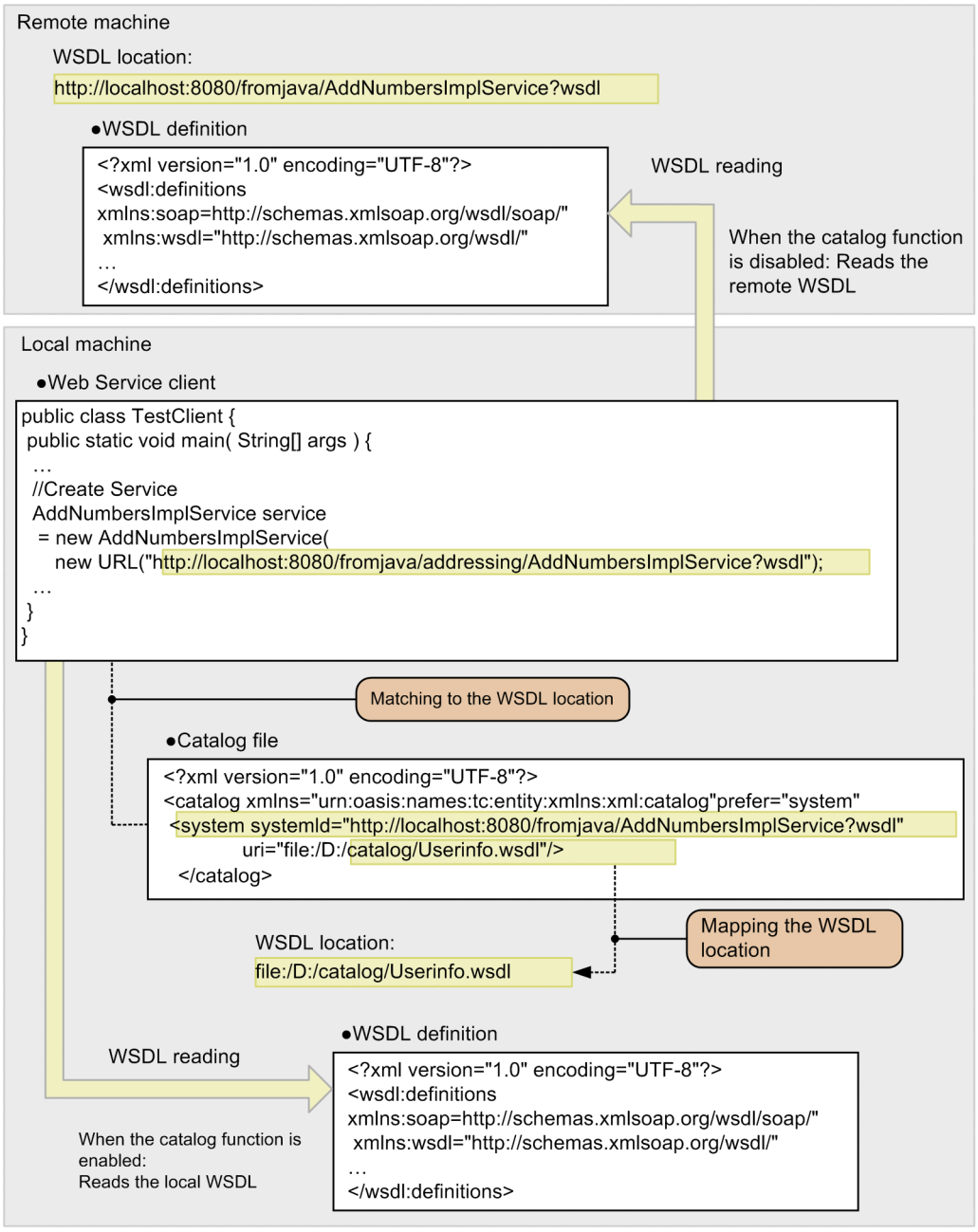
- Mapping of namespace URIs of XML schemas
- Mapping related to schema location

In the catalog file, therefore, describe the mapping information of the WSDL location only.

(3) Example of mapping

The following figure shows an example of mapping a WSDL location by using the catalog functionality, when generating a service class in a Web Services client.

Figure 27‒2: Example of mapping a WSDL location



In the example in this figure, a remote WSDL location is specified in the argument of the constructor of the service class. The figure also shows the processes when the catalog functionality is disabled and enabled.

When the catalog functionality is disabled

The Web Services client on the local machine reads the remote WSDL and generates a service class.

When the catalog functionality is enabled

As per the information on mapping the catalog file, the Web Services client on the local machine maps the remote WSDL location specified in the arguments to a local WSDL location, reads the local WSDL and generates a Java code.

27.4 Performance of the catalog functionality

Increase in the overhead when using the catalog functionality

The catalog functionality operates when calling an API of the `javax.xml.ws.Service` class or service class constructor when starting the Web Services client. Generating a service class or executing the `javax.xml.ws.Service` class APIs, therefore, increase overhead that follows reading the catalog file, as compared to mapping without using the catalog functionality.

Decrease in the overhead when using the catalog functionality

We recommend that you use the following method to decrease overhead of the catalog functionality when executing the Web Services client:

- Reusing the generated services

  For details, see *3.6.1(3) Generating service classes and acquiring ports*.
- Annotating the `javax.xml.ws.WebServiceRef` annotation to fields and injecting service classes

  For details, see *19.3 Support range of annotations*.

27.5 Notes when using the catalog functionality

- You cannot use the catalog and addressing functionalities at the same time, otherwise the operation is not guaranteed.
- When executing the Web services client, if the URI reference pointing to the WSDL location to be mapped to the URI reference pointing to the remote WSDL location by using the catalog functionality, the following properties will not be applied:
  - `com.cosminexus.jaxws.connect.timeout`
  - `com.cosminexus.jaxws.request.timeout`
  - `com.cosminexus.xml.ws.client.http.HostnameVerificationProperty`
  - `javax.xml.ws.security.auth.username`
  - `javax.xml.ws.security.auth.password`
- If you are using the catalog functionality when starting a Web Service, store the catalog file in a directory consisting single byte alphanumeric characters, (0 to 9, A t o Z, a to z), spaces, periods (.), underscores (_), colons (:), slashes (/), and ¥, otherwise the operation is not guaranteed.

27.6 Catalog file

A catalog file is stored when using the catalog functionality. This file contains the mapping information. For details on the support range of catalog files, see *21. Support Range of XML Catalogs 1.1*. This section describes the syntax of a catalog file and storing the file.

27.6.1 Syntax of the catalog file

Describe the syntax of a catalog file within the scope of XML Catalogs 1.1 schemas (W3C XML Schemas for an XML Catalog), and the JAX-WS functionality of Application Server.

27.6.2 Storing the catalog file

As per the restrictions on the file system on your OS and the Java EE 6 specifications, store the catalog file in the following places.

(1) When developing a Web Services client

Specify the path of the catalog file in the arguments of the `-catalog` option of the `cjwsimport` command. You can specify any name for the catalog file. For details, see *Notes for specifying the -catalog option* of *14.1(2) List of options*. If reading of the catalog file fails, a warning message (KDJW51221-W) is output to the standard error output and the process continues with the catalog functionality disabled.

(2) When starting the Web client

Save the catalog file as `jax-ws-catalog.xml` directly under the META-INF directory. If reading of the catalog file fails, a message (KDJW30023-W) is output to the log and the process continues with the catalog functionality disabled. The operation is not guaranteed if multiple catalog files are stored. For example, the operation is not guaranteed if multiple JAR files with the name `jax-ws-catalog.xml` are added to the class path.

27.6.3 Example of coding the catalog file

The following example shows the coding of a catalog file when mapping the remote WSDL location to a local WSDL location by using the catalog functionality:

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog" prefer="system">
    <system systemId="http://localhost:8080/fromjava/AddNumbersImplService?
wsdl"
                uri="./wsdl/UserInfo.wsdl"/>
</catalog>
```

28 Attachment Functionality (wsi:swaRef format)

If you use the attachment functionality, you can handle the text data, and also the image data and voice data with Web Services.

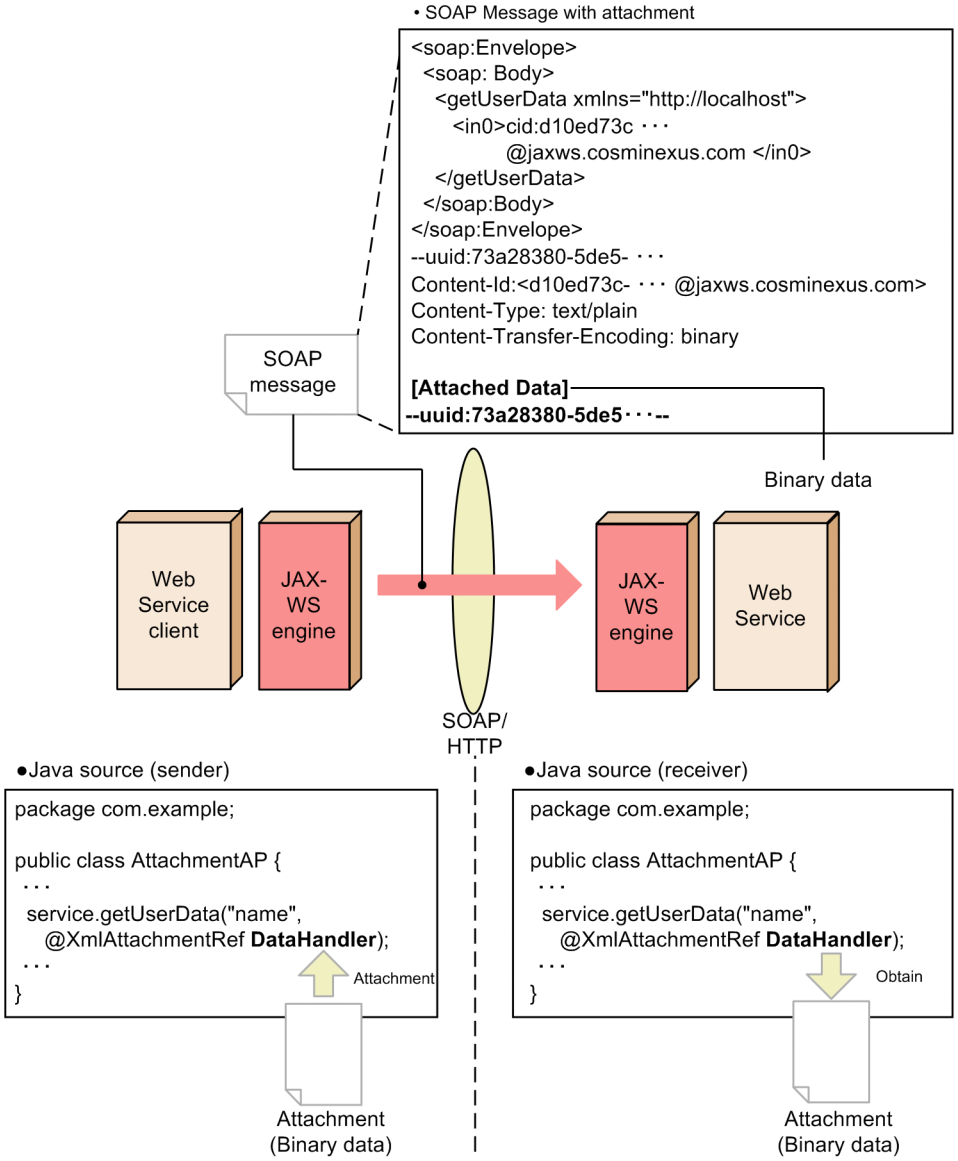
This chapter provides an overview of the attachment functionality and describes the mapping rules for using the attachment functionality.

28.1 What is the attachment functionality (wsi:swaRef format)

The attachment functionality is a functionality that sends and receives SOAP Messages with attachments wherein text and image (binary data) files are attached to SOAP Messages.

This section provides an overview of the attachment functionality based on the following example:

Figure 28－1: Sending and receiving binary data using the attachment functionality



To attach a file to the SOAP Message, use `javax.activation.DataHandler` class in an argument of Java method that invokes the Web Service. A Java source with a file attached is marshalled by the SOAP Messages using the sending JAX-WS engine and is un-marshalled by the Java source using the receiving JAX-WS engine. The receiving JAX-WS engine obtains the attachment from the argument of the un-marshalled Java source method.

Note that this functionality complies with WS-I Attachments Profile - Version 1.0 specifications; therefore, WSDL of `wsi:swaRef` type is used. Also, since the SOAP Messages with Attachments protocol is used; the SOAP Messages with attachments are encoded using the MIME Multipart/ Related structure.

To send and receive SOAP messages with attachments, using the APIs of the SAAJ 1.3 specifications, see the section *22.1 Support range of the SAAJ 1.3 specifications*.

28.2 Java interface of attachments (wsi:swaRef format)

This subsection describes the Java types for which attachments are specified in the Java interface.

(1) Java types that can be used in attachments

The following table describes the usability of Java type of attachments in the Java interface:

You need to add `javax.xml.bind.annotation.XmlAttachmentRef` annotation in the Java types that can be used in attachments. If you add this annotation in the types that cannot be used in attachments, the operations might not function properly.

Table 28-1: Usability of Java types as attachments

No.	Java type	Usability
1	`javax.activation.DataHandler`	Y
2	`javax.xml.ws.Holder<DataHandler>`	Y
3	Array type of `javax.activation.DataHandler`	R
4	Array type of `javax.xml.ws.Holder<DataHandler>`	N
5	Data type that inherits `javax.activation.DataHandler`	N

Legend:

Y: Can be used as attachment.

R: Only one-dimensional array can be used as attachment. If a multi-dimensional array is used, the operations might not function properly.

N: Cannot be used as attachment.

(2) Locations in which attachments can be specified

You can specify an attachment in the fields of the method argument, method return value, and user-defined type of Java interface. You cannot specify an attachment in a user-defined exception.

The following table describes the location for specifying an attachment in the Java interface and the whether the Java type can be specified:

Table 28-2: Location of specification and specifiability of Java types of attachments

No.	Location of specification in the Java interface	Java type of attachment	Can be specified or not
1	Method arguments	`javax.activation.DataHandler`	Y
2		`javax.xml.ws.Holder<DataHandler>`	Y
3		Array type of `javax.activation.DataHandler`	R
4	Method return values	`javax.activation.DataHandler`	Y
5		`javax.xml.ws.Holder<DataHandler>`	N
6		Array type of `javax.activation.DataHandler`	R
7	Fields of the user-defined types	`javax.activation.DataHandler`	Y
8		`javax.xml.ws.Holder<DataHandler>`	N
9		Array type of `javax.activation.DataHandler`	R
10	Fields of the user-defined exceptions	`javax.activation.DataHandler`	N
11		`javax.xml.ws.Holder<DataHandler>`	N

No.	Location of specification in the Java interface	Java type of attachment	Can be specified or not
12	Fields of the user-defined exceptions	Array type of `javax.activation.DataHandler`	N

Legend:

Y: Can be specified.

R: Can be specified only when one-dimensional array is used. If specified using a multi-dimensional array, the operations might not function properly.

N: Cannot be specified.

(3) Attachment that can be specified in javax.activation.DataHandler type

The `javax.activation.DataHandler` type is a type of JavaBeans Activation Framework (JAF), so you can specify any MIME type attachment. For details on the extensions for the attachments and the MIME type mapping set up by default, see *28.4.2(3) Mapping between the attachment extension and MIME types*.

28.3 WSDL for attachments (wsi:swaRef format)

When you use the SOAP Messages with attachments, you can define an attachment on WSDL. This section describes the WSDL coding and the mapping between WSDL and Java types when attachments are used.

28.3.1 WSDL coding when attachments are used (wsi:swaRef format)

To handle attachments in WSDL, code using the `wsi:swaRef` type. The `wsi:swaRef` type is defined in the WS-I Attachments Profile - Version 1.0 as the type for handling attachments in WSDL

The following is an example of WSDL coded with the `wsi:swaRef` type on the basis of the type defined in WS-I Attachments Profile - Version 1.0:

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://localhost"
 xmlns:wsi="http://ws-i.org/profiles/basic/1.1/xsd" ...>
 <wsdl:types>
 <schema elementFormDefault="qualified" targetNamespace="http://localhost"
 xmlns="http://www.w3.org/2001/XMLSchema">
 <import namespace="http://ws-i.org/profiles/basic/1.1/xsd"/>
 <element name="getUserData">
 <complexType>
 <sequence>
 <element name="in0" type="xsd:string"/>
 <element name="in1" type="wsi:swaRef"/>
 </sequence>
 </complexType>
 </element>
 ...
 </schema>
 </wsdl:types>
 <wsdl:message name="getUserDataRequest">
 <wsdl:part element="intf:getUserData" name="parameters"/>
 </wsdl:message>
 ...
 <wsdl:portType name="UserInfo">
 <wsdl:operation name="getUserData">
 <wsdl:input message="intf:getUserDataRequest" name="getUserDataRequest"/>
 ...
 </wsdl:operation>
 </wsdl:portType>
 ...
</wsdl:definitions>
```

In this example, the `wsi:swaRef` type, indicating an attachment, is specified in the `getUserData` method argument (type of the `element` element in the schema definition).

Precautions for handling attachments in WSDL

- The `swaRef` type can be specified in the schema element type (`type` attribute of the `element` element). If the `swaRef` type is specified in the attribute type (`type` attribute of the `attribute` element), the operations might not function properly.
- In the XML Schema that specifies the `swaRef` type, you must import the name space `http://ws-i.org/profiles/basic/1.1/xsd` as shown in the WSDL example. If the name space is not imported, a Cosminexus XML Processor error is output.

  The following table describes the types and usability of the `schemaLocation` attribute of the `xsd:import` element that imports the namespace :

  Table 28-3: Type and usability of the schemaLocation attribute of the xsd:import element (when swaRef type is used)

No.	schemaLocation attributes	Usability
1	Not specified	Can be used. #1
2	`http://wsi.org/profiles/basic/1.1/swaref.xsd`	Can be used. #1
3	Other than No.1 and No.2	Cannot be used. #2

#1

The `cjwsimport` command references the schema of the name space `http://ws-i.org/profiles/basic/1.1/xsd` stored in the Cosminexus JAX-WS functionality.

#2

The `cjwsimport` command references the schema present in the location specified in the `schemaLocation` attribute. However, because the command references a schema for which contents cannot be guaranteed with the Cosminexus JAX-WS functionality, this is not supported.

- Attachments cannot be used in the user-defined exceptions, so the `swaRef` type cannot be specified in the schema element type (`type` attribute of the `element` element) referenced from the `wsdl:fault` element. If the `swaRef` type is specified in the schema element type referenced from the `wsdl:fault` element, the operations might not function properly.

28.3.2 Mapping of Java type of attachments and WSDL (wsi:swaRef format)

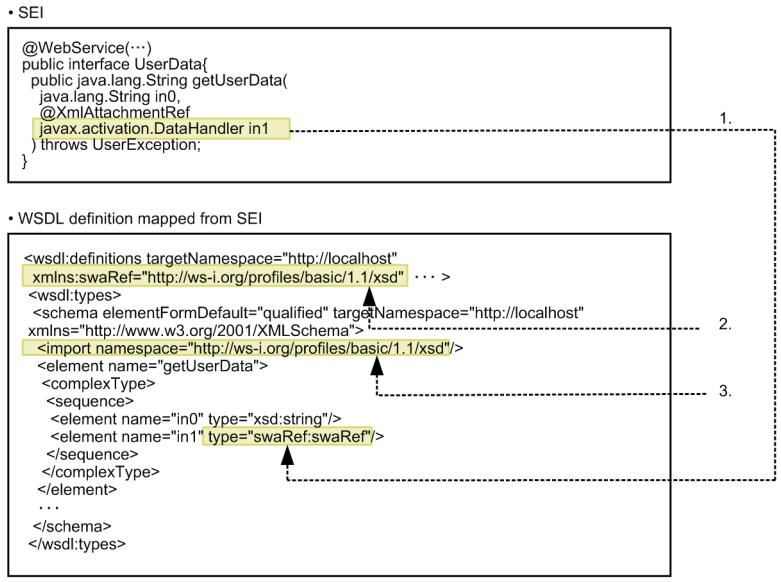
This subsection describes the rules for mapping the Java type of attachments to WSDL:

Rules for mapping the Java type of attachments to WSDL

1. The Java type of attachments is mapped to `swaRef` type.
2. The namespace of `swaRef` type is declared with the `wsdl:definitions` element.
3. The namespace of `swaRef` type is imported with the XML Schema that specifies the `swaRef` type.

The following figure shows an example of mapping of Java type of attachments and WSDL. The numbers in 'Rules for mapping the Java type of attachments to WSDL' above correspond to the numbers in the following figure:

Figure 28-2: Example of mapping Java type of attachments to WSDL



28.3.3 Mapping WSDL to the Java type of attachments (wsi:swaRef format)

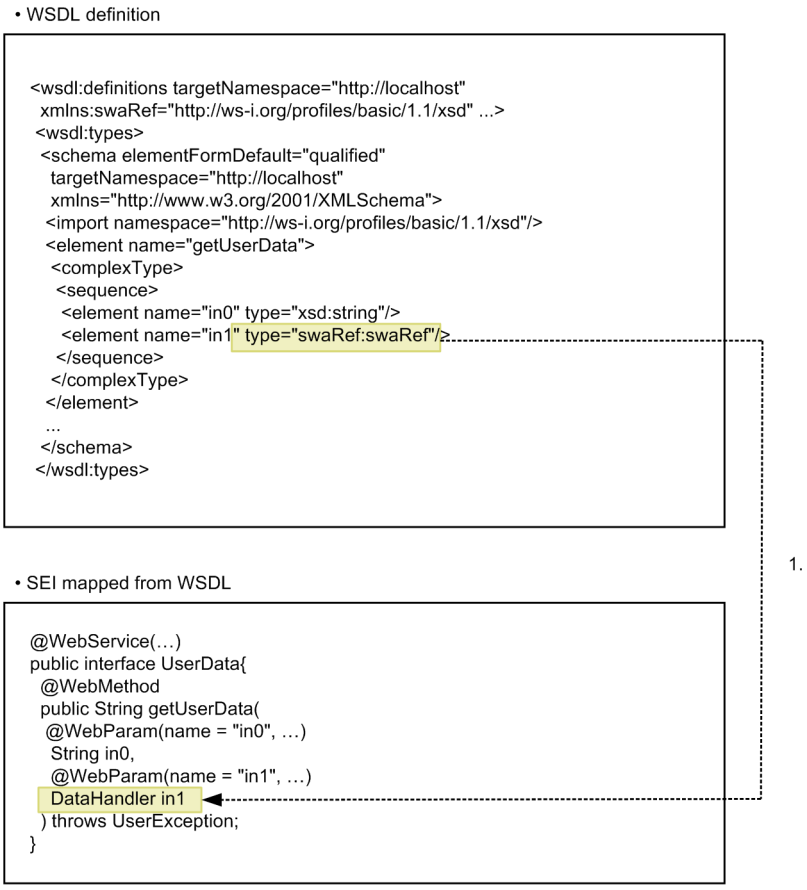
This subsection describes the rules for mapping WSDL to the Java type of attachments.

Rules for mapping WSDL to the Java type of attachments

1. The `swaRef` type of WSDL is mapped to the Java type of attachments.

The following figure shows example of mapping WSDL to the Java type of attachments. The numbers in 'Rules for mapping WSDL to the Java type of attachments' above correspond to the numbers in the following figure:

Figure 28-3: Example of mapping WSDL to Java type of attachments

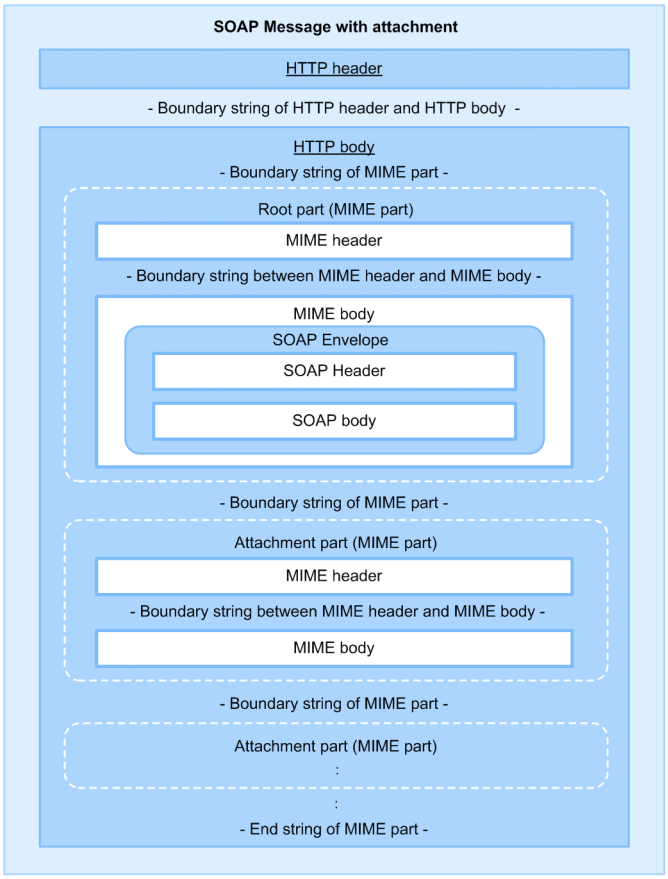


28.4 SOAP Messages with attachments (wsi:swaRef format)

The SOAP Messages with attachments use the SOAP Messages with Attachments protocol and are encoded using the MIME Multipart/ Related structure.

The following figure shows the structure of the SOAP Messages with attachments:

Figure 28-4: Structure of SOAP Messages with attachments



The following table describes each part of the SOAP Messages with attachments:

Table 28-4: Description about each part of SOAP Messages with attachments

Name of each part	Description
HTTP header	This is the header information dependent on the HTTP protocol.
Boundary string of the HTTP header and HTTP body	This is a string indicating the boundary of the HTTP header and HTTP body.
HTTP body	This describes the message to be sent. Consists of the root part and attachment part.
Boundary string of the MIME part	This is a string indicating the boundary of each MIME part.
Root part	This part describes the main message. Consists of the MIME header and MIME body. One root part is necessarily defined.
MIME header	This is the header information of the root part.

Name of each part	Description
Boundary string between the MIME header and MIME body	This is a string indicating the boundary between the MIME header and MIME body of the root part.
MIME body	This describes the main message.
SOAP Envelope	This describes the SOAP Envelope.
SOAP Header	This describes the header information of the SOAP Message.
SOAP Body	This describes the SOAP Message text (XML).
Boundary string of the MIME part	This is a string indicating the boundary of each MIME part.
Attachment part	This part describes the attachment contents. Consists of the MIME header and MIME body. Define 0 or more attachment parts.
MIME header	This is the header information of the attachment part.
Boundary string between the MIME header and MIME body	This is a string indicating the boundary between MIME header and MIME body of the attachment part.
MIME body	This describes the attachment contents (binary data).
End string of the MIME part	This is a string indicating the end of the MIME part.

28.4.1 Mapping an attachment to a SOAP Message (wsi:swaRef format)

This subsection describes the settings when you map an attachment to a SOAP message. For details about the mapping rules, check this subsection together with *28.4.2 Precautions on mapping an attachment to a SOAP message (wsi:swaRef format)*.

(1) HTTP header

The following table describes the values set for the fields and parameters of the HTTP header when attachments are used:

Table 28-5: Values set in the fields and parameters of HTTP header

No.	Fields name	Parameter name	Settings
1	`Content-Type`	`--`	`multipart/related` is set.
2		`type`	SOAP 1.1 specifications `text/xml` is set. SOAP 1.2 specifications `application/soap+xml` is specified.
3		`boundary`	The boundary string of the MIME part is set.

Legend:

--: Indicates that the name is directly set in Content-Type.

When 1 or more non-root MIME parts are enumerated (when SOAP Messages with attachments are sent and received), multipart/related is set in the Content-Type. If there is no MIME part, `text/xml` is set for the SOAP 1.1 specifications and `application/soap+xml` is set for the SOAP 1.2 specifications.

(2) HTTP body

The HTTP body consists of the root part, attachment part, and boundary string of each part. The following points describe the contents generated in each part and boundary string and the settings when you use attachments:

(a) MIME header of the root part

The following table lists the values set up in the Content-Type field of the root part when you use attachments:

Table 28-6: Values set up in the root part field

No.	Field name	Settings
1	`Content-Type`	SOAP 1.1 specifications `text/xml` is set up. SOAP 1.2 specifications `application/soap+xml` is set up.
2	`Content-Id`	`Globally unique value"+"@"+"jaxws.cosminexus.com` is set up.

(b) MIME body of the root part

When you use attachments, SOAP Envelope is saved in the MIME body of the root part as it is. The CID URL scheme is used as the method for referencing the attachments from the SOAP Body in the SOAP Envelope.

The following is the CID URL scheme type:

```
"cid:"  +  Content-Id-of-the-attachment-part
```

(c) Boundary string of the MIME header and MIME body of the root part

"CRLF" is set as the boundary string between the MIME header and MIME body of the root part.

(d) MIME header of the attachment part

The following table describes the values set in the MIME header of the attachment part when attachments are used:

Table 28-7: Values set in the MIME header of the attachment part

No.	Fields name	Parameter name	Settings
1	`Content-Type`	`--`	MIME type#1 is set according to the attachment type.
2		`charset`	The character code#2 is specified when the `DataHandler` object that is generated, is set.
3	`Content-Transfer-Encoding`	None	"binary" is set.
4	`Content-Id`	None	"Globally unique value" + "@" + "jaxws.cosminexus.com" is set.

Legend:

--: Indicates that the name is directly set in Content-Type.

#1

The settings for the `Content-Type` field differ as follows depending on the method of generating the `DataHandler` object:

- When the object is generated using `DataHandler(DataSource)` constructor

  For `FileDataSource`, the MIME type determined by JAF from the extension of the input attachment is set as the value of `Content-Type` field. For details about the mapping between the attachment extension and the MIME types, see *28.4.2(3) Mapping between the attachment extension and MIME types*.

- When the object is generated using `DataHandler(Object, String)` constructor

  When the `DataHandler` object is generated, the contents specified in the second argument of the constructor (MIME type) are set as the value of the `Content-Type` field as it is.

#2

For example, when the following `DataHandler` object is generated, the `charset` parameter value 'UTF-8' specified in the second argument is set in `charset` of `Content-Type`:

```
DataHandler dhandler = new DataHandler ("abcde", "text/plain; charset=Shift_JIS");
```

The `javax.activation.DataHandler` type that is a Java type of attachment is a type of JavaBeans Activation Framework (JAF); therefore, you can specify any MIME type attachment. For details about the correspondence between the attachment extension and the MIME types set by default when `FileDataSource` is used to generate the `DataHandler` object, see *28.4.2(3) Mapping between the attachment extension and MIME types*.

(e) MIME body of the attachment part

The binary data indicating the attachment contents is stored in the MIME body of attachment part.

(f) Boundary string of the MIME header and MIME body of the attachment part

"CRLF" is set as the boundary string of the MIME header and MIME body of the attachment part, as for the root part.

(g) Boundary string between MIME parts

The following string is set as the boundary string between the root part and attachment part and between the attachment parts:

```
"CRLF" + "--" + "Boundary-parameter-value-of-HTTP-header"
```

(h) End string of the MIME part

The following string is set as the end string at the end of MIME part:

```
"CRLF" + "--" + "<Boundary-parameter-value-of-HTTP-header>" + "--"
```

(3) Boundary string of the HTTP header and HTTP body

"CRLF" is set as the boundary string between the HTTP header and HTTP body.

28.4.2 Precautions on mapping from an attachment to a SOAP Message (wsi:swaRef format)

This subsection describes the precautions for mapping the SOAP Message from the attachment.

(1) Coding order of MIME part

A MIME part consists of 1 root part and 0 or more attachment parts.

The root part is coded at the beginning of the MIME part. After the root part, the attachment part is coded.

The arguments and return values specified in the Java interface are mapped in the attachment part. The following table describes the relationship between the contents specified in the Java interface and the coding order in the attachment part:

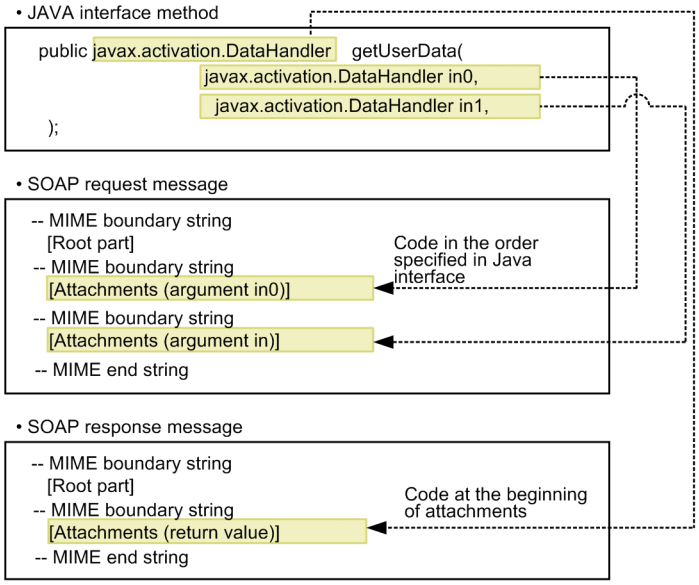
Table 28-8: Contents specified in Java interface and coding order of the attachment part

No.	Location for specifying in Java interface	Coding order of the attachment part
1	Method argument	Coded in the order specified in the method argument.
2	Method return value	Coded at the beginning of the attachment part.
3	Array type	Coded in the order of the elements in the array.

No.	Location for specifying in Java interface	Coding order of the attachment part
4	User-defined type	Coded in the order of specification of the user-defined type. When specifying the Java type of attachments using the user-defined types specified in the user-defined types, the attachment parts are coded in the depth first order.

The following is an example of mapping the Java interface and attachment parts:

Figure 28-5: Example of mapping the Java interface and attachment parts



(2) Mapping the root part to the attachment

The `Content-Id` of the corresponding attachment part is coded in the CID URL scheme that is coded in the SOAP Body of the root part. The corresponding attachment part is referenced from the root part using the `Content-Id`.

The following are some of the SOAP Messages with attachments. The part in bold is the CID URL scheme and the `Content-Id` of the corresponding attachment part.

```
--uuid:73a28380-5de5-45e8-af15-c879e65d62a0
Content-Type:text/xml

<?xml version="1.0" ?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
 <S:Body>
 <getUserData xmlns="http://localhost">
 <in0>cid:d10ed73c-e9b7-418e-8201-ba920cccb214@jaxws.cosminexus.com</in0>
 </getUserData>
 </S:Body>
</S:Envelope>

--uuid:73a28380-5de5-45e8-af15-c879e65d62a0
Content-Id:<d10ed73c-e9b7-418e-8201-ba920cccb214@jaxws.cosminexus.com>
Content-Type:text/plain
Content-Transfer-Encoding:binary

Attached-data
--uuid:73a28380-5de5-45e8-af15-c879e65d62a0--
```

(3) Mapping between the attachment extension and MIME types

The following table describes the mapping between the attachment extensions and the MIME types set by default:

Table 28-9: Correspondence table of attachment extensions and MIME types

No.	Attachment extensions	Set up MIME types
1	html, htm	text/html
2	txt, text	text/plain
3	gif, GIF	image/gif
4	ief	image/ief
5	jpeg, jpg, jpe, JPG	image/jpeg
6	tiff, tif	image/tiff
7	xwd	image/x-xwindowdump
8	ai, eps, ps	application/postscript
9	rtf	application/rtf
10	tex	application/x-tex
11	texinfo, texi	application/x-texinfo
12	t, tr, roff	application/x-troff
13	au	audio/basic
14	midi, mid	audio/midi
15	aifc	audio/x-aifc
16	aif, aiff	audio/x-aiff
17	wav	audio/x-wav
18	mpeg, mpg, mpe	video/mpeg
19	qt, mov	video/quicktime
20	avi	video/x-msvideo

If extensions that do not exist in this table are specified, 'application/octet-stream' is set in the MIME type.

(4) Mapping the attachment data size and the attachment parts

When the attachment data size specified in the Java interface is null and when data of 0 or more bytes exists, the method of mapping to the attachment part differs. The following points describe the mapping in each of the cases:

(a) When the attachment data size is null

When the attachment data size is null, the attachment data size is not mapped to the attachment part. Only the SOAP Envelope is coded without generating the MIME part. Also, an empty element is specified in the element of the relevant argument of the SOAP Body.

The following is an example of mapping when the attachment data size is null:

- Web Service invoking program

```
...
UserInfoService service = new UserInfoService();
UserInfo impl = service.getUserInfo();

result = impl.getUserData( null );
...
```

- SOAP Message

```
<?xml version="1.0" ?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
 <S:Body>
 <getUserData xmlns="http://localhost">
 </getUserData>
 </S:Body>
</S:Envelope>
```

(b) When the attachment data size is 0 or more bytes

When the attachment data size is 0 or more bytes, the attachment data size is mapped to the attachment part. In this case, the root part and attachment part together form the multi part. When multiple attachments are specified in the Java interface, multiple attachment parts are coded. However, for 0 bytes, the MIME body inside the attachment part is empty.

28.4.3 Mapping the SOAP message to the attachment (wsi:swaRef format)

JAX-WS engine guarantees the mapping of SOAP message with an attachment to the attachment data only when the SOAP message with the received attachments complies with the WS-I Attachments Profile - Version 1.0 specification. If any other SOAP message is received, the operation is not guaranteed.

28.5 Generating and obtaining the Java instance of the attachment (wsi:swaRef format)

To handle an attachment with a program, you must use the `javax.activation` package of JAF. This section describes the method of generating the attachment instance and the method of obtaining data using the `javax.activation` package.

For details about JAF specifications, see API specifications for JAF.

28.5.1 Method of generating the attachment instance (wsi:swaRef format)

The `javax.activation.DataHandler` object is generated as an attachment object. The following table lists the constructors of the `javax.activation.DataHandler` class:

Table 28-10: Constructors of the javax.activation.DataHandler class

No.	Use case	Constructor	Description of the argument
1	Attaching a file	`DataHandler(javax.activation.DataSource ds)`	[First argument] This is the `javax.activation.DataSource` object. The object of the `javax.activation.FileDataSource` class can be specified.
2	Attaching an object that is on memory	`DataHandler(java.lang.Object obj, java.lang.String mimeType)`	[First argument] This is the Java object. [Second argument] This is the MIME type of the object. For details about the MIME type that can be specified, see *28.4.2(3) Mapping between the attachment extension and MIME types*.

Depending on the objects sent as attachments, the method of generating the `javax.activation.DataHandler` object is different. The following points describe the generation method for each object to be sent:

(1) Method of sending an existing file as an attachment

To attach and send an existing file:

1. Generate the javax.activation.FileDataSource object.
   In the argument, specify the file path of attachment to be sent and generate the `javax.activation.FileDataSource` object.

```
javax.activation.FileDataSource fdSource =
 new javax.activation.FileDataSource("/tmp/sample.jpg");
```

2. Generate the javax.activation.DataHandler object.
   Specify the `javax.activation.FileDataSource` object in the argument and generate `javax.activation.DataHandler` object.

```
javax.activation.DataHandler dhandler =
 new javax.activation.DataHandler(fdSource);
```

(2) Method of sending a Java object as an attachment

To send a Java object as an attachment:

1. Generate the Java object.
   Here, you generate the `java.awt.Image` object to be sent as an attachment:

```
java.awt.Image attachments =
 Toolkit.getDefaultToolkit().createImage("sample.jpg");
```

2. Generate the javax.activation.DataHandler object.
   Specify the Java object and MIME type corresponding to the Java object in the argument and generate `javax.activation.DataHandler` object.

```
javax.activation.DataHandler dhandler = new
 javax.activation.DataHandler(attachments, "image/jpeg");
```

(3) Method of sending the java.lang.String object as an attachment

To send the java.lang.String object as an attachment:

1. Generate the java.lang.String object.
   Generate the `java.lang.String` object of the string to be sent as an attachment.

```
java.lang.String attachments = new java.lang.String("abcde");
```

2. Generate the javax.activation.DataHandler object.
   Specify the `java.lang.String` object and MIME type in the argument and generate the `javax.activation.DataHandler` object. The `java.lang.String` object is encoded using the character code specified in `charset` parameter.

```
javax.activation.DataHandler dhandler = new
javax.activation.DataHandler(attachments, "text/plain; charset=UTF-8");
```

(4) Precautions on creating the javax.activation.DataHandler object

If the object that you want to send as an attachment in the *wsi:swaRef* format is a text file, an XML file, or a `java.lang.String` object (string), you can specify the character code of the string included in the object by using the `DataHandler (Object, String)` constructor of the `javax.activation.DataHandler` class.

If you do not specify the character code when creating the `javax.activation.DataHandler` object, the object to be sent is encoded by the default character code (`US-ASCII`). If the object contains characters other than the default character code (`US-ASCII`), the attachment that is sent is treated as invalid. Therefore, specify an appropriate character code especially when you want to include Japanese characters.

If the object to be sent is an XML file, the character code specified in the XML declaration is not used. The XML file is encoded using the default character code (`US-ASCII`), or the character code specified when creating the `javax.activation.DataHandler` object, and then sent.

To specify a character code, assign the `charset` parameter to the *MIME* type of the object to be specified in the second argument of the `DataHandler(Object, String)` constructor. The following is the coding format of the second argument of the `DataHandler(Object, String)` constructor.

```
Mime type of the object to be sent+";"+"charset"+"="+character code#
```

#

The character code is not case sensitive. Also, you cannot specify the character code that is not supported by the JDK and a blank character.

28.5.2 Method of obtaining the attachment data (wsi:swaRef format)

This subsection describes about how to obtain the attachment data. The "dhandler" indicated in each example shows the received javax.activation.DataHandler object.

(1) Method of obtaining an attachment as a java.io.InputStream object

To obtain the received attachment as the java.io.InputStream object, obtain the java.io.InputStream object using the getInputStream method from the received javax.activation.DataHandler object.

```
java.io.InputStream stream = dhandler.getInputStream();
```

(2) Method of obtaining an attachment as a javax.activation.DataSource object

To obtain the received attachment as the javax.activation.DataSource object, obtain the javax.activation.DataSource object linked using the getDataSource method from the received javax.activation.DataHandler object.

```
javax.activation.DataSource datasource = dhandler.getDataSource();
```

(3) Method of obtaining an attachment as a Java object

The procedure for obtaining the received attachment as a Java object is as follows. An example of obtaining the java.awt.Image object is described here.

1. Obtain the attachment data as an object.
   Obtain the object using the getContent method from the received javax.activation.DataHandler object.

   ```
   java.lang.Object content = dhandler.getContent();
   ```

2. Obtain the MIME type of the attachment.
   Execute the getContentType method for the received javax.activation.DataHandler object. By executing the getContentType method, you can obtain the MIME type of the attachment.

   ```
   java.lang.String mimetype = dhandler.getContentType();
   Obtained-mimetype-contents image/jpeg
   ```

3. Cast the object to the appropriate type.
   Cast the object to the appropriate type according to the MIME type of the attachment.

   ```
   java.awt.Image attachment = (java.awt.Image) content;
   ```

(4) Method of obtaining an attachment as a java.lang.String object

To obtain the received attachment as a Java object:

1. Obtain the attachment data as an object.
   Obtain the object using the getContent method from the received javax.activation.DataHandler object.

   ```
   java.lang.Object content = dhandler.getContent();
   ```

2. Obtain the MIME type of the attachment.
   Execute the getContentType method for the received javax.activation.DataHandler object. By executing the getContentType method, you can obtain the MIME type and character code of the attachment.

   ```
   java.lang.String mimetype = dhandler.getContentType();
   Obtained-mimetype-contents text/plain; charset=UTF-8
   ```

3. Cast the object to the appropriate type.
   Cast the object to the appropriate type according to the MIME type of the attachment.

```
java.lang.String attachment = (java.lang.String) content;
```

(5) Notes when acquiring a javax.activation.DataHandler object

Receiving a SOAP message of the MIME Multipart or related structure that contains an attachment of the `wsi:swaRef` format enables you to handle the SOAP messages as an attachment of the streamed `wsi:swaRef` format. Because the receiving process requires all the data to be imported from the input stream of the `javax.activation.DataHandler` object to completely receive the attachment of the `wsi: swaRef` format, the sender is in stand-by status until all the data is imported from the input stream.To resolve this condition, you must either import all the data from the `java.io.InputStream` object of the `javax.activation.DataHandler` object or write the data from the input stream to the output stream by using the `writeTo(java.io.OutputStream)` method of the `javax.activation.DataHandler` class.

29 Examples of the Development Starting from SEI (When using Attachments of the wsi:swaRef format)

This chapter describes the examples for the development of Web Services starting from SEI, using attachments.

29.1 Configuration examples of development (Starting from SEI and attachments of wsi:swaRef format)

In the development examples described in this chapter, develop a Web Service, starting from SEI. The developed Web Service uses attachments.

This section provides an overview and describes the information used for the Web Service whose development will be illustrated.

Overview of development example

A new Web Service will be developed that will manage user information such as employee identification number, photograph, employee name, and affiliation and return the processing result for the input from the Web Service client.

The following tables describe the request information from the Web Service client and the response information from the server:

Table 29-1: Request information from the Web Service client

Information name	Java data type
Employee identification number	`java.lang.String`
Photograph	`javax.activation.DataHandler`

Table 29-2: Response information from the server

Information name	Java data type
Confirm registration message	`java.lang.String`
Name	`java.lang.String`
Affiliation	`java.lang.String`

The response information from the server is stored using the UserData class of the user-defined type class.

The following table describes the configuration of the current directory used in the Web Service development.

Table 29-3: Configuration of the current directory (Starting from SEI and attachments)

Directory	Description
`c:\temp\jaxws\works\attachments`	This is the current directory.
`server\`	Used for Web Service development.
`META-INF\`	Corresponds to the META-INF directory of the EAR file.
`application.xml`	Created in *29.3.4 Creating application.xml*.
`src\`	Stores the source file (`*.java`) for the Web Service. Used in *29.3.1 Creating Web Services Implementation Class* and *29.3.2 Compiling Web Services Implementation Class*.
`WEB-INF\`	Corresponds to the WEB-INF directory of the WAR file.
`web.xml`	Created in *29.3.3 Creating web.xml*.
`classes\`	Stores the compiled class file (`*.class`). Used in *29.3.2 Compiling Web Services Implementation Class*.
`attachments_dynamic_generate.ear`	Created in *29.3.5 Creating EAR files*.
`attachments_dynamic_generate.war`	Created in *29.3.5 Creating EAR files*.

Directory	Description
client\	Used for the development of the Web Service client.
src\	Stores the source file (`*.java`) of the Web Service client. Used in *29.5.1 Generating a service class* and *29.5.2 Creating Web Services Implementation Class* .
classes\	Stores the compiled class file (`*.class`). Used in *29.5.3 Compiling Implementation Class for the Web Services client*.
image.jpg	Used in the JPEG file to be used in the Web Service client.
usrconf.cfg	Created in *29.6.1 Creating an option definition file for Java applications*.
usrconf.properties	Created in *29.6.2 Creating the user property file for Java applications*.

Change the current directory path according to the environment to be developed.

Note that the directory and file names listed in the above table will be used in the description hereafter. The part formatted in *bold* in the command execution examples and in the Java source indicates the specified values and generated values that are used in this example. Read according to the environment you want to build.

Furthermore, in the development examples described in this chapter, the Web Service and Web Service client are developed in the same environment, but you can also develop them in separate environments. When you want to develop the Web Service and Web Service client in different environments, read the current directory path suitable to the respective environments.

29.2 Example of the development flow (Starting from SEI and attachments of wsi:swaRef)

In the development examples described in this chapter, the flow of development and execution is as follows.

Developing a Web Service

1. Creating Web Services Implementation Class (*29.3.1*)
2. Executing the `javac` command and compiling Web Services Implementation Class (*29.3.2*)
3. Creating `web.xml` (*29.3.3*)
4. Creating `application.xml` (*29.3.4*)
5. Creating EAR files (*29.3.5*)

Deploying and starting

1. Deploying EAR files (*29.4.1*)
2. Starting Web Services (*29.4.2*)

Developing a Web Service client

1. Executing the `cjwsimport` command and generating the service class (*29.5.1*)
2. Creating Implementation Class for the Web Services client (*29.5.2*)
3. Compiling Implementation Class for the Web Services client (*29.5.3*)

Executing a Web Service

1. Creating an option definition file for Java applications (*29.6.1*)
2. Creating the user property file for Java applications (*29.6.2*)
3. Executing the Web Services client (*29.6.3*)

29.3 Examples of Web Service development (Starting from SEI and attachments of wsi:swaref format)

This section describes the examples for the development of Web Services starting from SEI (using attachments).

29.3.1 Creating the Web Service Implementation Class

Create the Web Service Implementation Class that codes the processing of the Web Service. In this subsection, you calculate the contents of the received request message and create the Web Service Implementation Class that returns the response message.

The following is an example of the creation of a Web Service Implementation Class:

```
package com.sample;

import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;
import javax.xml.bind.annotation.XmlAttachmentRef;
import javax.activation.DataHandler;

@javax.jws.WebService(serviceName="UserInfoService",targetNamespace="http://
sample.com")
public class UserInfoImpl{

 public UserData getUserData( String in0,
 @XmlAttachmentRef javax.activation.DataHandler in1 )
 throws UserInfoException{

//Processing-for-registering-the-photograph-into-the-employee-information
 ...

 UserData userdata = new UserData();
 //Set-up-the-registered-employee-name-and-affiliation
 if ( in0.equals("1") )
 {
 userdata.setName("Hitachi Taro");
 userdata.setSection("The personnel section");
 } if (...) {
 ...
 } ...

 //Set-up-the-confirm-registration-message
 if ( in1 == null )
 {
 userdata.setMessage("Failure(no image).");
 } else {
 userdata.setMessage("Success.");
 }
 return userdata;
 }

}
```

The created `UserInfoImpl.java` is stored in `c:\temp\jaxws\works\attachments\server\src\com\sample\directory` with the UTF-8 format.

The user-defined type class `com.sample.UserData` used in `com.sample.UserInfoImpl` is also created. Normally, the creation of the user-defined type class is optional, but a user-defined type class will be created here.

The following is an example of the creation of a user-defined type class:

```
package com.sample;

public class UserData{

 private java.lang.String message;
 private java.lang.String name;
 private java.lang.String section;

 public UserData(){
 }
```

```
  public java.lang.String getMessage() {
  return this.message;
  }

  public void setMessage(java.lang.String message) {
  this.message = message;
  }

 public java.lang.String getName() {
  return this.name;
  }

  public void setName(java.lang.String name) {
  this.name = name;

 }

 public java.lang.String getSection() {
  return this.section;
  }

 public void setSection(java.lang.String section) {
  this.section = section;
  }
 }
```

The created `UserInfoImpl.java` is stored in `c:\temp\jaxws\works\attachments\server\src\com\sample\directory` with the UTF-8 format.

The exception class `com.sample.UserInfoException` thrown in `com.sample.UserInfoImpl` is also created. Normally, the creation of the exception class is optional, but an exception class will be created here.

The following is an example of the creation of an exception class:

```
package com.sample;

public class UserInfoException extends Exception {
 String detail;

 public UserInfoException (String message, String detail) {
 super (message);
 this.detail = detail;
 }

 public String getDetail () {
 return detail;
 }
}
```

The created `UserInfoException.java` is stored in `c:\temp\jaxws\works\attachments\server\src\com\sample\directory` with the UTF-8 format.

29.3.2 Compiling Web Services Implementation Class

Compile Web Services Implementation Class, by executing the `javac` command. For details on the `javac` command, see the *JDK documentation*.

The following example describes the execution of the `javac` command.

```
> cd c:\temp\jaxws\works\attachments\server\
> mkdir WEB-INF\classes\
> javac -cp
"%COSMINEXUS_HOME%\jaxws\lib\cjjaxws.jar;%COSMINEXUS_HOME%\CC\client\lib\j2ee-
javax.jar;%COSMINEXUS_HOME%\CC\client\lib\HiEJBClientStatic.jar;%COSMINEXUS_HOME%\jaxp
\lib\csmjaxb.jar;%COSMINEXUS_HOME%\jaxp\lib\csmjaxp.jar;%COSMINEXUS_HOME%\jaxp\lib
\csmstax.jar;%HNTRLIB2_HOME%\classes\hntrlib2j.jar;%HNTRLIB2
_HOME%\classes\hntrlibMj.jar" -d WEB-INF\classes\ -s src src\com\sample
\UserInfoImpl.java src\com\sample\UserData.java src\com\sample\UserInfoException.java
```

On successful execution of the `javac` command, the compiled classes are output to the following path:`c:\temp\jaxws\works\attachments\server\WEB-INF\classes\com\sample\Directory`You can execute

the cjwsgen command for the compiled Web Services Implementation Class to check errors in advance. For details on the cjwsgen command, see *10.23 (1)Using the cjwsgen command for checking errors*.

29.3.3 Creating web.xml

Create the web.xml that is required as a WAR file component.

The following is an example of the creation of the web.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/
javaee/web-app_3_0.xsd">
  <description>Sample web service &quot;fromjava&quot;</description>
  <display-name>Sample_web_service_fromjava</display-name>
  <listener>
    <listener-class>
      com.cosminexus.xml.ws.transport.http.servlet.WSServletContextListener
    </listener-class>
  </listener>
  <servlet>
    <description>Endpoint servlet for Cosminexus JAX-WS</description>
    <display-name>Endpoint_servlet_for_Cosminexus_JAX_WS</display-name>
    <servlet-name>CosminexusJaxwsServlet</servlet-name>
    <servlet-class>
      com.cosminexus.xml.ws.transport.http.servlet.WSServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>CosminexusJaxwsServlet</servlet-name>
    <url-pattern>/UserInfoService</url-pattern>
  </servlet-mapping>
  <session-config>
    <session-timeout>60</session-timeout>
  </session-config>
</web-app>
```

When creating web.xml of version 2.5, specify 2.5 in the version attribute of the web-app element and specify http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd as the second location information in the xsd:schemaLocation attribute.

The created web.xml is stored in c:\temp\jaxws\works\attachments\server\WEB-INF \directory with the UTF-8 format. For details about the web.xml settings, see *3.4 Creating web.xml*.

29.3.4 Creating application.xml

Create the application.xml that is required as an EAR file component.

The following is an example of the creation of application.xml. Note that no items are set in application.xml as the Web Service.

```
<?xml version="1.0" encoding="UTF-8"?>
<application version="6" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/
javaee/application_6.xsd">

  <description>Sample application &quot;fromjava&quot;</description>
  <display-name>Sample_application_fromjava</display-name>
  <module>
    <web>
      <web-uri>attachments_dynamic_generate.war</web-uri>
      <context-root>attachments_dynamic_generate</context-root>
    </web>
  </module>
</application>
```

When creating `web.xml` of version 5, specify `5` in the `version` attribute of the `application` element and specify `http://java.sun.com/xml/ns/javaee/application_5.xsd` as the second location information in the `xsd:schemaLocation` attribute.

The created `application.xml` is stored in `c:\temp\jaxws\works\attachments\server\META-INF\directory` with the UTF-8 format. For details about the precautions for creating `application.xml`, see *5.2.2 Notes on editing application.xml* in the *uCosminexus Application Server Application Development Guide*.

29.3.5 Creating EAR files

Use the `jar` command to create an EAR file containing the hitherto created files.

The following is an example of the creation of an EAR file:

```
> cd c:\temp\jaxws\works\attachments\server\
> jar cvf attachments_dynamic_generate.war .\WEB-INF
> jar cvf attachments_dynamic_generate.ear .\attachments_dynamic_generate.war .\META-
INF\application.xml
```

If the processing ends normally, `attachments_dynamic_generate.ear` is created in `c:\temp\jaxws\works\attachments\server\directory`.

For details about the `jar` command, see the JDK documentation.

29.4 Examples of deployment and startup (Starting from SEI and attachments of wsi:swaRef format)

This section describes the examples for the deployment and the startup starting from SEI, using attachments.

29.4.1 Deploying EAR files

Use the `cjimportapp` command to deploy the created EAR file on the J2EE server.

The following is an example of deployment:

```
> cd c:\temp\jaxws\works\attachments\server\
> "%COSMINEXUS_HOME%\CC\admin\bin\cjimportapp" jaxwsserver -nameserver
corbaname::testserver:900 -f attachments_dynamic_generate.ear
```

For details about the `cjimportapp` command, see *cjimportapp (Importing J2EE applications)* in the *uCosminexus Application Server Command Reference Guide*.

For how to deploy (import) J2EE applications by using the management portal, see *12.3.3 Importing J2EE applications* in the *uCosminexus Application Server Management Portal User Guide*.

29.4.2 Starting Web Services

Use the `cjstartapp` command to start the Web Service.

The following is an example of Web Service startup:

```
> cd c:\temp\jaxws\works\attachments\server\
> "%COSMINEXUS_HOME%\CC\admin\bin\cjstartapp" jaxwsserver -nameserver
corbaname::testserver:900 -name Sample_application_fromjava
```

For details about the `cjstartapp` command, see *cjstartapp (Starting J2EE applications)* in the *uCosminexus Application Server Command Reference Guide*.

For how to start J2EE applications by using the management portal, see *12.3.1 Starting J2EE applications* in the *uCosminexus Application Server Management Portal User Guide*.

29.5 Examples of Web Service client development (Starting from SEI and attachments of wsi:swaRef format)

This section describes the examples for the development of Web Service clients starting from SEI, using attachments.

29.5.1 Generating a service class

If you execute the `cjwsimport` command, the Java source, such as service class, required for Web Service client development is generated. For details about the `cjwsimport` command, see *14.1 cjwsimport command*.

The following is an example of the execution of the `cjwsimport` command.

```
> cd c:\temp\jaxws\works\attachments\client\
> mkdir src\
> mkdir classes\
> "%COSMINEXUS_HOME%\jaxws\bin\cjwsimport.bat" -s src -d classes http://webhost:8085/
attachments_dynamic_generate/UserInfoServicwsdl
```

If the execution ends normally, the Java source is generated in `c:\temp\jaxws\works\attachments\client\src\com\sample\directory`.

The following table describes a list of products.

Table 29-4: Products during service class generation (Starting from SEI and attachments)

File name	Description
`GetUserData.java`	This is the `JavaBean` class corresponding to type referenced by the wrapper element of the request message in the WSDL definition.
`GetUserDataResponse.java`	This is the `JavaBean` class corresponding to type referenced by the wrapper element of the response message in the WSDL definition.
`ObjectFactory.java`	This is the `ObjectFactory` class of the JAXB 2.2 specifications.
`package-info.java`	This is the `package-info.java` file.
`UserData.java`	This is the `JavaBean` class corresponding to `UserData`.
`UserInfoImpl.java`	This is SEI corresponding to 'Service' in the WSDL definition.
`UserInfoService.java`	This is the service class.
`UserInfoException.java`	This is the `JavaBean` class corresponding to `UserInfoException`.
`UserInfoException_Exception.java`	This is the wrapper exception class of the fault bean.

29.5.2 Creating the Web Service Implementation Class

You create the implementation class for the Web Service client that uses the Web Service.

The following is an example of creation of a Web Service client that invokes the Web Service once:

```
package com.sample.client;

import java.io.File;

import javax.activation.DataHandler;
import javax.activation.FileDataSource;

import com.sample.UserInfoImpl;
```

```
import com.sample.UserData;
import com.sample.UserInfoService;
import com.sample.UserInfoException_Exception;

public class TestClient {
 public static void main( String[] args ) {
 try {
 //Generating-the-DataHandler-object
 File imagefile = new File("image.jpg");
 FileDataSource fdSource = new FileDataSource(imagefile);
 DataHandler dhandler = new DataHandler(fdSource);

 UserInfoService service = new UserInfoService();
 UserInfoImpl_port = service.getUserInfoImplPort();

 UserData userdata = port.getUserData("1", dhandler );

System.out.print( "[RESULT] " + userdata.getMessage() );
            System.out.println( " Name:" + userdata.getName()
                                    + ", Section:" + userdata.getSection() );
 }
 catch( UserInfoException_Exception e ){
 e.printStackTrace();
        catch( Exception e ){
            e.printStackTrace();
        }
    }

}
```

The created `TestClient.java` is stored in `c:\temp\jaxws\works\attachments\client\src\com\sample\client\directory` with the UTF-8 format.

29.5.3 Compiling the implementation class for the Web Service client

Use the `javac` command to compile the created Web Service client.

The following is an example of compilation:

```
> cd c:\temp\jaxws\works\attachments\client\
> javac -encoding UTF-8 -cp "%COSMINEXUS_HOME%\jaxws\lib\cjjaxws.jar;%COSMINEXUS_HOME%
\CC\client\lib\j2ee-javax.jar;%COSMINEXUS_HOME%\jaxp\lib\csmjaxb.jar;.\classes" -d .
\classes src\com\sample\client\TestClient.java
```

If the processing ends normally, the `TestClient.class` is generated in `c:\temp\jaxws\works\attachments\client\classes\com\sample\client\directory`.

For details about the `javac` command, see the JDK documentation.

29.6 Examples of Web Service execution (Starting from SEI and attachments of wsi:swaRef format)

This section describes the example for the execution of Web Service clients starting from SEI, using attachments.

29.6.1 Creating the option definition file for Java applications

Create the option definition file for Java applications (`usrconf.cfg`) required for executing a Web Service.

The following is an example of creation of the option definition file for Java applications:

```
add.class.path=Cosminexus-installation-directory\jaxws\lib\cjjaxws.jar
add.class.path=.\classes
ejb.client.log.directory=logs
add.jvm.arg=-Dcosminexus.home=Cosminexus-installation-directory
```

For the *Cosminexus-installation-directory* part, use the absolute path to specify the path where Cosminexus is installed.

The created option definition file for Java applications is stored in `c:\temp\jaxws\works\attachments\client\directory`. For details about the option definition file for Java applications, see *14.2 usrconf.cfg (Option definition file for Java applications)* in the *uCosminexus Application Server Definition Reference Guide*.

29.6.2 Creating the user property file for Java applications

Create the user property file for Java applications required for executing a Web Service.

Since the settings are not specially changed here, create an empty file named `usrconf.properties` in `c:\temp\jaxws\works\attachments\client\directory`. For details about the user property file for Java applications, see *14.3 usrconf.properties (User property file for Java applications)* in the *uCosminexus Application Server Definition Reference Guide*.

29.6.3 Executing the Web Service client

Use the `cjclstartap` command to execute the Web Service client.

The following is an example of the execution of the Web Service client:

```
> cd c:\temp\jaxws\works\attachments\client\
> "%COSMINEXUS_HOME%\CC\client\bin\cjclstartap" com.sample.client.TestClient
```

If the execution ends normally, the result of Web Service client execution is displayed. The following is an example of the display of the execution result:

```
KDJE40053-I The cjclstartap command will now start. (directory for the user
definition file = c:\temp\jaxws\works\attachments\client, PID = 2636)
[RESULT] Success. Name:Hitachi Taro, Section:The personnel section
KDJE40054-I The cjclstartap command was stopped. (PID = 2636, exit status = 0)
```

The part in italics changes according to the execution timing and environment.

For details about the `cjclstartap` command, see *cjclstartap (Starting Java applications)* in the *uCosminexus Application Server Command reference Guide*.

30 Attachment functionality (MTOM/XOP)

Attachment functionality enables handling of not only the text data but also images and voice data through Web Services.

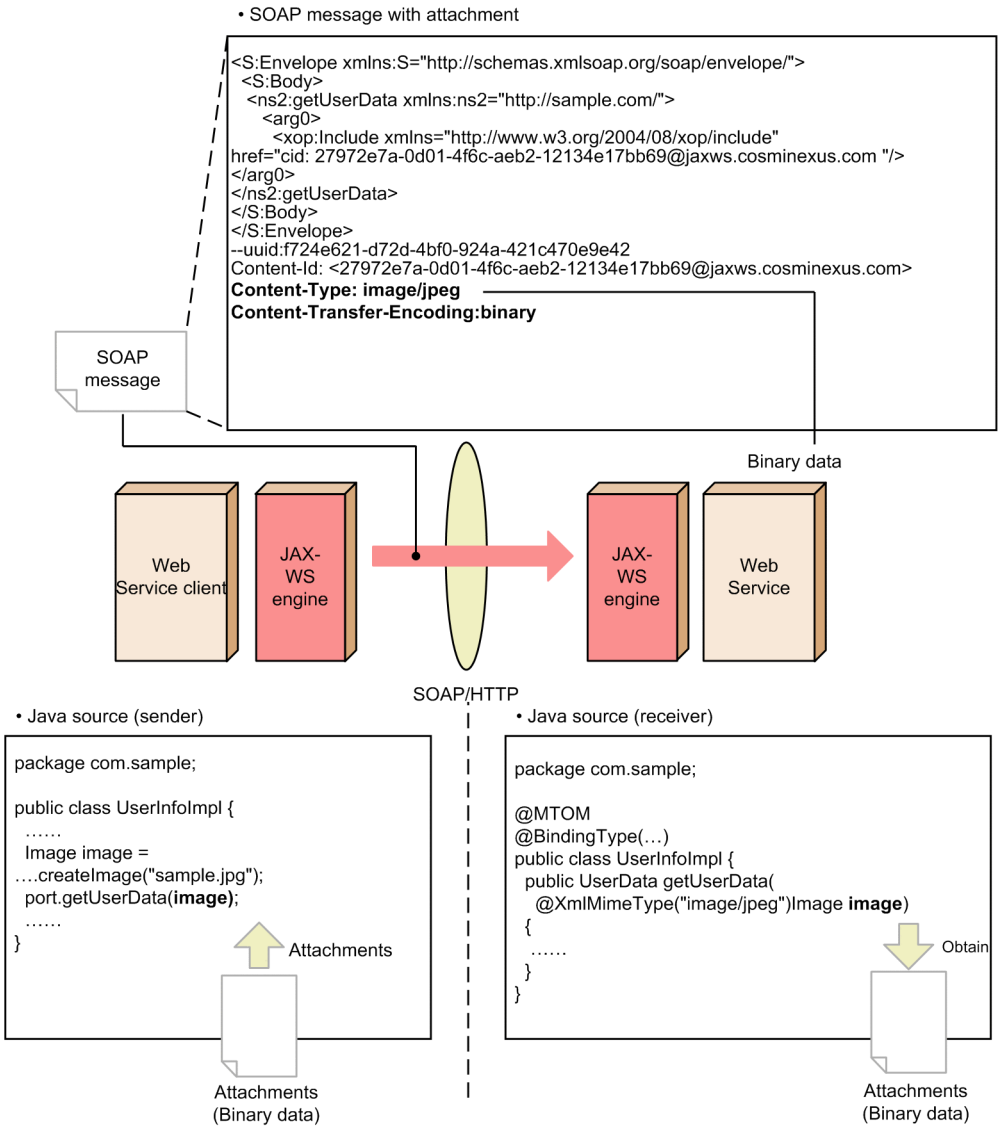
This document gives an overview of the attachment functionality and the mapping conventions involved when the attachment functionality is used.

30.1 Description of the attachment functionality (MTOM/XOP)

The attachment functionality of the MTOM/XOP specification format is sending and receiving the Base64 format data of the SOAP messages (XML data of the `xsd:base64 Binary` type) as the binary data by using SOAP Message Transmission Optimization Mechanism (MTOM) and XML-binary Optimized Packaging (XOP). The data sent and received through this functionality is attached to the SOAP messages of MIME Multipart/Related structure.

The following example gives an overview of the attachment functionality:

Figure 30-1: Sending and receiving binary data by using the attachment functionality



30.2 Java interface of an attachment (MTOM/XOP)

This section describes the Java type that is used for specifying an attachment in a Java interface.

(1) Target of attachments in MTOM/XOP specification format

All the Java types, which are mapped to the Base64 format data of the following items, are targeted for the attachments in the MTOM/XOP specification format:

- Method arguments
- Method return values
- User-defined type fields

If there are multiple Java types, which are mapped to the Base64 format data, you cannot specify only a particular Java type as the target of the attachments in the MTOM/XOP specification format.

(2) Annotations used in an attachment in MTOM/XOP specification format

Following annotations are used in an attachment in MTOM/XOP specification format:

- `javax.xml.ws.soap.MTOM`
- `javax.xml.bind.annotation.XmlMimeType`

(3) How to use the attachments in MTOM/XOP specification format

Annotate the `javax.xml.ws.soap.MTOM` annotation in the Web Service implementation class to use the MTOM/XOP specification format attachments in the Web Service. The following is an example of a Web Service implementation class that uses the attachments in MTOM/XOP specification format:

```
package com.sample;

@MTOM
@BindingType(...)
public class UserInfoImpl implements UserInfo {

    public UserData setUserData(Image image)
        throws UserDefinedException {
        ......
    }
}
```

You can also use the attachments in MTOM/XOP specification format by specifying the field value of the `javax.xml.ws.soap.SOAPBinding` interface in the `javax.xml.ws.BindingType` annotation instead of the `javax.xml.ws.soap.MTOM` annotation. The following table describes the relation between the field values of the `javax.xml.ws.soap.SOAPBinding` interface specified in the `javax.xml.ws.soap.MTOM` annotation and the `javax.xml.ws.BindingType` annotation.

Table 30-1: Relation between the MTOM annotation and SOAPBinding interface fields

No.	MTOM annotation		BindingType annotation		MTOM/XOP type attachment
	If annotated	enabled element value	SOAPBinding interface field value	Availability of the field value	
1	YES	true	`SOAP11HTTP_BINDING`	YES	Y
2			`SOAP12HTTP_BINDING`	YES	Y
3			`SOAP11HTTP_MTOM_BINDING`	YES	Y
4			`SOAP12HTTP_MTOM_BINDING`	YES	Y

No.	MTOM annotation		BindingType annotation		MTOM/XOP type attachment
	If annotated	enabled element value	SOAPBinding interface field value	Availability of the field value	
5	YES	false	SOAP11HTTP_BINDING	YES	N
6			SOAP12HTTP_BINDING	YES	N
7			SOAP11HTTP_MTOM_BINDING	NO#	--
8			SOAP12HTTP_MTOM_BINDING	NO#	--
9	NO	--	SOAP11HTTP_BINDING	YES	N
10			SOAP12HTTP_BINDING	YES	N
11			SOAP11HTTP_MTOM_BINDING	YES	Y
12			SOAP12HTTP_MTOM_BINDING	YES	Y

Legend:

Y: Enabled

N: Disabled

--: Not Applicable

#

`javax.xml.ws.WebServiceException` occurs when the Web Service starts.

The following is an example of a Web Service implementation class that uses the field values of the `javax.xml.ws.soap.SOAPBinding` interface instead of the `javax.xml.ws.soap.MTOM` annotation:

```
package com.sample;

@BindingType(SOAPBinding.SOAP11HTTP_MTOM_BINDING)
public class UserInfoImpl implements UserInfo {

    public UserData setUserData(Image image)
        throws UserDefinedException {
        ......
    }
}
```

(4) Document Bare style attachments in MTOM/XOP specification format

In a Document Bare style Java class, if you want to link Java type and MIME type by using an attachment in MTOM/XOP specification format, you must not use the Java type as a return value of the service method or as a parameter. To link Java type and MIME type, use Java type in the user-defined type field and annotate the `getter` method of that field with the `javax.xml.bind.annotation.XmlMimeType` annotation.

30.3 Attachment WSDL (MTOM/XOP)

WSDL does not have an element or attribute to know whether an attachment in the MTOM/XOP specification format is being used. Hence, you cannot know from WSDL whether the Web Service is using an attachment in the MTOM/XOP specification format.

30.3.1 non-wrapper style attachments in MTOM/XOP specification format (MTOM/XOP)

In a non-wrapper style WSDL, to specify MIME type in the `xsd:base64Binary` type by using an attachment in the MTOM/XOP specification format, do not use the `xsd:base64Binary` type for the `type` attribute of the `xsd:element` element that is referenced from the `wsdl:part` element. To specify MIME type as the `xsd:base64Binary` type, use compound type as the `type` attribute of the `xsd:element` element that is referenced from the `wsdl:part` element, and specify the `xsd:base64Binary` type in the `xsd:element` element of the said compound type and attach the `xmime:expectedContentTypes` attribute.

(1) Using the xop:Include element in WSDL

The `xop:Include` element appears in the SOAP messages of attachments in the MTOM/XOP specification format and links the route part and attachment part. Hence, you cannot use the `xop:Include` element in the WSDL schema declaration. Behavior is not guaranteed if the `xop:Include` element is used in the WSDL schema declaration.

30.4 Behavior of the JAX-WS engine

This section describes the behavior of the Web Service side and Web Service client side JAX-WS engine.

30.4.1 Behavior of the JAX-WS engine on a Web Service machine

The availability of attachments in MTOM/XOP specification format in the Web Service side JAX-WS engine depends upon the field value of the `SOAPBinding` interface specified in the `javax.xml.ws.soap.MTOM` annotation or `javax.xml.ws.BindingType` annotation specified in the Web Service implementation class. The following table describes the usage of attachments in MTOM/XOP specification format, and the behavior upon receiving the request messages:

Table 30-2: Availability of attachments in MTOM/XOP specification format and the behavior upon receiving the request messages

No.	Attachment in MTOM/XOP specification format	Data included in the received messages	Successfully received/ failed	Relation between the threshold and the attachment to be sent#	Data included in the response messages to be sent
1	Used	Binary data	Successful	Threshold ≤ attachment size	Binary data
2		Base64 type data	Successful	Threshold>attachment size	
3	Not used	Binary data	Successful	None	Base64 type data
4		Base64 type data	Successful	None	

#
Not determined with the threshold if the `javax.activation.DataHandler` is used. Always sent as binary data.

If the `javax.xml.ws.soap.MTOM` annotation is not specified in the Web Service implementation class, or if `SOAPBinding.SOAP11HTTP_MTOM_BINDING` and `SOAPBinding.SOAP12HTTP_MTOM_BINDING` are not specified in the field values of the `SOAPBinding` interface that is specified in the `javax.xml.ws.BindingType` annotation, attachments in MTOM/XOP specification format are not used. Irrespective of whether binary data is included, all request messages are received. Also, response messages with Base64 type data are sent.

(1) Changes due to the javax.xml.bind.annotation.XmlMimeType annotation

Depending upon whether the `javax.xml.bind.annotation.XmlMimeType` annotation, which links Java type and MIME type, is annotated or not annotated in SEI etc, the value of the Content-Type field in the attachment part of the messages sent through the attachments in MTOM/XOP specification format changes. See below for how the Content-Type value in the attachment part changes depending upon whether the `javax.xml.bind.annotation.XmlMimeType` annotation is used.

- `XmlMimeType` annotation is used

  The Content-Type field value in the attachment part will be the value of the `value` element in the `XmlMimeType` annotation.

- `XmlMimeType` annotation is not used

  The Content-Type field value in the attachment part will be the initial value of the corresponding Java type being used. The following table lists the initial values:

  Table 30-3: Content-Type initial values corresponding Java type

No.	Java type	Content-Type initial value
1	`java.awt.Image`	*image/png*

No.	Java type	Content-Type initial value
2	`javax.xml.transform.Source`	*application/xml*
3	`javax.activation.DataHandler`	*javax.activation.DataHandler object MIME type#*
4	`Array type of java.awt.Image`	*image/png*
5	`Array type of javax.activation.DataHandler`	*javax.activation.DataHandler object MIME type#*
6	`java.util.List<Image>`	*image/png*
7	`java.util.List<DataHandler>`	*javax.activation.DataHandler object MIME type#*
8	`javax.xml.ws.Holder<Image>`	*image/png*
9	`javax.xml.ws.Holder<Source>`	*application/xml*
10	`javax.xml.ws.Holder<DataHandler>`	*javax.activation.DataHandler object MIME type#*
11	`byte[]`	*application/octet-stream*

#

If you use `DataHandler`, the value set in the Content-Type field differs according to the method used for generating the `DataHandler` object.

- If generated by using the `DataHandler (DataSource)` constructor

  In `FiledataSource`, from the extension of the file data that is used as an input, the MIME type decided by JAF is set as the value of the Content-Type field.

- If generated by using the `DataHandler (Object, String)` constructor

  The contents (MIME type) specified in the 2nd argument of the constructor when the `DataHandler` object is generated is set as is as the Content-Type field value.

30.4.2 Behavior of the JAX-WS engine on a Web Service client machine

The availability of attachments in MTOM/XOP specification format in the Web Service client side JAX-WS engine depends upon the `MTOMFeature` class used to obtain an SEI. The following table describes the availability of attachments in MTOM/XOP specification format, and the behavior upon sending/receiving the messages.

Table 30–4: Availability of attachments in MTOM/XOP specification format and the behavior upon sending/receiving the messages

No.	Attachment in MTOM/XOP specification format	Relation between the threshold and the size of the attachment to be sent#	Data included in the request messages to be sent	Data included in the received response messages	Successfully received/ failed
1	Used	Threshold ≦ attachment size	Binary data	Binary data	Successful
2				Base64 type data	Successful
3		Threshold>attachment size	Base64 type data	Binary data	Successful
4				Base64 type data	Successful
5	Not used	--	Base64 type data	Binary data	Successful
6				Base64 type data	Successful

Legend:

--: Not Applicable

#
Not determined with the threshold if the `javax.activation.DataHandler` is used. Always sent as binary data.

The `MTOMFeature` class can be used concurrently with the other Feature classes. The following is one such example:

```
package com.sample;

......

public class TestClient {

    public static void main(String[] args) {
        try {
            File portrait = new File("portrait.png");
            if (!portrait.exists() ) {
                throw new RuntimeException("Cannot file \"portrait.png\".");
            }
            BufferedImage image = ImageIO.read(portrait);

            AddressingFeature addressingFeature = new AddressingFeature();
            MTOMFeature mtomFeature = new MTOMFeature();

            UserInfoService service = new UserInfoService();
            UserInfoImpl port = service.getUserInfoImplPort(addressingFeature,
mtomFeature);

            UserData userData = port.getUserData(image);
            ......
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

You can set the availability of attachments in MTOM/XOP specification format by using the `setMTOMEnabled` method of `javax.xml.ws.soap.SOAPBinding` instead of the `MTOMFeature` class. The following example describes how to set the availability of attachments in MTOM/XOP specification format by using the `setMTOMEnabled` method of `javax.xml.ws.soap.SOAPBinding`:

```
package com.sample;

......

public class TestClient {

    public static void main(String[] args) {
        try {
            File portrait = new File("portrait.png");
            if (!portrait.exists() ) {
                throw new RuntimeException("Cannot file \"portrait.png\".");
            }
            BufferedImage image = ImageIO.read(portrait);

            UserInfoService service = new UserInfoService();
            UserInfoImpl port = service.getUserInfoImplPort();

            BindingProvider bindingProvider = (BindingProvider)port;
            Binding         binding         = bindingProvider.getBinding();
            SOAPBinding     soapBinding     = (SOAPBinding)binding;
            soapBinding.setMTOMEnabled(true);

            UserData userData = port.getUserData(image);
            ......
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

When setting up the availability of attachments in MTOM/XOP specification format by using the `setMTOMEnabled` method of `javax.xml.ws.soap.SOAPBinding`, if the availability is already set up by the `setMTOMEnabled` method of the `javax.xml.ws.soap.SOAPBinding` and the `MTOMFeature` class beforehand, the values that are set up later by the `setMTOMEnabled` method become invalid and the availability cannot be set.

If you do not use the `MTOMFeature` class when obtaining an SEI, or if you do not set the availability of attachments in MTOM/XOP specification format in the `setMTOMEnabled` method of `javax.xml.ws.soap.SOAPBinding`, the attachments in MTOM/XOP specification format are not used. The request messages sent will include Base64 type data. Also, irrespective of whether the messages include binary data, all response messages are received.

(1) Changes due to the `xmime:expectedContentTypes` attribute

Depending upon the availability of the xmime:`expectedContentTypes` attribute in the WSDL schema element, the Content-Type field value present in the attachment part of the messages sent by the attachments in MTOM/XOP specification format changes. See below for how the Content-Type value in the attachment part changes depending upon whether the `xmime:expectedContentTypes` attribute is used.

- `xmime:expectedContentTypes` attribute exists

  The Content-Type field value in the attachment part will be the value that is specified in the `xmime:expectedContentTypes` attribute.

- `xmime:expectedContentTypes` attribute does not exist

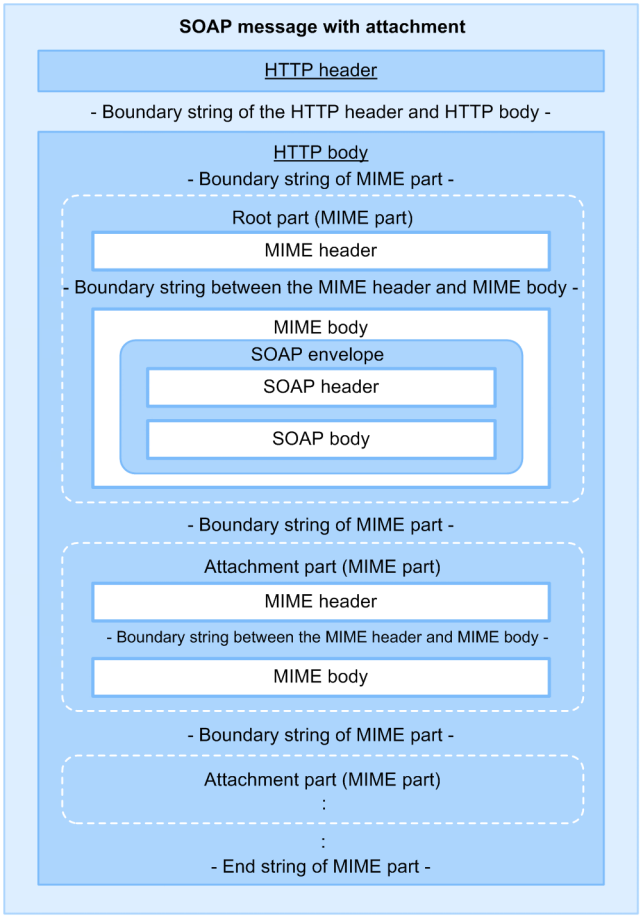
  The Content-Type field value in the attachment part will be `application/octet-stream`.

30.5 SOAP messages of the attachments in the MTOM/XOP specification format

When the attachments in MTOM/XOP specification format are used, the related SOAP messages have the MIME multipart/related structure that is generated by using the SOAP Messages with Attachments protocol. This section describes the SOAP messages with attachments in MTOM/XOP specification format.

The following figure shows the structure of the SOAP messages with attachments in MTOM/XOP specification format.

Figure 30-2: Structure of the SOAP messages with attachments in MTOM/XOP specification format



The following table describes each part of a SOAP message with attachments in MTOM/XOP specification format.

Table 30-5: Description of each part of a SOAP message with attachments in MTOM/XOP specification format

Name of each part		Explanation
HTTP header		This is the HTTP dependent header information.
Border string between the HTTP header and HTTP body		This is a string that shows the border between the HTTP header and HTTP body.
HTTP body		Messages to be sent are described here. This is formed of the route part and attachment part.
	MIME part border string	This is a string that shows the border between each MIME part.
	Route part	Message body is described in this part.

Name of each part	Explanation
Route part	This is formed of MIME header and MIME body. You must define at least one such part.
MIME header	This is the header information of the route part.
Border string between the MIME header and MIME body	This is a string that shows the border between the route part and MIME body.
MIME body	Message body is described in this part.
SOAP envelope	SOAP envelope is described here.
SOAP header	Header information of the SOAP messages is described here.
SOAP body	SOAP message body (XML) is described here.
MIME part border string	This is a string that shows the border between each MIME part.
Attachment part	Contents of the attachments in MTOM/XOP specification format are described in this part. This part comprises of the MIME header and MIME body and you can define 0 or more such parts.
MIME header	This is the header information of the attachment part.
Border string between the MIME header and MIME body	This is a string that shows the border between the MIME header and MIME body of the attachment part.
MIME body	The contents (binary data) of the attachments in MTOM/XOP specification format are described here.
MIME part end string	This is a string that shows the end of the MIME part.

30.5.1 Mapping the attachments to the SOAP messages (MTOM/XOP)

This subsection describes the settings required for mapping the attachments to the SOAP messages. For the mapping conventions, also see *30.5.2 Notes on mapping from the attachments to the SOAP messages (MTOM/XOP)*.

(1) HTTP header

The following table lists the values that are set in the HTTP header field and parameters when attachments in MTOM/XOP specification format are used.

Table 30-6: HTTP header fields and parameter settings

No.	Field name	Parameter name	Settings
1	Content-Type	--	"multipart/related" is set.#
2		`start`	Route part Content-Id is set.
3		`type`	"application/soap+xml" is set.
4		`boundary`	MIME part border string is set.
5		`start-info`	SOAP 1.1 specifications "text/xml" is set. SOAP 1.2 specifications "application/soap+xml" is set.

Legend:

--: Shows that the value is directly set in the Content-Type field.

#

Even if the MIME part is only a route part, or if at least 1 attachment MIME part is used (It is a SOAP message with MTOM/XOP type attachment), you must set "multipart/related".

(2) HTTP body

HTTP body has a route part, an attachment part, and the border strings between each part. This section describes the contents generated in each part and the values set when using the attachments.

(a) Route part MIME header

The following table lists the values that are set in the route part fields when using the attachments.

Table 30-7: Route part fields and parameter settings

No.	Field name	Parameter name	Settings
1	Content-Type	--	"application/xop+xml" is set.
2		`charset`	"utf-8" is set.
3		`type`	SOAP 1.1 specifications "text/xml" is set. SOAP 1.2 specifications "application/soap+xml" is set.
4	Content-Transfer-Encoding	None	"binary" is set.
5	Content-Id	None	"rootpart*"+" globally unique value "+"@"+"jaxws.cosminexus.com" is set.

Legend:

--: Shows that the value is directly set in the Content-Type field.

(b) Route part MIME body

When using attachments, SOAP envelope is stored as is in the MIME body of the route part. XOP is used for referencing the MTOM/XOP specification format attachments from the SOAP body that exists inside the SOAP envelope.

(c) Attachment part MIME header

The following table lists the values that are set in the MIME header of the attachment part when using attachments.

Table 30-8: Attachment part fields and parameter settings

No.	Field name	Parameter name	Settings
1	Content-Type	--	Text expression of the MIME type specified in the `value` element of the `XmlMimeType` annotation is set.[#1, #2, #3]
2		`charset`	If the target of MTOM/XOP specification format attachments is `Source` type, "UTF-8" is set.[#4] If the target of MTOM/XOP specification format attachments is `DataHandler` type, the character code specified when generating the `DataHandler` object is set.[#5] If the target of MTOM/XOP specification format attachments is a `Java` type other than those mentioned above, the parameter does not appear.
3	Content-Transfer-Encoding	None	"binary" is set.

No.	Field name	Parameter name	Settings
4	Content-Id	None	"globally unique value"+"@"+"jaxws.cosminexus.com" is set.

Legend:
--: Indicates that the value is directly set in the Content-Type field.

#1
It is a MIME type set in the Content-Type field. For the "text/xml" and "application/xml" MIME types that represent XML, we recommend "application/xml".

#2
If a MIME type with a parameter is specified in the `value` element of the `XmlMimeType` annotation, that parameter is also set in the Content-Type field.

#3
The value set in the Content-Type field changes as follows according to the method used for generating the `DataHandler` object:

- If the object is generated in the `DataHandler(DataSource)` constructor
  In `FiledataSource`, from the extension of the attachment that is used as an input, the MIME type decided by JAF is set as the value of the Content-Type field.
- If the object is generated in the `DataHandler(Object, String)` constructor
  The contents (MIME type) specified in the 2nd argument of the constructor when the `DataHandler` object is generated is set as is as the Content-Type field value.

#4
If the MIME type specified in the `xmime:expectedContentTypes` attribute or `javax.xml.bind.annotation.XmlMimeType` annotation includes charset, the charset value in the Content-Type field will be the charset value included in the specified MIME type.

#5
If the `DataHandler` object is generated and sent as an MTOM/XOP specification format attachment in the following manner, the charset parameter value `Shift_JIS` specified in the 2nd argument will be set for the Content-Type charset.

```
DataHandler dhandler = new DataHandler ("Double byte character", "text/plain;
charset=Shift_JIS");
```

The `javax.activation.DataHandler` Java type of MTOM/XOP specification format attachment is a `JavaBeans Activation Framework(JAF)` type same as the `javax.activation.DataHandler` Java type that can be used as wsi:swaRef format attachment. Hence, you can specify attachment data of any MIME type for the `javax.activation.DataHandler` type.

30.5.2 Notes on mapping from the attachments to the SOAP messages (MTOM/XOP)

This subsection describes the notes on mapping attachments to SOAP messages.

(1) Coding order of the MIME part

A MIME part has one route part and none or more attachment parts.

The route part is defined in the beginning of the MIME part. After the route part, the attachment part is defined.

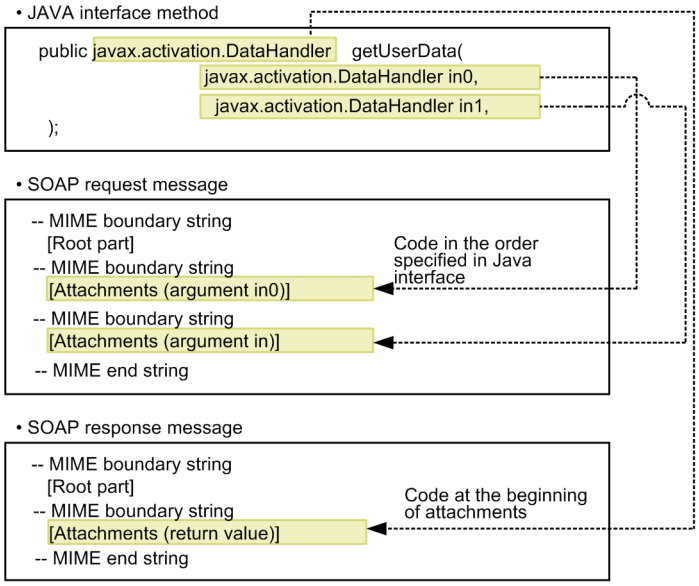
In the attachment part, the arguments or return values specified in the Java interface are mapped. The following table describes the mapping between the contents specified in the Java interface and the definition sequence in the attachments.

Table 30-9: Contents specified in the Java interface and the definition sequence in the attachment part

No.	Contents specified in the Java interface	Definition sequence in the attachment part
1	Method arguments	Defined in sequence as specified in the method arguments.
2	Method return values	Defined in the beginning of the attachment part.
3	Array type	Defined in sequence of the elements within the array.
4	User definition type	It is defined in order specified inside the user-defined type. When specifying the Java type of the attachment in the user-defined type inside the user-defined type, the attachment part is defined by depth-based priority order.

The following figure shows an example of mapping the Java interface to the attachment part.

Figure 30-3: Java interface and attachment mapping: Example



(2) Mapping the route part to the attachment

When the attachment is of MTOM/XOP specification format, the XOP information set defined in the XOP specifications is used for mapping the route part to the attachment. The XOP information is coded in the SOAP body of the route part, and the Content-Id of the corresponding attachment is set. The corresponding attachment can be referenced from the route part by the Content-Id. The following is an example of referencing an attachment from the route part. The `bold` portion indicates the CID URL schema and the Content-Id of the corresponding attachment.

```
-uuid:e63fe7dc-ad8a-4fb1-8f56-ce7b5841a06f
Content-Type: text/xml

<?xml version="1.0" ?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <getUserData xmlns="http://localhost">
      <in0>
        <xop:Include xmlns="http://www.w3.org/2004/08/xop/include" href="cid:
39820675-44bb-4e28-9926-577bf27fa07c@jaxws.cosminexus.com"/>
      </in0>
    </getUserData>
  </S:Body>
</S:Envelope>

--uuid:e63fe7dc-ad8a-4fb1-8f56-ce7b5841a06f
Content-Id:<39820675-44bb-4e28-9926-577bf27fa07c@jaxws.cosminexus.com>
Content-Type: image/jpeg
```

```
Content-Transfer-Encoding: binary

[attached data]
--e63fe7dc-ad8a-4fb1-8f56-ce7b5841a06f--
```

(a) Format of the XOP information set

The format of the XOP information set defined in the SOAP body of the route part is as follows:

```
"<xop: Include href=" + CID URL-schema + "/>"
```

(b) Mapping based on the data size of the attachments in MTOM/XOP specification format

The method of mapping the attachments in MTOM/XOP specification format to the attachment part differs according to the data size of the attachment specified in the Java type and the Java interface of the mapping source. The following table lists various mapping methods.

Table 30-10: Mapping the MTOM/XOP specification format attachments to the attachment part based on the data size

No.	Java type	Data size of the attachment	Mapping to the attachment part
1	`byte[]`	null	Mapping is not done#1
2		0 byte data	Mapping is done#2
3		Data bigger than 0 bytes	Mapping is done#3
4	`java.awt.Image` type	null	Mapping is not done#1
5		0 byte data	Mapping is not done#1
6		Data bigger than 0 bytes	Mapping is done#3
7	`javax.xml.transform.Source` type	Null	Mapping is not done#1
8		0 byte data	Mapping is done#2
9		Data bigger than 0 bytes	Mapping is done#3
10	`javax.activation.DataHandler` type	null	Mapping is not done#1
11		0 byte data	Mapping is done#2
12		Data bigger than 0 bytes	Mapping is done#3

#1

Unlike the wsi:swaRef format attachments, this is the multipart that is formed only of the route part. SOAP envelopes are stored in the MIME body of the route part. No element of the argument corresponding to the SOAP body appears. The following is an example of XML mapping when the attachment data is null:

- The program that invokes the Web Service

```
...
UserInfoService service = new UserInfoService();
UserInfo impl = service.getUserInfo(new MTOMFeature());

result = impl.getUserData(  null  );
...
```

- SOAP messages

```
--uuid:cbc0221b-8ee3-40a3-adc1-d5fa52a8d66e
Content-Id: <rootpart*cbc0221b-8ee3-40a3-adc1-d5fa52a8d66e@jaxws.cosminexus.com>
Content-Type: application/xop+xml;charset=utf-8;type="text/xml"
Content-Transfer-Encoding: binary

<?xml version="1.0" ?>
```

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <ns2:getUserData xmlns:ns2="http://localhost">
    </ns2:getUserData>
  </S:Body>
</S:Envelope>
--uuid:cbc0221b-8ee3-40a3-adc1-d5fa52a8d66e--
```

#2

This is a multipart that is formed of the attachment part that has no route part and data (MIME body is empty). If multiple attachment data is specified in the Java interface, the attachment parts will also be multiple.

#3

This is a multipart that is formed of a route part and an attachment part. If multiple attachment data is specified in the Java interface, the attachment parts will also be multiple.

30.5.3 Mapping the SOAP messages to the attachments (MTOM/XOP)

In the JAX-WS engine, mapping between the SOAP message with an attachment and the attached data is guaranteed only if the received SOAP message (with an attachment) adheres to the MTOM/XOP specifications. Behavior is not guaranteed if other type of SOAP messages is received.

30.6 Precautions

The following are the precautions when using the attachments in MTOM/XOP specification format:

- You cannot use them concurrently with the wsi:swaRef format attachments. Behavior is not guaranteed if used concurrently.
- You cannot send or receive them concurrently with the wsi:swaRef type attachments in a single operation.
- You cannot use them in the provider implementation class (class that implements the `javax.xml.ws.provider` interface). Behavior is not guaranteed if attachments in MTOM/XOP specification format are used in the provider implementation class.
- You cannot use them along with the handler framework. Behavior is not guaranteed if used along with the handler framework. To send and receive attachments when using the handler framework, use the attachment functionality (`wsi:swaRef` format).
- You cannot use them along with WSS (Cosminexus Web Services-Security). The behavior is not guaranteed if used along with WSS.

30.7 Data that can be sent and received and the Java types that can be used in the attachment (MTOM/XOP format)

The `byte[]` and other Java types can be used in the MTOM/XOP format attachment depending on the data to be sent and received. The following table describes the Java types, which can be used and the data that can be sent and received through an attachment in the MTOM/XOP format.

Table 30-11: Java types and data that can be sent and received through an attachment in the MTOM/XOP format

No.	Java type	Data that can be sent and received
1	`byte[]`	All the data can be sent and received.
2	`javax.activation.DataHandler`	
3	`java.awt.Image`	Image data can be sent and received.
4	`javax.xml.transform.Source`	XML file can be sent and received.

30.7.1 How to create Java objects for data to be sent

This subsection describes how to create a Java object depending on the data to be sent as an attachment in the MTOM/XOP format.

(1) Sending an existing text file

You can send an existing text file by using either of the two methods such as `byte[]` or `javax.activation.DataHandler`. The details on the respective procedures are as follows:

(a) Using byte[]

The procedure to send an existing text file by using `byte[]` is as follows:

1. Generating the `java.io.FileInputStream` object

   Generate the `java.io.FileInputStream` object by specifying the file path of the attachment to be sent, in an argument.

```
java.io.FileInputStream fileInputStream =
    new java.io.FileInputStream("sample.txt");
```

2. Generating `byte[]`

   Write the byte data read from the `java.io.FileInputStream` object to the `java.io.ByteArrayOutputStream` object.

   Then, generate `byte[]` by using the `toByteArray()` method of the `java.io.ByteArrayOutputStream` class.

```
java.io.ByteArrayOutputStream byteArrayOutputStream =
    new java.io.ByteArrayOutputStream();

int i = 0;
while ((i = fileInputStream.read()) != -1) {
    byteArrayOutputStream.write(i);
}

byte[] bytes = byteArrayOutputStream.toByteArray();
```

(b) Using javax.activation.DataHandler

The procedure to attach and send an existing file by using `javax.activation.DataHandler` is as follows:

1. Generating the `javax.activation.DataSource` object

   Generate the `javax.activation.FileDataSource` object by specifying the file path of the attachment to be sent, in an argument.

```
javax.activation.FileDataSource fileDataSource =
    new javax.activation.FileDataSource("sample.txt");
```

2. Generating the `javax.activation.DataHandler` object

   Generate the `javax.activation.DataHandler` object by specifying the `javax.activation.FileDataSource` object in an argument.

```
javax.activation.DataHandler dataHandler =
    new javax.activation.DataHandler(fileDataSource);
```

(2) Sending an existing image file

You can send an existing image file by using any of the three methods such as `byte[]`, `javax.activation.DataHandler`, or `java.awt.Image`. The details on the respective procedures are as follows:

(a) Using byte[]

The procedure to send an existing image file by using `byte[]` is as follows:

1. Generating the `java.io.FileInputStream` object

   Generate the `java.io.FileInputStream` object by specifying the file path of the attachment to be sent, in an argument.

```
java.io.FileInputStream fileInputStream =
    new java.io.FileInputStream("sample.png");
```

2. Generating `byte[]`

   Write the byte data read from the `java.io.FileInputStream` object to the `java.io.ByteArrayOutputStream` object.

   Then, generate `byte[]` by using the `toByteArray()` method of the `java.io.ByteArrayOutputStream` class.

```
java.io.ByteArrayOutputStream byteArrayOutputStream =
    new java.io.ByteArrayOutputStream();

int i = 0;
while ((i = fileInputStream.read()) != -1) {
    byteArrayOutputStream.write(i);
}

byte[] bytes = byteArrayOutputStream.toByteArray();
```

(b) Using javax.activation.DataHandler

The procedure to send an existing image file by using `javax.activation.DataHandler` is as follows:

1. Generating the `javax.activation.DataSource` object

   Generate the `javax.activation.FileDataSource` object by specifying the file path of the attachment to be sent, in an argument.

```
javax.activation.FileDataSource fileDataSource =
    new javax.activation.FileDataSource("sample.png");
```

2. Generating the `javax.activation.DataHandler` object

   Generate the `javax.activation.DataHandler` object by specifying the `javax.activation.FileDataSource` object in an argument.

```
javax.activation.DataHandler dataHandler =
    new javax.activation.DataHandler(fileDataSource);
```

(c) Using java.awt.Image

The procedure to send an existing image file by using `java.awt.Image` is as follows:

1. Generating the `java.awt.Image` object

   Generate the `java.awt.Image` object by specifying the file path of the attachment to be sent to an argument of the `createImage(String)` method in the `java.awt.Toolkit` class.

```
java.awt.Image image =
    Toolkit.getDefaultToolkit().createImage("sample.png");
```

(3) Sending an existing XML file

You can send an existing XML file by using any of the three methods such as `byte[]`, `javax.activation.DataHandler`, or `javax.xml.transform.Source`. The details on the respective procedures are as follows:

(a) Using byte[]

The procedure to send an existing XML file by using `byte[]` is as follows:

1. Generating the `java.io.FileInputStream` object

   Generate the `java.io.FileInputStream` object by specifying the file path of the attachment to be sent, in an argument.

```
java.io.FileInputStream fileInputStream =
    new java.io.FileInputStream("sample.xml");
```

2. Generating `byte[]`

   Write the byte data read from the `java.io.FileInputStream` object to the `java.io.ByteArrayOutputStream` object. Then, generate `byte[]` by using the `toByteArray()` method of the `java.io.ByteArrayOutputStream` class.

```
java.io.ByteArrayOutputStream byteArrayOutputStream =
    new java.io.ByteArrayOutputStream();

int i = 0;
while ((i = fileInputStream.read()) != -1) {
    byteArrayOutputStream.write(i);
}

byte[] bytes = byteArrayOutputStream.toByteArray();
```

(b) Using javax.activation.DataHandler

The procedure to send an existing XML file by using `javax.activation.DataHandler` is as follows:

1. Generating the `javax.activation.DataSource` object

   Generate the `javax.activation.FileDataSource` object by specifying the file path of the attachment to be sent, in an argument.

```
javax.activation.FileDataSource fileDataSource =
    new javax.activation.FileDataSource("sample.xml");
```

2. Generating the `javax.activation.DataHandler` object

   Generate the `javax.activation.DataHandler` object by specifying the `javax.activation.FileDataSource` object in an argument.

```
javax.activation.DataHandler dataHandler =
    new javax.activation.DataHandler(fileDataSource);
```

(c) Using javax.xml.transform.Source

The procedure to send an existing XML file by using `javax.xml.transform.Source` is as follows:

1. Generating the `javax.xml.transform.Source` object

Generate the `javax.xml.transform.stream.StreamSource` object by specifying the file path of the attachment to be sent, in an argument.

```
javax.xml.transform.stream.StreamSource streamSource =
   new javax.xml.transform.stream.StreamSource("sample.xml");
```

(4) Sending java.lang.String object

You can send the `java.lang.String` object by using either of the two methods such as `byte[]` or `javax.activation.DataHandler`. The details on the respective procedures are as follows:

(a) Using byte[]

The procedure to send the `java.lang.String` object by using `byte[]` is as follows:

1. Generating the `java.lang.String` object

```
java.lang.String str =
    new java.lang.String("abcde");
```

2. Generating `byte[]`

Generate `byte[]` by using the `getBytes()` method of the `java.lang.String` class.

```
byte[] bytes =
    str.getBytes();
```

(b) Using javax.activation.DataHandler

The procedure to send the `java.lang.String` object by using `javax.activation.DataHandler` is as follows:

1. Generating the `java.lang.String` object

```
java.lang.String str =
    new java.lang.String("abcde");
```

2. Generating the `javax.activation.DataHandler` object

Generate the `javax.activation.DataHandler` object by specifying the `java.lang.String` object and the `MIME` type, in an argument.

```
javax.activation.DataHandler dataHandler =
    new javax.activation.DataHandler(str, "text/plain; charset=UTF-8");
```

(5) Precautions on generating the javax.activation.DataHandler object

When sending a text file, an XML file or a `java.lang.String` object (String) as a `javax.activation.DataHandler` object through an attachment in the MTOM/XOP specification format, you can specify the character code of the characters to be included in the object by using the `DataHandler (Object, String)` constructor of the `javax.activation.DataHandler` class in the same way as the codes are specified in an attachment in the `wsi:swaRef` format.

For details on how to specify a character code where the `DataHandler (Object, String)` constructor was used, see *28.5.1(4) Precautions on generating the javax.activation.DataHandler object*.

30.7.2 How to acquire the received data

If the data received through an attachment in the MTOM/XOP specification format is other than the `javax.activation.DataHandler` object (`byte[]`, `java.awt.Image` object, or `javax.xml.transform.Source` instance), the JAXB converts the received data into an appropriate Java object. Therefore, you need not convert the data in an application.

If the received data is a `javax.activation.DataHandler` object, you can acquire the received data by using the same method as that for an attachment in the `wsi:swaRef` format.

For details on how to acquire an attachment in the `wsi:swaRef` format, see *28.5.2 How to acquire the attachment data (wsi:swaRef format)*.

(1) Notes on acquiring the javax.activation.DataHandler object

Receiving a SOAP message of the MIME Multipart or related structure that contains an attachment in the MTOM and XOP specification format enables you to handle a SOAP message as an attachment of the streamed MTOM/XOP specification format. Because the receiving process does not complete at the receiver side of the attachment in the MTOM/XOP format unless all the data is imported from the input stream contained in the `javax.activation.DataHandler` object, the sending process at sender side waits until the receiving process completes at the receiving side.

To come out of this state, you must either import all the data from the `java.io.InputStream` object contained in the `javax.activation.DataHandler` object or export the imported streamed data to the output stream by using the `writeTo(java.io.OutputStream)` method of the `javax.activation.DataHandler` class.

(2) Notes on acquiring the javax.xml.transform.Source object

When the data received in an attachment in the MTOM/XOP specification format is the `javax.xml.transform.Source` object, you can handle the data as an attachment in the streamed MTOM/XOP specification format. The JAXB converts the `javax.xml.transform.Source` object into the `javax.xml.transform.stream.StreamSource` object.

Because the receiving process does not complete at the receiving side of the attachment in the MTOM and XOP specification format unless all the data is imported from the input stream contained in the `javax.xml.transform.stream.StreamSource` object, the sending process at the sender side waits until the receiving process completes at the receiving side.

To eliminate this state, you must import all the data from the `java.io.Reader` object contained in the `javax.xml.transform.stream.StreamSource` object.

31 Example of the development starting from SEI (when using attachments in the MTOM/XOP specification format)

This chapter describes an example of developing a Web Service starting from SEI by using attachments.

31.1 Configuration of the development example (starting from SEI or the attachments in the MTOM/XOP specification format)

This chapter describes an example of developing a Web Service starting from SEI. The Web Service to be developed uses the attachments in the MTOM/XOP specification format.

This section gives an overview of the Web Service to be developed and the information used.

Overview of the development example

This example describes how to develop a new Web Service that will manage the user information such as employee identification number, photograph, employee name, and affiliation, and return the processing results for the input from the Web Service client.

The following tables describe the information requested from the Web Service client and the server response.

Table 31-1: Request information from the Web Service client

Information	Java data type
Employee identification number	`java.lang.String`
Facial Photograph	`javax.activation.DataHandler`

Table 31-2: Response from the server

Information	Java data type
Registration confirmation message	`java.lang.String`
Employee name	`java.lang.String`
Affiliation	`java.lang.String`

The response information from the server is stored in the user-defined `Userdata` class.

The following is the configuration of the current directory of the Web Service to be developed.

Table 31-3: Configuration of the current directory (starting from SEI and attachments)

Directory	Explanation
`c:\temp\jaxws\works\mtom\`	This is the current directory.
`server\`	Used for developing the Web Services.
`META-INF\`	Corresponds to the META-INF directory of the EAR files.
`application.xml`	Created in *31.3.4 Creating application.xml*.
`src\`	Stores the source file (`*.java`) for the Web Service. Used in *31.3.1 Creating Web Services Implementation Class*and *31.3.2 Compiling Web Services Implementation Classes*.
`WEB-INF\`	Corresponds to the WEB-INF directory of the WAR files.
`web.xml`	Created in *31.3.3 Creating web.xml*.
`classes\`	Stores the compiled class file (`*.class`). Used in *31.3.2 Compiling Web Services Implementation Classes*.
`mtom_dynamic_generate.ear`	Created in *31.3.5 Creating EAR files*.
`mtom_dynamic_generate.war`	

Directory	Explanation
client\	Used for developing the Web Service client.
src\	Stores the source file (`*.java`) of the Web Service client. Used in *31.5.1 Generating a service class*and *31.5.2 Creating Implementation Class for the Web Services client*.
classes\	Stores the compiled class file (`*.class`). Used in *27.5.3 Compiling an implementation class for the Web Service client*.
portrait.png	Used in the PNG file that is used in the Web Service client.
usrconf.cfg	Created in *31.6.1 Creating an option definition file for Java applications*.
usrconf.properties	Created in *31.6.2 Creating a user property file for Java applications*.

Change the current directory path according to the development environment.

Note that the description below uses the directory and files names mentioned in the above table. The values in *bold* used in the command execution examples or Java source are the values that are specified or generated in this example. Use appropriate values according to the environment you build.

Furthermore, in the development example described in this chapter, the Web Service and the Web Service client are developed in the same environment; however, they can be developed on different environments as well. To develop the Web Service and the Web Service client in separate environments, you must change the current directory path according to the respective environment.

31.2 Flow of the development example (starting from SEI or attachments in MTOM/XOP specification format)

The procedure for developing and executing a Web Service described in the development examples of this chapter is as follows:

Developing a Web Service

1. Create Web Services Implementation Class (*31.3.1*)
2. Compiling Web Services Implementation Class (*31.3.2*)
3. Create a `web.xml` file (*31.3.3*)
4. Create an `application.xml` file (*31.3.4*)
5. Create an EAR file (*31.3.5*)

Deploying and starting the service

1. Deploy the EAR file (*31.4.1*)
2. Start the Web Service (*31.4.2*)

Developing the Web Service client

1. Execute the `cjwsimport` command to generate a service class (*31.5.1*)
2. Create Implementation Class for the Web Services client (*31.5.2*)
3. Compile Implementation Class for the Web Services client (*31.5.3*)

Executing the Web Service

1. Create an option definition file for Java applications (*31.6.1*)
2. Create a user property file for Java applications (*31.6.2*)
3. Execute the Web Services client (*31.6.3*)

31.3 Example of Web Service development(starting from SEI or attachment in MTOM/XOP specification format)

This section describes an example of developing a Web Service (by using attachments) starting from SEI.

31.3.1 Creating a Web Service Implementation Class

Create a Web Service Implementation Class that codes the processing of the Web Service.

This example describes how to calculate the contents of the received request message and create a Web Service Implementation Class that returns a response message.

The following example shows how to create a Web Service Implementation Class.

```
package com.sample;

import java.awt.Image;
import javax.jws.Web Service;
import javax.xml.ws.soap.MTOM;
import javax.xml.bind.annotation.XmlMimeType;

@MTOM
@Web Service(serviceName="UserInfoService",targetNamespace="http://sample.com")
public class UserInfoImpl {

    public UserData getUserData(String in0, @XmlMimeType("image/png")Image in1)
        throws UserInfoException {

//Register the photograph to employee information
......

        UserData userdata = new UserData();
        //Set the registered employee name and affiliation
        if (in0.equals("1")) {
            userdata.setName("HitachiTaro");
            userdata.setSection("The personnel section");
        } if ( ......) {
            ......
    } ......

//Set registration confirmation message
        if (in1 == null) {
            userdata.setMessage("Failure(no image).");
        } else {
            userdata.setMessage("Success.");
        }
        return userdata;
}
}
```

Save the created `UserInfoImpl.java` in the `c:\temp\jaxws\works\mtom\server\src\com\sample\` directory in UTF-8 format.

Also, create a user-defined class `com.sample.UserData` used in `com.sample.UserInfoImpl`. Typically, creation of an exception class is optional, but in this example an exception class is created:

The following is an example for creating a user-defined class:

```
package com.sample;

import java.lang.String;

public class UserData {

    private String message;
    private String name;
    private String section;

    public UserData() {
    }
```

```
    public String getMessage() {
        return this.message;
    }

    public void setMessage(String message) {
        this.message = message;
    }

    public String getName() {
        return this.name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getSection() {
        return this.section;
    }

    public void setSection(String section) {
        this.section = section;
    }
}
```

The created `UserData.java` is stored in the `c:\temp\jaxws\works\mtom\server\src\com\sample\` directory in UTF-8 format.

Also, create an exception class `com.sample.UserInfoException` thrown in `com.sample.UserInfoImpl`. Typically, creation of an exception class is optional, but in this example an exception class is created:

The following is an example for creating an exception class:

```
package com.sample;

import java.lang.Exception;
import java.lang.String;

public class UserInfoException extends Exception {

    String detail;

    public UserInfoException(String message, String detail) {
        super(message);
        this.detail = detail;
    }

    public String getDetail() {
        return detail;
    }
}
```

The created `UserInfoException.java` class is stored in the `c:\temp\jaxws\works\mtom\server\src\com\sample\` directory in UTF-8 format.

31.3.2 Compiling Web Services Implementation Classes

Execute the `javac` command to compile Web Services Implementation Class. Include the Web Service implementation class during compilation. For details on the `javac` command, see the *JDK documentation*.

The following example describes the execution of the `javac` command:

```
> cd c:\temp\jaxws\works\mtom\server\
> mkdir WEB-INF\classes\
> javac -cp
"%COSMINEXUS_HOME%\jaxws\lib\cjjaxws.jar;%COSMINEXUS_HOME%\CC\client\lib\j2ee-
javax.jar;%COSMINEXUS_HOME%\CC\client\lib\HiEJBClientStatic.jar;%COSMINEXUS_HOME%\jaxp
\lib\csmjaxb.jar;%COSMINEXUS_HOME%\jaxp\lib\csmjaxp.jar;%COSMINEXUS_HOME%\jaxp\lib
\csmstax.jar" -d WEB-INF\classes\ -s src src\com\sample\UserInfoImpl.java src\com
\sample\UserData.java src\com\sample\UserInfoException.java
```

On successful execution of the `javac` command, the compiled classes are output to the following path: `c:\temp\jaxws\works\mtom\server\WEB-INF\classes\com\sample\Directory`

You can execute the `cjwsgen` command for the compiled Web Services Implementation Classes to check errors in advance. For details on the `cjwsgen` command, see *10.23(1) Using the cjwsgen command for checking errors*.
Creating web.xml

Create a `web.xml` file that is required as a WAR file component.

The following is an example for creating a `web.xml` file:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/
javaee/web-app_3.0.xsd">
  <description>Sample Web Service &quot;mtom_dynamic_generate &quot;</description>
  <display-name>Sample_web_service_mtom_dynamic_generate </display-name>
  <listener>
    <listener-class>
      com.cosminexus.xml.ws.transport.http.servlet.WSServletContextListener
    </listener-class>
  </listener>
  <servlet>
    <description>Endpoint servlet for Cosminexus JAX-WS</description>
    <display-name>Endpoint_servlet_for_Cosminexus_JAX_WS</display-name>
    <servlet-name>CosminexusJaxwsServlet</servlet-name>
    <servlet-class>
      com.cosminexus.xml.ws.transport.http.servlet.WSServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>

    <servlet-name>CosminexusJaxwsServlet</servlet-name>
    <url-pattern>/UserInfoService</url-pattern>
  </servlet-mapping>
  <session-config>
    <session-timeout>60</session-timeout>
  </session-config>
</web-app>
```

When creating `web.xml` of version 2.5, specify `2.5` in the `version` attribute of the `web-app` element and specify `http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd` as the second location information in the `xsd:schemaLocation` attribute.

The created `web.xml` is stored in the `c:\temp\jaxws\works\mtom\server\WEB-INF\` directory in UTF-8 format. For the `web.xml` settings, see *3.4 Creating web.xml*.

31.3.3 Creating a web.xml file

Create a `web.xml` file that is required as a WAR file component.

The following is an example for creating a `web.xml` file:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/
javaee/web-app_3_0.xsd">
  <description>Sample web service &quot;mtom_dynamic_generate&quot;</description>
  <display-name>Sample_web_service_mtom_dynamic_generate</display-name>
  <listener>
    <listener-class>
      com.cosminexus.xml.ws.transport.http.servlet.WSServletContextListener
    </listener-class>
  </listener>
  <servlet>
    <description>Endpoint servlet for Cosminexus JAX-WS</description>
    <display-name>Endpoint_servlet_for_Cosminexus_JAX_WS</display-name>
    <servlet-name>CosminexusJaxwsServlet</servlet-name>
    <servlet-class>
      com.cosminexus.xml.ws.transport.http.servlet.WSServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
```

```
  </servlet>
  <servlet-mapping>

    <servlet-name>CosminexusJaxwsServlet</servlet-name>
    <url-pattern>/UserInfoService</url-pattern>
  </servlet-mapping>
  <session-config>
    <session-timeout>60</session-timeout>
  </session-config>
</web-app>
```

When creating `web.xml` of version 2.5, specify `2.5` in the `version` attribute of the `web-app` element and specify `http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd` as the second location information in the `xsd:schemaLocation` attribute.

The created `web.xml` file is stored in the `c:\temp\jaxws\works\wsrm\server\WEB-INF\` directory in the UTF-8 format. For the `web.xml` settings, see *3.4 Creating web.xml*.

31.3.4 Creating an application.xml file

Create an `application.xml` file that is required as an EAR file component.

The following is an example for creating an `application.xml` file:

```
<?xml version="1.0" encoding="UTF-8"?>
<application version="6" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/
javaee/application_6.xsd">

  <description>Sample application &amp;quot;mtom_dynamic_generate&amp;quot;</description>
  <display-name>Sample_application_mtom_dynamic_generate</display-name>
  <module>
    <web>
      <web-uri>mtom_dynamic_generate.war</web-uri>
      <context-root>mtom_dynamic_generate</context-root>
    </web>
  </module>
</application >
```

When creating `application.xml` of version 5, specify `5` in the `version` attribute of the `application` element and specify `http://java.sun.com/xml/ns/javaee/application_5.xsd` as the second location information in the `xsd:schemaLocation` attribute.

The created `application.xml` file is stored in the `c:\temp\jaxws\works\wsrm\server\META-INF\` directory in the UTF-8 format. For notes on creating an `application.xml` file, see *5.2.2 Notes on editing application.xml* in the *uCosminexus Application Server Application Development Guide*.

31.3.5 Creating EAR files

Use the `jar` command to create EAR files.

The following is an example for creating an EAR file:

```
> cd c:\temp\jaxws\works\mtom\server\
> jar cvf mtom_dynamic_generate.war .\WEB-INF
> jar cvf mtom_dynamic_generate.ear .\mtom_dynamic_generate.war .\META-INF
\application.xml
```

On successful termination of the `jar` command, `wsrm.ear` is created in the `c:\temp\jaxws\works\mtom_dynamic_generate\server\directory.`

For the `jar` command, see the *JDK documentation*.

31.4 Examples of deployment and startup (Starting from SEI or attachments in the MTOM/XOP specification format)

This section describes how to deploy and start services for the development starting from SEI by using attachments.

31.4.1 Deploying EAR files

Use the `cjimportapp` command to deploy the created EAR file to a J2EE server.

The following is an example of deployment:

```
> cd c:\temp\jaxws\works\mtom\server\
> "%COSMINEXUS_HOME%\CC\admin\bin\cjimportapp" jaxwsserver -nameserver
corbaname::testserver:900 -f mtom_dynamic_generate.ear
```

For the `cjimportapp` command, see *cjimportapp (Importing J2EE applications)* in the *uCosminexus Application Server Command Reference Guide*.

For the method to deploy (or import) J2EE applications by using the management portal, see *12.3.3 Importing J2EE Applications* in the *uCosminexus Application Server Management Portal User Guide*.

31.4.2 Starting Web Service

Use the `cjstartapp` command to start the Web Service.

The following is an example for starting the Web Service:

```
> cd c:\temp\jaxws\works\mtom\server\
> "%COSMINEXUS_HOME%\CC\admin\bin\cjstartapp" jaxwsserver -nameserver
corbaname::testserver:900 -name Sample_application_mtom_dynamic_generate
```

For the `cjstartapp` command, see *cjimportapp (Starting J2EE applications)* in the *uCosminexus Application Server Command Reference Guide*.

For the method to start J2EE applications by using the management portal, see *12.3.1 Starting J2EE Applications* in the *uCosminexus Application Server Management Portal User Guide*.

31.5 Examples of developing the Web Service clients (starting from SEI or attachments in the MTOM/XOP specification format)

This section describes the examples for the development of Web Service client starting from SEI by using attachments.

31.5.1 Generating a service class

If you execute the `cjwsimport` command, the Java source, such as a service class that is required for developing the Web Service client will be generated. For the `cjwsimport` command, see *14.1 cjwsimport command*.

The following is an example for executing the `cjwsimport` command:

```
> cd c:\temp\jaxws\works\mtom\client\
> mkdir src\
> mkdir classes\
> "%COSMINEXUS_HOME%\jaxws\bin\cjwsimport.bat" -s src -d classes http://webhost:8085/
mtom_dynamic_generate/UserInfoService?wsdl
```

On successful termination of the `cjwsimport` command, the Java source is generated in the `c:\temp\jaxws\works\mtom\client\src\com\sample\` directory.

The following table lists the generated products:

Table 31-4: Products generated when the service class is generated (starting from SEI or attachments)

File name	Explanation
`GetUserData.java`	This is a `JavaBean` class corresponding to the type referenced by the wrapper element of the request message in the WSDL definition.
`GetUserDataResponse.java`	This is a `JavaBean` class corresponding to the type referenced by the wrapper element of the response message in the WSDL definition.
`ObjectFactory.java`	This is an `ObjectFactory` class of the JAXB 2.2 specifications.
`package-info.java`	This is a `package-info.java` file.
`UserData.java`	This is the `UserData` corresponding to `JavaBean` class.
`UserInfoImpl.java`	This is the SEI corresponding to 'Service' in the WSDL definition.
`UserInfoService.java`	This is a service class.
`UserInfoException.java`	This is a `JavaBean` class corresponding to `UserInfoException`.
`UserInfoException_Exception.java`	This is a wrapper exception class of the fault bean.

31.5.2 Creating an implementation class for the Web Service client

Create an implementation class for the Web Service client that uses the Web Service.

The following is an example for creating a Web Service client that invokes Web Services once:

```
package com.sample.client;

import java.awt.Image;
import java.io.File;
```

```
import javax.imageio.ImageIO;
import javax.xml.ws.soap.MTOMFeature;

import com.sample.UserInfoImpl;
import com.sample.UserData;
import com.sample.UserInfoService;
import com.sample.UserInfoException_Exception;

public class TestClient {

    public static void main( String[] args ) {
        try {
            // Generate image object
            File imageFile = new File("portrait.png");
            if (!imageFile.exists()) {
                throw new RuntimeException("Cannot find the file \"portrait.png\".");
            }
            Image image = ImageIO.read(imageFile);

            UserInfoService service = new UserInfoService();
            UserInfoImpl port = service.getUserInfoImplPort(new MTOMFeature());

            UserData userdata = port.getUserData("1", image);

            System.out.print("[RESULT] " + userdata.getMessage());
            System.out.println(" Name:" + userdata.getName()
                               + ", Section:" + userdata.getSection());
        } catch(UserInfoException_Exception e) {
            e.printStackTrace();
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

The created `TestClient.java` is stored in the `c:\temp\jaxws\works\mtom\client\src\com\sample\client\` directory in UTF-8 format.

31.5.3 Compiling an implementation class for the Web Service client

Use the `javac` command to compile the created Web Service client.

The following example describes how to compile the created Web Service client:

```
> cd c:\temp\jaxws\works\mtom\client\
> javac -encoding UTF-8 -cp "%COSMINEXUS_HOME%\jaxws\lib\cjjaxws.jar;%COSMINEXUS_HOME%
\CC\client\lib\j2ee-javax.jar;%COSMINEXUS_HOME%\jaxp\lib\csmjaxb.jar;.\classes" -d .
\classes src\com\sample\client\TestClient.java
```

On successful termination of the `javac` command, the `TestClient.class` is generated in the `c:\temp\jaxws\works\mtom\client\classes\com\sample\client\` directory. For the `javac` command, see the *JDK documentation*.

31.6 Examples for executing the Web Services (starting from SEI or attachments in the MTOM/XOP specification format)

This section describes the example for the execution of Web Service clients starting from SEI by using attachments.

31.6.1 Creating an option definition file for Java applications

Create an option definition file for Java applications (`usrconf.cfg`) required for executing the Web Service.

The following is an example for creating the option definition file for Java applications:

```
add.class.path=Cosminexus-Installation-directory\jaxws\lib\cjjaxws.jar
add.class.path=.\classes
ejb.client.log.directory=logs
add.jvm.arg=-Dcosminexus.home=Cosminexus-installation-directory
```

For the *Cosminexus-installation-directory* part, use the absolute path to specify the path where Cosminexus is installed

The created option definition file for Java applications is stored in the `c:\temp\jaxws\works\mtom\client\` directory. For the option definition file for Java applications, see *14.2.usrconf.cfg (Option definition file for Java applications) in the uCosminexus Application Server Definition Reference Guide*.

31.6.2 Creating a user property file for Java applications

Create a user property file for the Java applications required for executing the Web Service.

Because the settings are not particularly changed, create an empty file named `usrconf.properties` in `c:\temp\jaxws\works\mtom\client\` directory. For the user property file for Java applications, see *14.3 usrconf.properties (User property file for Java applications)* in the *uCosminexus Application Server Definition Reference Guide.*

31.6.3 Executing Web Service clients

Use the `cjclstartap` command to execute a Web Service client.

The following is an example for executing the Web Service client:

```
> cd c:\temp\jaxws\works\mtom\client\
> "%COSMINEXUS_HOME%\CC\client\bin\cjclstartap"  com.sample.client.TestClient
```

On successful termination of the `cjclstartap` command, the execution results of the Web Service client are displayed. The following is an example for displaying the execution results:

```
KDJE40053-I The cjclstartap command will now start. (directory for the user
definition file = c:\temp\jaxws\works\mtom\client, PID = 2636)
[RESULT] Success. Name: HitachiTaro, Section:The personnel section
KDJE40054-I The cjclstartap command was stopped. (PID = 2636, exit status = 0)
```

The part in italics changes according to the execution timing and the environment.

For the `cjclstartap` command, see *cjclstartap (Starting Java applications)* in the *uCosminexus Application Server Command Reference Guide.*

32 Streaming

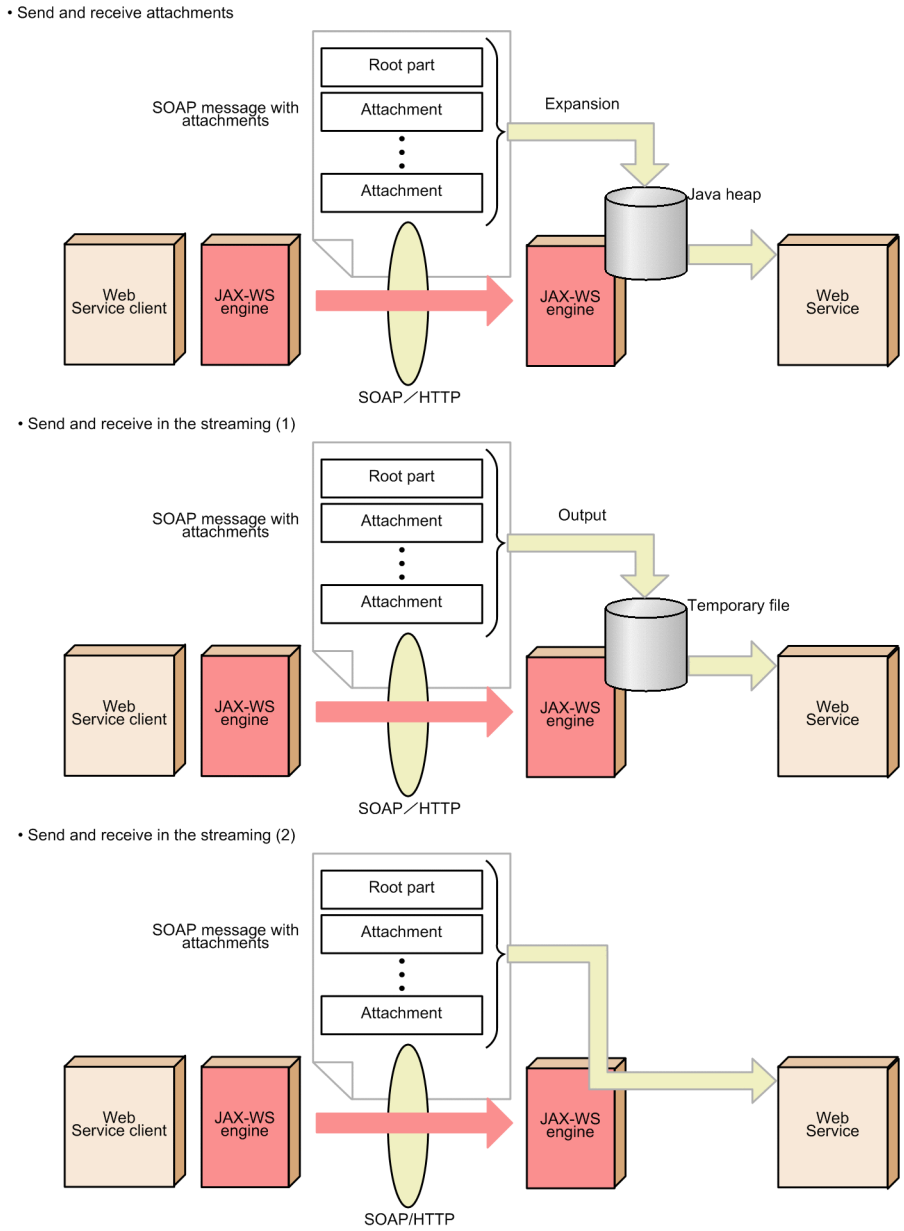
This chapter gives an overview of the Cosminexus Streaming functionality, how to use the functionality, and the settings required to use the functionality.

32.1 What is the Streaming functionality

Streaming is a functionality used in the attachment functionality to receive SOAP Messages (of MIME Multipart/Related structure) containing large attachments, without Java heap size restrictions. This is possible, because the processing is carried out without extracting the big size MIME body included in the SOAP Messages to the memory. You can use the Streaming functionality to map the SOAP messages containing the received attachment to the `javax.activation.DataHandler` class.

The following figure shows the send/receive of binary data by using the Streaming functionality:

Figure 32-1: Binary data send/receive by using Streaming



You can use Streaming only when using attachments in the MTOM/XOP specification format. You cannot use Streaming for attachments in the `wsi:swaRef` format.

32.2 How to use Streaming

To use Streaming, you must set on the receiver side of the SOAP messages of MIME Multipart/Related structure containing attachments. This section describes how to use the Web Service side and the Web Service client side.

32.2.1 Web Services machine

To use Streaming in the Web Service, annotate the `com.sun.xml.ws.developer.StreamingAttachment` annotation in the Web Service Implementation Class. For the `com.sun.xml.ws.developer.StreamingAttachment` annotation, see *16.2.2 com.sun.xml.ws.developer.StreamingAttachment annotation*.

The following is an example of the Web Service Implementation Class that uses Streaming. In this example, "`C:/TMP`" directory is specified as the output destination of the temporary file created by Streaming. Perform detailed parsing of SOAP messages containing attachments and output MIME bodies of 50,000 bytes or more as temporary files.

```
package com.sample;

......

@MTOM
@StreamingAttachment(dir="C:/TMP", parseEagerly=true, memoryThreshold=50000L)
@BindingType(...)
public class UserInfoImpl implements UserInfo {

    public DataHandler getUserInfo(DataHandler dataHandler)
        throws UserDefinedException {
        if (dataHandler instanceof StreamingDataHandler) {
            StreamingDataHandler sdh = null;
            try {
                sdh = (StreamingDataHandler)dataHandler;
                ......
            } finally {
                try {
                    if (sdh != null) {
                        sdh.close();
                    }
                } catch(Exception ex) {
                    ......
                }
            }
        }
        ......    }
}
```

If you are using Streaming in a Web Service side, and you receive a request message that satisfies all the following conditions, data of the response message will be in Base64 format.

- There is no `application/xop+xml` in the Accept field of the HTTP header of the request message.
- There is no `application/xop+xml` in the Content-Type of the root part of the request message.

32.2.2 Web Service client side

To use Streaming in the Web Service client, set the `com.sun.xml.ws.developer.StreamingAttachmentFeature` class while acquiring SEI.

For the `com.sun.xml.ws.developer.StreamingAttachmentFeature` class, see *19.2.4 (1) com.sun.xml.ws.developer.StreamingAttachmentFeature class*.

The following is an example of a Web Service client that uses Streaming. In this example "`C:/TMP`" directory is specified as the output destination of the temporary file created by Streaming. Perform detailed parsing of SOAP messages containing attachments and output the MIME bodies of 50,000 bytes or more as temporary files.

```
package com.sample;
```

```
......

public class TestClient {

    public static void main(String[] args) {
        try {
            File portrait = new File("portrait.png");
            FileDataSource fileDataSource = new FileDataSource(portrait);
            DataHandler dataHandler = new DataHandler(fileDataSource);

            MTOMFeature mtomFeature = new MTOMFeature();
            StreamingAttachmentFeature streamingAttachmentFeature = new
StreamingAttachmentFeature("C:/TMP", true, 50000L);

            UserInfoService service = new UserInfoService();
            UserInfoImpl port = service.getUserInfoImplPort(mtomFeature,
streamingAttachmentFeature);
            DataHandler userData = port.getUserInfo(dataHandler);
            if (userData instanceof StreamingDataHandler) {
                StreamingDataHandler sdh = null;
                try {
                    sdh = (StreamingDataHandler)userData;
                    sdh.moveTo(file);
                    ......
                } finally {
                    try {
                        if (sdh != null) {
                            sdh.close();
                        }
                    } catch(Exception ex) {
                        ......
                    }
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

32.2.3 Variations due to parseEagerly

In Streaming, the timing of parsing the SOAP messages containing attachments, analysis results, and exceptions that occur if there are errors in SOAP messages vary according to the value specified in the `parseEagerly` in the argument of the `com.sun.xml.ws.developer.StreamingAttachmentFeature` class and `com.sun.xml.ws.developer.StreamingAttachment` annotation element.

The following table lists the `parseEagerly` values and variations:

Table 32-1: parseEagerly values and variations

No.	`parseEagerly` value	Timing for parsing the SOAP messages	Exception thrown if there is any error in SOAP messages
1	`true`	When a SOAP message containing an attachment is unmarshalled.	`org.jvnet.mimepull.MIMEParsingException`
2	`false`	When a streamed attachment is operated.	`java.io.IOException`

Specify `true` as the `parseEagerly` value when an error in the SOAP message containing an attachment is detected not in user applications, but while un-marshaling.

32.2.4 Operating streamed attachments

When using Streaming, if you receive a SOAP message of MIME Multipart/Related structure containing an attachment, the received attachment is mapped to the `com.sun.xml.ws.developer.StreamingDataHandler` class instead of the `javax.activation.DataHandler` class in JAX-WS, and can be handled as a streamed attachment.

For the `com.sun.xml.ws.developer.StreamingDataHandler` class, see *19.2.4 (2) com.sun.xml.ws.developer.StreamingDataHandler class*.

The following is an example of operating a streamed attachment. In this example, the streamed attachment is output with a different name as "`C:/portrait.png`":

```
package com.sample;

......

@MTOM
@StreamingAttachment(dir="C:/TMP", parseEagerly=true, memoryThreshold=50000L)
@BindingType(...)
public class UserInfoImpl implements UserInfo {

    public DataHandler getUserInfo(DataHandler dataHandler)
        throws UserDefinedException {
        if (dataHandler instanceof StreamingDataHandler) {
            File file = new File("C:/portrait.png");
            StreamingDataHandler sdh = null;
            try {
                sdh = (StreamingDataHandler)dataHandler;
                sdh.moveTo(file);
                ......
            } finally {
                try {
                    if (sdh != null) {
                        sdh.close();
                    }
                } catch(Exception ex) {
                    ......
                }
            }
        }
    }
}
```

(1) Points to be noted when operating the streamed attachments

The following are the points to be noted when operating the streamed attachments:

- When using Streaming, you must distinguish the `com.sun.xml.ws.developer.StreamingDataHandler` class with the help of an `instanceof` operator.
- Irrespective of whether the attachments are used, you must close the attachments that are output in the temporary files by using the `com.sun.xml.ws.developer.StreamingDataHandler#close()` method. If you do not close the attachments, JAX-WS will not remove the temporary files.
- You can invoke only the `com.sun.xml.ws.developer.StreamingDataHandler#close()` method after invoking the `com.sun.xml.ws.developer.StreamingDataHandler#readOnce()` and `com.sun.xml.ws.developer.StreamingDataHandler#moveTo(File)` methods. The operation is not guaranteed if the methods other than the `com.sun.xml.ws.developer.StreamingDataHandler#close()` method are invoked.
- If you invoke the `com.sun.xml.ws.developer.StreamingDataHandler#readOnce()` method, you must close the input stream after reading all the data from the input stream acquired when the attachment was no longer required. Thereafter, you must invoke the `com.sun.xml.ws.developer.StreamingDataHandler#close()` method. The following is an example of invoking the `com.sun.xml.ws.developer.StreamingDataHandler#readOnce()` method:

```
package com.sample;
...

@MTOM
@StreamingAttachment(...)
...
public class UserInfoImpl implements UserInfo {

    public @XmlMimeType("application/octet-stream")
           DataHandler getUserInfo(
               @XmlMimeType("application/octet-stream")
```

```
                DataHandler dataHandler)
        throws ... {
        if (dataHandler instanceof StreamingDataHandler) {
            StreamingDataHandler sdh = null;
            try {
                sdh = (StreamingDataHandler)dataHandler;
                ...
                InputStream inputStream = sdh.readOnce();
                ...
                // read all the data of the attachment from the input stream
         while ((inputStream.read()) != -1);

                //close the input stream
                inputStream.close();
            } finally {
                try {
                    if (sdh != null) {
                        //invoke the close() method of the StreamingDataHandler
                        //class
                        sdh.close();
                    }
                } catch(Exception ex) {
                    ...
                }
            }
        }
        ...
    }
}
```

- If you invoke the `com.sun.xml.ws.developer.StreamingDataHandler#moveTo(File)` method, you must invoke the `com.sun.xml.ws.developer.StreamingDataHandler#close()` method when the attachment is no longer required.
- If an exception occurs when you invoke the `com.sun.xml.ws.developer.StreamingDataHandler#moveTo(File)` method, you must acquire the input stream by invoking the `com.sun.xml.ws.developer.StreamingDataHandler#readOnce()` method. Close the input stream after reading all the data from that input stream. Thereafter, you must invoke the `com.sun.xml.ws.developer.StreamingDataHandler#close()` method. The following is an example of invoking the `com.sun.xml.ws.developer.StreamingDataHandler#moveTo(File)` method:

```
package com.sample;
...

@MTOM
@StreamingAttachment(...)
...
public class UserInfoImpl implements UserInfo {

    public @XmlMimeType("application/octet-stream")
           DataHandler getUserInfo(
               @XmlMimeType("application/octet-stream")
               DataHandler dataHandler)
        throws ... {
        if (dataHandler instanceof StreamingDataHandler) {
            StreamingDataHandler sdh = null;
            try {
                File file = new File(...);
                sdh = (StreamingDataHandler)dataHandler;
                ...
                sdh.moveTo(file);
            } catch(Exception e)
                try {
                    if (sdh != null) {
                        InputStream inputStream = sdh.readOnce();

                        //read all the data of the attachment from the
                        //input stream
                        while ((inputStream.read()) != -1);

                        //close the input stream
                        inputStream.close();
                    }
                } catch(Exception ex) {
                    ...
                }
```

```
            } finally {
                try {
                    if (sdh != null) {
                        //invoke the close()method of the StreamingDataHandler
class
                        sdh.close();
                    }
                } catch(Exception ex) {
                    ...
                }
            }
        }
        ...
    }
}
```

- If you do not invoke the `com.sun.xml.ws.developer.StreamingDataHandler#readOnce()` and `com.sun.xml.ws.developer.StreamingDataHandler#moveTo(File)` methods, you must acquire the input stream by invoking the `com.sun.xml.ws.developer.StreamingDataHandler#readOnce()` method when the attachment is no longer required. Close the input stream after reading all the data from that input stream. Thereafter, you must invoke the `com.sun.xml.ws.developer.StreamingDataHandler#close()` method.
- Among the methods of the `javax.activation.DataHandler` class, the `com.sun.xml.ws.developer.StreamingDataHandler` class can invoke only the `getContentType()` method. The operation is not guaranteed if any other method is invoked.
- Do not send the streamed attachments of the received `javax.activation.DataHandler` class or the `javax.xml.ws.Holder<DataHandler>` class as it is in the Web Service client or Web Service Implementation Class after using them. While sending, generate a new `javax.activation.DataHandler` object, and then send the received streamed attachment.

32.3 Temporary files (Streaming)

When using Streaming, if you receive a SOAP message of MIME Multipart/Related structure containing an attachment, the MIME body contained in the SOAP message might be output to a temporary file and not to the memory. Whether the MIME body contained in the SOAP message is extracted to the memory or output to the temporary file is determined based on the value specified in the `parseEagerly`, type of MIME parts, and the MIME body size.

The following table lists the output destination of the MIME body contained in the SOAP message:

Table 32-2: Output destination of MIME bodies contained in SOAP messages

No.	Type of MIME Parts	Value specified in parseEagerly	MIME body size	Output destination of MIME body
1	Root Part	--	Greater than the threshold value #	Output to the temporary file instead of deploying to the memory.
2			Less than or equal to the threshold value#	Deploy to the memory instead of an output to a temporary file.
3	Attachment Part	true	Greater than threshold #	Outputs to the temporary file instead of extracting to the memory.
4			Below the threshold#	Extracts to the memory instead of performing output to a temporary file.
5		false	None	Outputs neither to a temporary file nor to the memory.

Legend:

--: The specified value does not have an effect on the output destination of the MIME body.

#

Threshold is the value of the `memoryThreshold` element of the `com.sun.xml.ws.developer.StreamingAttachment` annotation or the `memoryThreshold` value of the `com.sun.xml.ws.developer.StreamingAttachmentFeature` class.

Determine whether to output the MIME body included in the SOAP message to the temporary file after evaluating MIME bodies of each MIME part. If there is one root part and two attachment parts in the SOAP message of the MIME Multipart/Related structure that contains the received attachment, the evaluation to determine whether to output a temporary file will be executed three times.

32.3.1 Naming convention

The name of the temporary file output by Streaming will have the prefix "MIME", and the suffix ".tmp". This name is automatically assigned by JAX-WS and therefore cannot be renamed.

The following is an example of a temporary file name output by Streaming:

```
MIME6838906861691549713.tmp
```

In the following cases, if you specify the same directory as the output destination of the temporary file, the temporary file name might get duplicated.

- When the Web Service or the Web Service client for which Streaming is used is deployed to multiple J2EE servers
- When using streaming on the Web Service clients with different processes
- A combination of the above mentioned cases

Therefore, you must change the output destination of the temporary file for each J2EE server if you deploy the Web Service or the Web Service client (for which Streaming is used) to multiple J2EE servers, or for each client if you choose to perform streaming on the Web Service clients with different processes.

32.3.2 Output and Deletion

A temporary file will be output when a SOAP message of MIME Multipart/Related structure containing an attachment is received. The temporary file corresponding to the MIME body of the root part is deleted by the JAX-WS engine after un-marshaling the SOAP message. The temporary file corresponding to the MIME body of the attachment part is deleted when the streamed attachment is closed (when the `StreamingDataHandler#close()` method is invoked).

While receiving the SOAP messages of the MIME Multipart/Related structure, the temporary file might remain undeleted if the connection breaks or if the JVM shuts down. Delete the temporary file if it is not deleted automatically.

32.3.3 How to estimate

The size of the temporary file output by Streaming is of the same size as the MIME body contained in a received SOAP Message of MIME Multipart/Related structure. You can calculate the maximum disk usage of the temporary files by using the following formula:

- Maximum disk usage of the temporary file (Web Service):

  Maximum disk usage of the temporary files = Maximum size of the MIME body to be received × XXX123123xxxXXX Maximum number of MIME bodies to be received × XXX123123xxxXXX Number of concurrent executions of the Web Services using Streaming

- Maximum disk usage of the temporary files (Web Service client):

  Maximum disk usage of the temporary files = Maximum size of the MIME body to be received × XXX123123xxxXXX Maximum number of MIME bodies to be received × XXX123123xxxXXX Web Service invocation count

33 Example of the development starting from SEI (when using streaming)

This chapter describes the examples for the development of Web Services starting from SEI by using the streaming functionality.

33.1 Starting from development example (starting from SEI and streaming)

This chapter describes examples for developing the Web Services starting from SEI. Note that this section describes how to develop the Web Services in the implicit SEI format (a format that does not create SEI in the explicit format) as described in the paragraph 3.3 of the JAX-WS2.2 specifications.

The following table describes the configuration of the Web Service to be developed:

Table 33-1: Configuration of the Web Service (Starting from SEI and addressing)

No.	Item		Value
1	Name of the J2EE server to be deployed		`jaxwsserver`
2	Host name and port number of the Web server		`webhost:8085`
3	Naming Server URL		`corbaname::testserver:900`
4	Context route		`addressing_dynamic_generate`
5	Style		`document/literal/wrapped`
6	Namespace URI		`http://sample.com`
7	Port type	Number	`1`
8		Local name	`AddNumbersImpl`
9	Operation	Number	`3`
10		Local name 1	`add`
11		Local name 2	`add2`
12		Local name 3	`add3`
13	Service	Number	`1`
14		Local name	`AddNumbersImplService`
15	Port	Number	`1`
16		Local name	`AddNumbersImplPort`
17	Web Service Implementation Class		`com.sample.AddNumbersImpl`
18	Methods made public in the Web Service Implementation Class	Number	`3`
19		Method name 1	`add`
20		Method name 2	`add2`
21		Method name 3	`add3`
22	Exceptions thrown in the Web Service Implementation Class methods	Number	`1`
23		Class name	`com.sample.AddNumbersFault`

The following table describes the configuration of the current directory, when the Web Service is developed:

Table 33-2: Configuration of the current directory (Starting from SEI and addressing)

Directory	Explanation
`c:\temp\jaxws\works\ streaming`	This is the current directory.
`server\`	Used for Web Service development.

Directory	Explanation
META-INF\	Corresponds to the META-INF directory of the EAR file.
application.xml	Created in *33.3.4 Creating application.xml*.
src\	Saves the source file (*.java) for the Web Service. Used in *33.3.1 Creating Web Services Implementation Class*and *33.3.2 Compiling Web Services Implementation Class*.
WEB-INF\	Corresponds to the WEB-INF directory of the WAR file.
web.xml	Created in *33.3.3 Creating web.xml*.
classes\	Saves the compiled class file (*.class). Used in *33.3.2 Compiling Web Services Implementation Clas*s.
addressing_dynamic_generate.ear	Created in *33.3.5 Creating EAR files*.
addressing_dynamic_generate.war	Created in *33.3.5 Creating EAR files*.
client\	Used for the development of the Web Services client.
src\	Saves the source file (*.java) of the Web Service client. Used in *33.5.1 Generating a service class* and *33.5.2 Creating Implementation Class for the Web Services client*.
classes\	Saves the compiled class file (*.class). Used in *33.5.3 Compiling Implementation Class for the Web Services client*.
usrconf.cfg	Created in *33.6.1 Creating an option definition file for Java applications*.
usrconf.properties	Created in *33.6.2 Creating a user property file for Java applications*.

Change the current directory path according to the environment to be developed.

Note that the directory and file names listed in the above table will be used in the description hereafter. The part formatted in the *gothic* font, in the command execution example and in the Java source, indicates the specified values and generated values that are used in this example. Substitute and read according to the environment you want to build.

Also, in the development examples described in this chapter, the Web Service and Web Service client are developed in the same environment, but you can also develop them in separate environments. For developing the Web Service and Web Service client in different environments, substitute and read the current directory path suitable to the respective environments.

33.2 Flow of development examples (Starting from SEI and streaming)

This section describes the development examples for the development and execution flow.

Developing a Web Service

1. Creating Web Services Implementation Class (*33.3.1*)
2. Executing the `javac` command to compile Web Services Implementation Class(*33.3.2*)
3. Creating `web.xml` (*33.3.3*)
4. Creating `application.xml` (*33.3.4*)
5. Creating EAR files (*33.3.5*)

Deploying and starting

1. Deploying EAR files (*33.4.1*)
2. Starting Web Services (*33.4.2*)

Developing a Web Service client

1. Executing the `cjwsimport` command to generate a service class (*33.5.1*)
2. Creating Implementation Class for the Web Services client (*33.5.2*)
3. Compiling Implementation Class for the Web Services client (*33.5.3*)

Executing a Web Service

1. Creating an option definition file for Java applications (*33.6.1*)
2. Creating a user property file for Java applications (*33.6.2*)
3. Executing the Web Services client (*33.6.3*)

33.3 Examples of Web Service development (Starting from SEI and streaming)

This section describes the examples for developing Web Services starting from SEI (using streaming).

33.3.1 Creating the Web Service Implementation Class

Create a Web Service Implementation Class to code the processing of Web Services. This subsection describes how to calculate the contents of the received request message and create the Web Service Implementation Class that returns response messages.

The following is an example for creating a Web Service Implementation Class:

```
package com.sample;


import javax.activation.DataHandler;
import javax.jws.WebService;
import javax.xml.ws.soap.MTOM;
import javax.xml.bind.annotation.XmlMimeType;
import com.sun.xml.ws.developer.StreamingAttachment;
import com.sun.xml.ws.developer.StreamingDataHandler;

@MTOM
@StreamingAttachment(dir="C:/TMP", parseEagerly=true, memoryThreshold=50000L)
@WebService(serviceName="UserInfoService",targetNamespace="http://sample.com")
public class UserInfoImpl {

    public UserData getUserData(String in0, @XmlMimeType("application/octet-
stream")DataHandler in1)
        throws UserInfoException {

        if (in1 != null) {
            if (in1 instanceof StreamingDataHandler) {
                StreamingDataHandler sdh = null;
                try {
                    //Registering a photograph to the employee information
                    sdh = (StreamingDataHandler)in1;
                    ......
                } catch(Exception e) {
                    throw new UserInfoException("Exception occurred.",
e.getMessage());
                } finally {
                    try {
                        if (sdh != null) {
                            //Closing the data
                            sdh.close();
                        }
                    } catch(Exception ex) {
                        ......
                    }
                }
            }
        }

        UserData userdata = new UserData();
        //Setting the name and affiliation of the registered employee
        if (in0.equals("1")) {
            userdata.setName("Hitachi Taro ");
            userdata.setSection("The personnel section");
        } if (....) {
            ....
        } ...

        //Setting the registration confirmation message
        if (in1 == null) {
            userdata.setMessage("Failure(no image).");
        } else {
            userdata.setMessage("Success.");
        }
        return userdata;
    }
}
```

The created AddNumbers.java is saved in the c:\temp\jaxws\works\addressing\server\src\com\sample\ directory with the UTF-8 format.

Next, create the main Web Service that implements SEI. This subsection describes how to calculate the contents of the received request message and create the Web Service Implementation Class com.sample.AddNumbersImpl that is returned as a response message.

The following is an example for creating the main Web Service:

```
package com.sample;

import javax.jws.WebService;
import javax.xml.ws.soap.Addressing;

@Addressing
@WebService(endpointInterface = "com.sample.AddNumbers")
public class AddNumbersImpl implements AddNumbers {

    public int add(int number1, int number2) throws AddNumbersFault {
        return impl(number1, number2);
    }

    public int add2(int number1, int number2) throws AddNumbersFault {
        return impl(number1, number2);
    }

    public int add3(int number1, int number2) throws AddNumbersFault {
        return impl(number1, number2);
    }

    int impl(int number1, int number2) throws AddNumbersFault {
        if (number1 < 0 || number2 < 0) {
            throw new AddNumbersFault("Negative numbers can't be added!",
                                      "Numbers: " + number1 + ", " + number2);
        }
        return number1 + number2;
    }
}
```

The created AddNumbersImpl.java is saved in the c:\temp\jaxws\works\addressing\server\src\com\sample\ directory with the UTF-8 format.

Also create an exception class com.sample.AddNumbersFault thrown in the com.sample.AddNumbersImpl class.

The following is an example for creating an exception class:

```
package com.sample;

public class AddNumbersFault extends Exception {

    String detail;

    public AddNumbersFault(String message, String detail) {
        super(message);
        this.detail = detail;
    }

    public String getDetail() {
        return detail;
    }
}
```

The created AddNumbersFault.java is saved in the c:\temp\jaxws\works\addressing\server\src\com\sample\ directory with the UTF-8 format.

33.3.2 Compiling Web Services Implementation Class

Compile the Web Services Implementation Class, by executing the javac command. For details on the javac command, see the *JDK documentation*.

The following is the execution example of the javac command.

```
> cd c:\temp\jaxws\works\streaming\server\
> mkdir WEB-INF\classes\
> javac -encoding UTF-8 -cp "%COSMINEXUS_HOME%\jaxws\lib\cjjaxws.jar;%COSMINEXUS_HOME%
\CC\client\lib\j2ee-javax.jar;%COSMINEXUS_HOME%\jaxp\lib\csmjaxb.jar;%COSMINEXUS_HOME%
\jaxp\lib\csmjaxp.jar;%COSMINEXUS_HOME%\jaxp\lib\csmstax.jar;%HNTRLIB2_HOME%\classes
\hntrlib2j.jar;%HNTRLIB2_HOME%\classes\hntrlibMj.jar" -d WEB-INF\classes\ -s src src
\com\sample\AddNumbers.java src\com\sample\AddNumbersImpl.java src\com\sample
\AddNumbersFault.java
```

The compiled classes are output to the `c:\temp\jaxws\works\addressing\server\WEB-INF\classes\com\sample\ directory`, if the `javac` command successfully ends. Executing the `cjwsgen` command for the compiled Web Services Implementation Classes enables you to check errors in advance. For details on the `cjwsgen` command, see *14.3 cjwsgen command* and for details on error checking, see *10.23 (1) Using the cjwsgen command for checking errors*.

33.3.3 Creating web.xml

Create `web.xml` that is required as a WAR file component.

The following is an example for creating `web.xml`:

```
<<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/
javaee/web-app_3_0.xsd">
<description>Sample web service
&quot;streaming_dynamic_generate&quot;</description>
  <display-name>Sample_web_service__streaming_dynamic_generate</display-name>
addressing</display-name>
  <listener>
    <listener-class>


com.cosminexus.xml.ws.transport.http.servlet.WSServletContextListener
    </listener-class>
  </listener>
  <servlet>
    <description>Endpoint servlet for Cosminexus JAX-WS</description>
    <display-name>Endpoint_servlet_for_Cosminexus_JAX_WS</display-name>
    <servlet-name>CosminexusJaxwsServlet</servlet-name>
    <servlet-class>
      com.cosminexus.xml.ws.transport.http.servlet.WSServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>CosminexusJaxwsServlet</servlet-name>
    <url-pattern>/AddNumbersImplService</url-pattern>
  </servlet-mapping>
  <session-config>
    <session-timeout>60</session-timeout>
  </session-config>
</web-app>
```

When creating `web.xml` of version 2.5, specify `2.5` in the `version` attribute of the `web-app` element and specify `http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd` as the second location information in the `xsd:schemaLocation` attribute.

The created `web.xml` is saved in the `c:\temp\jaxws\works\addressing\server\WEB-INF\` directory with the UTF-8 format. For details about the `web.xml` settings, see the section *3.4 Creating web.xml*.

33.3.4 Creating application.xml

Create `application.xml` that is required as an EAR file component.

The following is an example for creating `application.xml`. Note that no items are set up in `application.xml` as the Web Service.

```
<?xml version="1.0" encoding="UTF-8"?>
<application version="6" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/
javaee/application_6.xsd">

  <description>Sample application &amp;quot;streaming_dynamic_generate&amp;quot;</description>
  <display-name>Sample_application_streaming_dynamic_generate</display-name>
_addressing</display-name>
  <module>
    <web>
<web-uri>streaming_dynamic_generate.war</web-uri>
<context-root>streaming_dynamic_generate</context-root>
    </web>
  </module>
</application>
```

When creating `web.xml` of version 5, specify `5` in the `version` attribute of the `application` element and specify `http://java.sun.com/xml/ns/javaee/application_5.xsd` as the second location information in the `xsd:schemaLocation` attribute.

The created `application.xml` is saved in the `c:\temp\jaxws\works\addressing\server\META-INF\` directory with the UTF-8 format. For notes on creating `application.xml`, see *5.2.2 Precautions for editing application.xml* in the *uCosminexus Application Server Application Development Guide*.

33.3.5 Creating EAR files

Use the `jar` command to create an EAR file containing the created files.

The following is an example for creating an EAR file:

```
> cd c:\temp\jaxws\works\addressing\server\
> jar cvf addressing_dynamic_generate.war .\WEB-INF
> jar cvf
addressing_dynamic_generate.ear .\addressing_dynamic_generate.war .\META-INF
\application.xml
```

If the `jar` command is terminated successfully, `streaming_dynamic_generate.ear` is created in the `c:\temp\jaxws\works\addressing\server\` directory. For details about the `jar` command, see the *JDK documentation*.

33.4 Examples of deployment and startup (Starting from SEI and streaming)

This section describes the examples for the deployment and the startup starting from SEI, using the streaming functionality.

33.4.1 Deploying EAR files

Use the `cjimportapp` command to deploy the created EAR file on the J2EE server.

The following is an example of deployment:

```
> cd c:\temp\jaxws\works\addressing\server\
> "%COSMINEXUS_HOME%\CC\admin\bin\cjimportapp" jaxwsserver -nameserver
corbaname::testserver:900 -f addressing_dynamic_generate.ear
```

For the `cjimportapp` command, see *cjimportapp (Importing J2EE applications)* in the *uCosminexus Application Server Command Reference Guide*.

For details on how to deploy (import) J2EE applications by using the management portal, see *12.3.3 Importing J2EE applications* in the *uCosminexus Application Server Management Portal User Guide*.

33.4.2 Starting Web Services

Use the `cjstartapp` command to start Web Services.

The following is an example for starting a Web Service:

```
> cd c:\temp\jaxws\works\addressing\server\
> "%COSMINEXUS_HOME%\CC\admin\bin\cjstartapp" jaxwsserver -nameserver
corbaname::testserver:900 -name Sample_application_addressing_dynamic_generate
```

For the `cjstartapp` command, see *cjstartapp (Starting J2EE applications)* in the *uCosminexus Application Server Command Reference Guide.*

For the method to start J2EE applications by using the management portal, see *12.3.1 Starting J2EE applications* in the *uCosminexus Application Server Management Portal User Guide*.

33.5 Examples of developing the Web Service client (Starting from SEI and streaming)

This section describes the examples for the development of Web Service clients (using the addressing functionality) starting from SEI.

33.5.1 Generating a service class

If you execute the `cjwsimport` command, the Java source such as a service class that is required for developing a Web Service client is generated. For details about the `cjwsimport` command, see the section *14.1 cjwsimport command*.

The following is an example of the execution of the `cjwsimport` command:

```
> cd c:\temp\jaxws\works\addressing\client\
> mkdir src\
> mkdir classes\
> "%COSMINEXUS_HOME%\jaxws\bin\cjwsimport.bat" -s src -d classes http://webhost:8085/
addressing_dynamic_generate/AddNumbersImplService?wsdl
```

If the `cjwsimport` command is terminated successfully, the Java source is generated in the `c:\temp\jaxws\works\addressing\client\src\com\sample\` directory.

The following table lists the products:

Table 33-3: Products during service class generation (Starting from SEI and addressing)

File name	Explanation
`Add.java`	This is the `JavaBean` class corresponding to the type referenced by the `wrapper` element of the request message of `operation` in the WSDL definition.
`AddResponse.java`	This is the `JavaBean` class corresponding to the type referenced by the `wrapper` element of the response message of `operation` in the WSDL definition.
`Add2.java`	This is the `JavaBean` class corresponding to the type referenced by the `wrapper` element of the request message of `operation` in the WSDL definition.
`Add2Response.java`	This is the `JavaBean` class corresponding to the type referenced by the `wrapper` element of the response message of `operation` in the WSDL definition.
`Add3.java`	This is the `JavaBean` class corresponding to the type referenced by the `wrapper` element of the request message of `operation` in the WSDL definition.
`Add3Response.java`	This is the `JavaBean` class corresponding to the type referenced by the `wrapper` element of the response message of `operation` in the WSDL definition.
`ObjectFactory.java`	This is the `ObjectFactory` class of the JAXB 2.2 specifications.
`package-info.java`	This is the `package-info.java` file.
`AddNumbers.java`	This is the Service Endpoint Interface (SEI) corresponding to `service` in the WSDL definition.
`AddNumbersImplService.java`	This is a service class.

File name	Explanation
AddNumbersFault.java	This is the JavaBean class corresponding to AddNumbersFault in the WSDL definition.
AddNumbersFault_Exception.java	This is the wrapper exception class of the fault bean.

The file names Add, AddNumbersImpl, and AddNumbersImplService change as per the coding of the local names in operation, port type, and service. For details about mapping of the local names in operation, port type, and service, see the following sections:

- *15.1.2 Mapping a port type to a SEI name*
- *15.1.3 Mapping the operations to the method names*
- *15.1.4 Mapping a message part to a parameter and return value (For the wrapper style)*
- *15.1.5 Mapping the message part to the parameter and return value (For non-wrapper style)*

33.5.2 Creating an implementation class for the Web Service client

Create an implementation class for the Web Service client that uses the Web Services.

The following is an example for creating a Web Service client that invokes the Web Service thrice:

```
package com.sample.client;

import javax.xml.namespace.QName;
import javax.xml.ws.soap.AddressingFeature;
import javax.xml.ws.wsaddressing.W3CEndpointReference;
import javax.xml.ws.wsaddressing.W3CEndpointReferenceBuilder;

import com.sample.AddNumbers;
import com.sample.AddNumbersImplService;
import com.sample.AddNumbersFault_Exception;

public class TestClient {

    int number1 = 10;
    int number2 = 10;
    int negativeNumber = -10;

    public static void main(String[] args) {
        TestClient client = new TestClient();

        client.existActionAnnotation1();
        client.existActionAnnotation2();
        client.notExistActionAnnotation();
        client.existFaultActionAnnotation();
        client.notExistFaultActionAnnotation();
    }

    public void existActionAnnotation1() {
        System.out.println("existActionAnnotation1");
        try {
            AddressingFeature feature = new AddressingFeature();

            AddNumbersImplService service = new AddNumbersImplService();
            AddNumbers stub = service.getAddNumbersImplPort(feature);
            int result = stub.add(number1, number2);
            assert result == 20;
        } catch (Exception ex) {
            ex.printStackTrace();
            assert false;
        }
    }

    public void existActionAnnotation2() {
        System.out.println("existActionAnnotation2");
        try {
            AddressingFeature feature = new AddressingFeature();
            W3CEndpointReferenceBuilder eprBuilder = new
W3CEndpointReferenceBuilder();
            eprBuilder.address("http://webhost:8085/addressing/
AddNumbersImplService");
```

```
            eprBuilder.serviceName(new QName("http://sample.com/",
"AddNumbersImplService"));
            eprBuilder.endpointName(new QName("http://sample.com/",
"AddNumbersImplPort"));
            W3CEndpointReference epr = eprBuilder.build();

            AddNumbersImplService service = new AddNumbersImplService();
            AddNumbers stub = service.getPort(epr, AddNumbers.class, feature);
            int result = stub.add(number1, number2);
            assert result == 20;
        } catch (Exception ex) {
            ex.printStackTrace();
            assert false;
        }
    }

    public void notExistActionAnnotation() {
        System.out.println("notExistActionAnnotation");
        try {
            AddressingFeature feature = new AddressingFeature();

            AddNumbersImplService service = new AddNumbersImplService();
            AddNumbers stub = service.getAddNumbersImplPort(feature);
            int result = stub.add2(number1, number2);
            assert result == 20;
        } catch (Exception ex) {
            ex.printStackTrace();
            assert false;
        }
    }

    public void existFaultActionAnnotation() {
        System.out.println("existFaultActionAnnotation");
        try {
            AddressingFeature feature = new AddressingFeature();

            AddNumbersImplService service = new AddNumbersImplService();
            AddNumbers stub = service.getAddNumbersImplPort(feature);
            stub.add3(negativeNumber, number2);
            assert false;
        } catch (AddNumbersFault_Exception e) {
            System.out.println("This is expected exception");
        } catch (Exception e) {
            e.printStackTrace();
            assert false;
        }
    }

    public void notExistFaultActionAnnotation() {
        System.out.println("notExistFaultActionAnnotation");
        try {
            AddressingFeature feature = new AddressingFeature();

            AddNumbersImplService service = new AddNumbersImplService();
            AddNumbers stub = service.getAddNumbersImplPort(feature);
            stub.add(negativeNumber, number2);
            assert false;
        } catch (AddNumbersFault_Exception ex) {
            System.out.println("This is expected exception");
        } catch (Exception e) {
            e.printStackTrace();
            assert false;
        }
    }
}
```

The created `TestClient.java` is saved in the `c:\temp\jaxws\works\addressing\client\src\com\sample\client\` directory with the UTF-8 format.

Note that `com.sample`, `AddNumbers`, `AddNumbersImplService`, `AddNumbersImplPort`, `add`, `add2`, and `add3` differ as per the package names, class names, and the method names in the classes of the generated Java source. For developing Web Services with different configurations, you must review the coding of the package names, class names, and method names in the classes.

33.5.3 Compiling the implementation class for the Web Service client

Use the `javac` command to compile the created Web Service client.

The following is an example of compilation:

```
> cd c:\temp\jaxws\works\addressing\client\
> javac -encoding UTF-8 -cp "%COSMINEXUS_HOME%\jaxws\lib\cjjaxws.jar;%COSMINEXUS_HOME%
\CC\client\lib\j2ee-javax.jar;%COSMINEXUS_HOME%\jaxp\lib\csmjaxb.jar;.\classes" -d .
\classes src\com\sample\client\TestClient.java
```

If the `javac` command is terminated successfully, `TestClient.class` is generated in the `c:\temp\jaxws\works\addressing\client\classes\com\sample\client\` directory. For details about the `javac` command, see the *JDK documentation*.

33.6 Examples of executing the Web Services (Starting from SEI and streaming)

This section describes the examples for executing Web Service clients (when using the streaming functionality) starting from SEI.

33.6.1 Creating option definition files for Java applications

Create option definition files for Java applications (`usrconf.cfg`) required for executing a Web Service.

The following is an example for creating option definition files for Java applications:

```
add.class.path=Cosminexus-installation-directory\jaxws\lib\cjjaxws.jar
add.class.path=.\classes
ejb.client.log.directory=logs
add.jvm.arg=-Dcosminexus.home=Cosminexus-installation-directory
add.jvm.arg=-Dejbserver.server.prf.PRFID=<PRF ID>
```

For *Cosminexus-installation-directory*, use the absolute path to specify the path where Cosminexus is installed.

The created option definition file for Java applications is stored in the `c:\temp\jaxws\works\streaming\client\` directory. For the option definition file for Java applications, see *14.2 usrconf.cfg (Option definition file for Java applications)* in the *uCosminexus Application Server Definition Reference Guide*.

33.6.2 Creating user property files for Java applications

Create user property files for Java applications required for executing Web Services.

Because the settings are not specially changed, create an empty file named `usrconf.properties` in the `c:\temp\jaxws\works\addressing\client\` directory. For the user property file for Java applications, see *14.3 usrconf.properties (User property file for Java applications)* in the *uCosminexus Application Server Definition Reference Guide*.

33.6.3 Executing the Web Service client

Use the `cjclstartap` command to execute Web Service clients.

The following is an example for executing a Web Service client:

```
> cd c:\temp\jaxws\works\addressing\client\
> "%COSMINEXUS_HOME%\CC\client\bin\cjclstartap" com.sample.client.TestClient
```

If the `cjclstartap` command is terminated successfully, the result of Web Service client execution is displayed. The following is an example for displaying the execution results:

```
KDJE40053-I The cjclstartap command will now start. (directory for the user
definition file = c:\temp\jaxws\works\streaming\client, PID = 2636)
existActionAnnotation1
existActionAnnotation2
notExistActionAnnotation
existFaultActionAnnotation
This is expected exception
notExistFaultActionAnnotation
This is expected exception
KDJE40054-I The cjclstartap command was stopped. (PID = 2636, exit status = 0)
```

The part in italics changes according to the execution timing and the environment.

For the `cjclstartap` command, see *cjclstartap (Starting Java applications)* in the *uCosminexus Application Server Command Reference Guide*.

34 WS-RM 1.2 Functionality

This chapter describes the WS-RM 1.2 functionality (Web Service Reliable Messaging functionality).

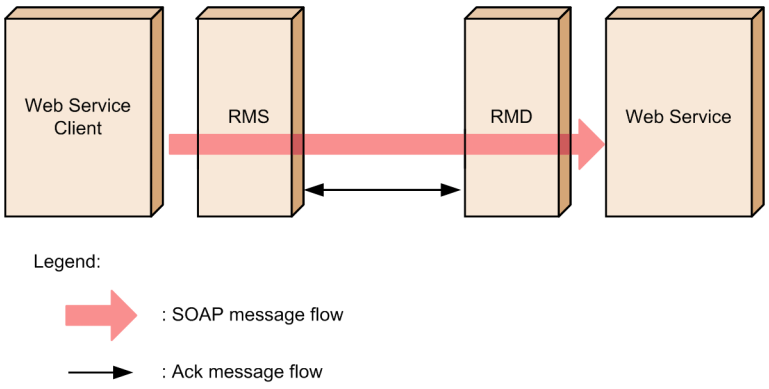
34.1 What is the WS-RM 1.2 functionality

The WS-RM 1.2 functionality is the functionality for reliably exchanging (sending and receiving) SOAP Messages between Web Services and Web Service clients through RMD (Reliable Messaging Destination) andRMS (Reliable Messaging Source).

This functionality monitors the sending and receiving of messages between RMS and RMD by sending an Ack message in the background when the recipient receives a message. As and when required, the WS-RM 1.2 functionality automatically re-sends the messages and removes the duplicate messages, thus increasing the reliability of the communication.

The following figure shows the flow of sending and receiving SOAP Messages by using the WS-RM 1.2 functionality.

Figure 34－1: Flow of sending and receiving SOAP Messages by using the WS-RM 1.2 functionality

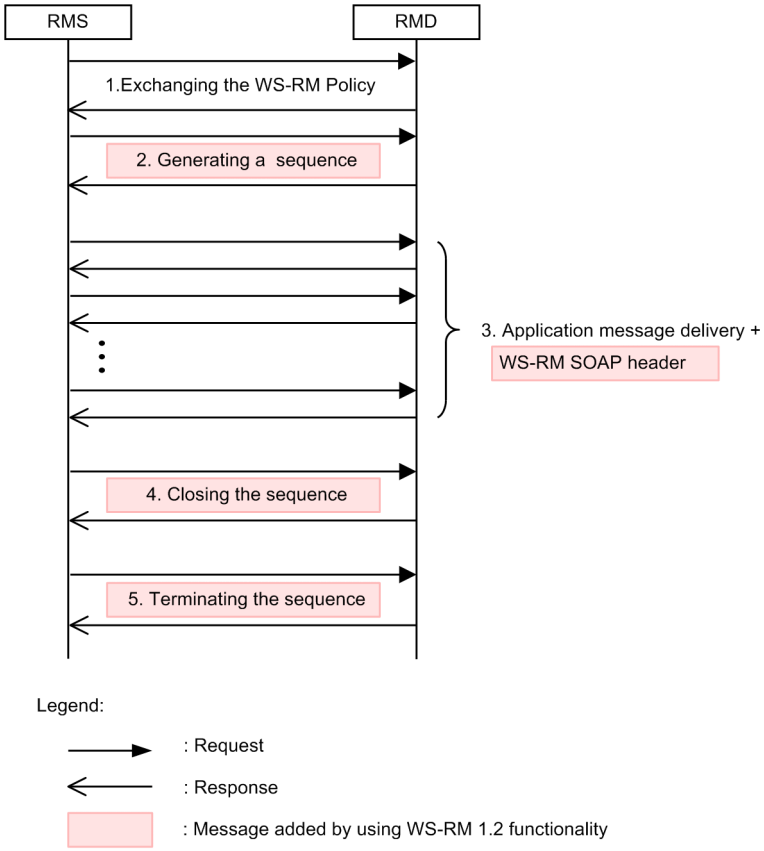


You cannot use the `Dispatch` interface and `Provider` interface.

34.2 Message flow when the WS-RM 1.2 functionality is used

The following figure shows the message flow when the WS-RM 1.2 functionality is used:

Figure 34-2: Message flow when the WS-RM 1.2 functionality is used



1. Exchanging the WS-RM Policy

   Exchange the availability and the WS-RM Policy of the WS-RM 1.2 functionality between the message source and the destination. Exchange the WS-RM Policy by coding the WS-RM Policy specifications in the WSDL file.

2. Generating a sequence

   Before sending the application message, generate a sequence and share RMS and RMD. A sequence is a context related to a set of a series of application messages that are sent by using the WS-RM 1.2 functionality. The sequence generation processing is automatically executed when the Web Service method is invoked for the first time for a port object.

3. Sending application messages

   After the sequence is generated, RMS sends the application messages to RMD. A sequence identifier and a message number are added to the application messages to be sent.

4. Closing the sequence

   When the sending of the application messages is completed, RMS can close the sequence. Once the sequence is closed, RMS cannot send and receive new application messages. Even if the sequence is closed, the related resources are saved and not destroyed. The sequence closing operation is executed when the `close` method is invoked on the Web Service client side.

5. Terminating the sequence

   When the sending of the application messages is completed, RMS terminates the sequence. When the sequence is terminated, the related resources are destroyed. To generate the sequence again, reacquire the port object and invoke the Web Service method.

Hereafter, the messages related to the sequence generation, the sequence closure, and the sequence termination are collectively called *Sequence life cycle messages*.

34.3 Delivery assurance functionality of WS-RM 1.2

Delivery assurance is the functionality for resending, removing the duplication, and controlling the order of SOAP Messages to send and receive the SOAP Messages reliably.

The following table lists the types of Delivery assuranceand the support range in the Cosminexus WS-RM 1.2 functionality.

Table 34-1: Types of Delivery assurance and the support range in the Cosminexus WS-RM 1.2 functionality

No.	Type	Operation	RMS processing	RMD processing	Support
1	`AtLeastOnce`	Deliver at least once	Resend	--	N
2	`AtMostOnce`	Deliver without duplication	--	Remove duplication	N
3	`ExactlyOnce`	Deliver only once	Resend	Remove duplication	Y
4	`InOrder`	Deliver as per order	--	Order control	N

Legend:

Y: Can be used.

N: Cannot be used.

--: Not applicable

(1) Resending

When using the WS-RM 1.2 functionality, if an application message is not delivered to the destination, the message is automatically resent from RMS for the maximum number of times as set in the WS-RM Policy for resending the application message. If the application message is not delivered even after resending it for the specified number of times, `javax.xml.ws.WebServiceException` is returned to the client application.

The following are the conditions for resending the messages:

- If a timeout occurs due to no response even after the read timeout value or the connection timeout value set in the JAX-WS Operation definition file or the request context for the client socket is exceeded.
- If the connection with the server is unexpectedly disconnected and no response could be received.
- If an Ack message with the blank `Body` element is received as a response to the application message or automatically resent application message.
- If HTTP status code 202 is received as a response to the application message or automatically resent application message.
- If the HTTP status code 2012, 300 to 399, 400 to 499, or 500 to 599 is received as a response to the application message that is automatically resent.

(2) Removing duplication

When you receive a message that is same as the message received earlier, it is not delivered to the destination and is destroyed at RMD. The application is not invoked.

The following messages are returned to the client, if duplicate messages are received:

- If you receivea request that is duplicating the request for which the response is already returned, contents same as theresponseoncereturnedare returned.
- While processing a request, if the same request is received again, an Ack message with a blank `Body` element is returned to the client. However, if there is no Ack to return, HTTP status code 202 is returned.

34.4 How to add the WS-RM Policy

Enable the WS-RM 1.2 functionality by adding the WS-RM Policy to the WSDL file. The following items are to be added:

1. Defining the name space prefix to be used in the WS-RM Policy
2. Defining a policy as the child element of the `wsdl:definitions` element
3. Referencing a policy in the child element of the `wsdl:binding` element

The following example explains how to add these items:

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="TestJaxWsService"
  <!--Partially omitted -->

<!-- 1. Define the name space prefix to be used in the WS-RM Policy -->
  xmlns:wsp="http://www.w3.org/ns/ws-policy"
  xmlns:wsrmp="http://docs.oasis-open.org/ws-rx/wsrmp/200702"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd"
  xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl">

<!-- 2. Define a policy as the child element of the wsdl:definitions element  -->

  <wsp:Policy wsu:Id="WSRM_policy">
    <wsp:ExactlyOne>
      <wsp:All>
        <wsrmp:RMAssertion>
          <wsp:Policy>
            <wsrmp:DeliveryAssurance>
              <wsp:Policy>
                <wsrmp:ExactlyOnce/>
              </wsp:Policy>
            </wsrmp:DeliveryAssurance>
          </wsp:Policy>
        </wsrmp:RMAssertion>
        <wsaw:UsingAddressing/>
        <!- Other settings -->
      </wsp:All>
    </wsp:ExactlyOne>
  </wsp:Policy>

  <wsdl:types>

<!-- Partially omitted -->

<!-- 3. Reference a policy in the child element of the wsdl:binding element -->
  <wsdl:binding name="testJaxWsBinding" type="tns:TestJaxWs">
    <wsp:PolicyReference URI="#WSRM_policy"/>

<!-- The rest is omitted -->
```

The *bold* values are the definitions or references of the name space prefix definition, WS-RM Policy definition, or WS-RM Policy.

Code the definitions of the WS-RM Policy as a child element of the `wsdl:definitions` element.

You can omit the elements under the `wsrmp:DeliveryAssurance` element. Code the elements under the `wsrmp:DeliveryAssurance` element and those other than "<!--Other settings-->" as is. Add settings for "<!--Other settings-->" as and when required.

For the properties to be added in a WSDL as "<!--Other settings-->", see the section *23.4 Settings by WS-Policy*.

35 Example of the Development Starting from WSDL (using WS-RM 1.2)

This chapter describes the examples for the development of Web Services starting from WSDL, by using the WS-RM 1.2 functionality.

35.1 Configuration of the development example (Starting from WSDL/WS-RM 1.2)

This section describes an example of developing a Web Service starting from WSDL.

The following table lists the configuration of the Web Service to be developed:

Table 35-1: Web Service configuration (Starting from WSDL)

No.	Item		Value
1	Name of the J2EE Sever to be deployed		`jaxwsserver`
2	The host name and the port number of the Web Server		`webhost:8085`
3	URL of the Naming Server		`corbaname::testserver:900`
4	Context route		`wsrm`
5	Style		`document/literal/wrapped`
6	Namespace URI		`http://example.com/sample`
7	Port type	No.	`1`
8		Local name	`TestJaxWs`
9	Operation	No.	`1`
10		Local name	`jaxWsTest1`
11	Service	No.	`1`
12		Local name	`TestJaxWsService`
13	Port	No.	`1`
14		Local name	`testJaxWs`
15	WSDL file name		`input.wsdl`

The following table lists the configuration of the current directory used in the Web Service development:

Table 35-2: Configuration of the current directory (Starting from WSDL)

Directory	Explanation
`c:\temp\jaxws\works\wsrm`	This is the current directory.
`server\`	This is used in the Web Service development.
`META-INF\`	Corresponds to the META-INF directory of the EAR files.
`application.xml`	Created in *35.3.7 Creating an application.xml file*.
`src\`	Stores the source files (`*.java`) of the Web Service. Used in *35.3.3 Creating SEI* and *35.3.5 Compiling Web Services Implementation Class*.
`WEB-INF\`	Corresponds to the WEB-INF directory of the WAR files.
`web.xml`	Created in *35.3.6 Creating web.xml*.
`classes\`	Stores the compiled class files (`*.class`).
`wsdl\`	Stores the created wsdl.
`temporary\`	Stores the temporary files when creating a WSDL based on the WSDL that is coded and converted by using Java. The creation of this directory is optional.
`src\`	

Directory	Explanation
classes\	Stores the temporary files when creating a WSDL based on the WSDL that is coded and converted by using Java. The creation of this directory is optional.
wsrm.ear	Created in *35.3.8 Creating EAR files*.
wsrm.war	
client\	This is used in the Web Service client development.
src\	Stores the source files (`*.java`) of the Web Service client. Used in *35.5.1 Generating a service class* and *35.5.2 Creating Implementation Class for the Web Services client*.
classes\	Stores the compiled class files (`*.class`). Used in *35.5.4 Compiling Implementation Class for the Web Services client*.
usrconf.cfg	Created in *35.6.1 Creating an option definition file for Java applications*.
usrconf.properties	Created in *35.6.2 Creating a user property file for Java applications*.

Change the current directory path according to the development environment.

Note that the description below uses the directory and file names mentioned in the above table. The values in *bold* used in the command execution examples or Java source in this document are the values that are specified or generated in the examples used within this document. Use appropriate values according to the environment you build.

Furthermore, in the development example described in this chapter, the Web Service and the Web Service client are developed in the same environment; however, they can be developed in different environments as well. To develop the Web Service and the Web Service client in separate environments, you must change the current directory path according to the respective environment.

35.2 Flow of the development example (Starting from WSDL/WS-RM 1.2)

The procedure for development and execution described in the development examples in this chapter are as follows:

Developing a Web Service

1. Create a WSDL file (*35.3.1*)
2. Add WS-RM Policy in the WSDL file (*35.3.2*)
3. Execute the `cjwsimport` command and generate SEI (*35.3.3*)
4. Create Web Services Implementation Class (*35.3.4*)
5. Compile Web Services Implementation Class (*35.3.5*)
6. Create a `web.xml` file (*35.3.6*)
7. Create an `application.xml` file (*35.3.7*)
8. Create an EAR file (*35.3.8*)

Deploying and starting the service

1. Deploy the EAR file (*35.4.1*)
2. Start the Web Services (*35.4.2*)

Developing the Web Service client

1. Execute the `cjwsimport` command and generate a service class (*35.5.1*)
2. Create Implementation Class for the Web Services client (*35.5.2*)
3. Add the sequence termination processing in Implementation Class for the Web Services client (*35.5.3*)
4. Compile Implementation Class for the Web Services client (*35.5.4*)

Executing the Web Service

1. Create an option definition file for Java applications (*35.6.1*)
2. Create a user property file for Java applications (*35.6.2*)
3. Execute the Web Services client (*35.6.3*)

35.3 Examples of developing a Web Service (Starting from WSDL/WS-RM 1.2)

This section describes an example of developing a Web Service starting from WSDL.

35.3.1 Creating a WSDL file

Create a WSDL file and define the meta-information of the Web Service. Define the WSDL definitions within the range specified in the following specifications:

- WSDL 1.1 specifications
  For the support range, see *20.1 Support range of WSDL 1.1 specifications*.
- XML Schema specifications
  For the support range, see the *uCosminexus Application Server XML Processor User Guide*.
- WS-I Basic Profile 1.1 specifications

There are two methods for creating a WSDL file. a) Create a new WSDL file or b) Create a WSDL file by converting the Java source.

(1) Creating a new WSDL file

Create a WSDL file (`input.wsdl`). The created WSDL file is stored in the `c:\temp\jaxws\works\wsrm\server\WEB-INF\wsdl\` directory in the UTF-8 format.

The following is an example for creating a WSDL file for SOAP 1.1:

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="TestJaxWsService"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:tns="http://example.com/sample"
  targetNamespace="http://example.com/sample">

  <wsdl:types>
    <xsd:schema targetNamespace="http://example.com/sample">
      <!-- wrapper element of the request message -->
      <xsd:element name="jaxWsTest1" type="tns:jaxWsTest1"/>

      <!-- wrapper element of the response message -->
      <xsd:element name="jaxWsTest1Response" type="tns:jaxWsTest1Response"/>

      <!-- wrapper element of the fault message -->
      <xsd:element name="UserDefinedFault" type="tns:UserDefinedFault"/>

      <!-- Type referenced by the wrapper element of the request message -->
      <xsd:complexType name="jaxWsTest1">
        <xsd:sequence>
          <xsd:element name="information" type="xsd:string"/>
          <xsd:element name="count" type="xsd:int"/>
        </xsd:sequence>
      </xsd:complexType>


      <!-- Type referenced by the wrapper element of the response message -->
      <xsd:complexType name="jaxWsTest1Response">
        <xsd:sequence>
          <xsd:element name="return" type="xsd:string"/>
        </xsd:sequence>
      </xsd:complexType>

      <!-- Type referenced by the wrapper element of the fault message -->
      <xsd:complexType name="UserDefinedFault">
        <xsd:sequence>
          <xsd:element name="additionalInfo" type="xsd:int"/>
          <xsd:element name="detail" type="xsd:string"/>
          <xsd:element name="message" type="xsd:string"/>
        </xsd:sequence>
```

```
      </xsd:complexType>
    </xsd:schema>
  </wsdl:types>

  <!-- Request message -->
  <wsdl:message name="jaxWsTest1Request">
    <wsdl:part name="inputParameters" element="tns:jaxWsTest1"/>
  </wsdl:message>

  <!-- Response message -->
  <wsdl:message name="jaxWsTest1Response">
    <wsdl:part name="outputParameters" element="tns:jaxWsTest1Response"/>
  </wsdl:message>

  <!-- Fault message -->
  <wsdl:message name="UserDefinedException">
    <wsdl:part name="fault" element="tns:UserDefinedFault"/>
  </wsdl:message>

  <!-- Port type -->
  <wsdl:portType name="TestJaxWs">
    <!-- Operation -->
    <wsdl:operation name="jaxWsTest1">
      <wsdl:input message="tns:jaxWsTest1Request"/>
      <wsdl:output message="tns:jaxWsTest1Response"/>
      <wsdl:fault name="UserDefinedFault"
        message="tns:UserDefinedException"/>
    </wsdl:operation>
  </wsdl:portType>

<!-- Binding (SOAP 1.1/HTTP binding) -->
<wsdl:binding name="testJaxWsBinding" type="tns:TestJaxWs">
  <!-- document/literal/wrapped -->
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <!-- Operation -->
    <wsdl:operation name="jaxWsTest1">
      <soap:operation/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
      <wsdl:fault name="UserDefinedFault">
        <soap:fault name="UserDefinedFault" use="literal"/>
      </wsdl:fault>
    </wsdl:operation>
  </wsdl:binding>

  <!-- Service -->
  <wsdl:service name="TestJaxWsService">
    <!-- Port -->
    <wsdl:port name="testJaxWs" binding="tns:testJaxWsBinding">
      <soap:address location="http://webhost:8085/wsrm/TestJaxWsService"/>
    </wsdl:port>
  </wsdl:service>

</wsdl:definitions>
```

The following is an example for creating a WSDL file for SOAP 1.2:

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="TestJaxWsService"
  xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:tns="http://example.com/sample"
  targetNamespace="http://example.com/sample">

  <wsdl:types>
    <xsd:schema targetNamespace="http://example.com/sample">
      <!-- wrapper element of the request message -->
      <xsd:element name="jaxWsTest1" type="tns:jaxWsTest1"/>

      <!-- wrapper element of the response message -->
      <xsd:element name="jaxWsTest1Response" type="tns:jaxWsTest1Response"/>

      <!-- wrapper element of the fault message -->
      <xsd:element name="UserDefinedFault" type="tns:UserDefinedFault"/>

      <!-- wrapper element of the fault message -->
```

```
      <xsd:complexType name="jaxWsTest1">
        <xsd:sequence>
          <xsd:element name="information" type="xsd:string"/>
          <xsd:element name="count" type="xsd:int"/>
        </xsd:sequence>
      </xsd:complexType>

      <!-- Type referenced by the wrapper element of the response message -->
      <xsd:complexType name="jaxWsTest1Response">
        <xsd:sequence>
          <xsd:element name="return" type="xsd:string"/>
        </xsd:sequence>
      </xsd:complexType>

      <!-- Type referenced by the wrapper element of the fault message -->
      <xsd:complexType name="UserDefinedFault">
        <xsd:sequence>
          <xsd:element name="additionalInfo" type="xsd:int"/>
          <xsd:element name="detail" type="xsd:string"/>
          <xsd:element name="message" type="xsd:string"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:schema>
  </wsdl:types>

  <!-- Request message -->
  <wsdl:message name="jaxWsTest1Request">
    <wsdl:part name="inputParameters" element="tns:jaxWsTest1"/>
  </wsdl:message>

  <!-- Response message -->
  <wsdl:message name="jaxWsTest1Response">
    <wsdl:part name="outputParameters" element="tns:jaxWsTest1Response"/>
  </wsdl:message>

  <!-- Fault message -->
  <wsdl:message name="UserDefinedException">
    <wsdl:part name="fault" element="tns:UserDefinedFault"/>
  </wsdl:message>

  <!-- Port type -->

  <wsdl:portType name="TestJaxWs">
    <!-- Operation -->
    <wsdl:operation name="jaxWsTest1">
      <wsdl:input message="tns:jaxWsTest1Request"/>
      <wsdl:output message="tns:jaxWsTest1Response"/>
      <wsdl:fault name="UserDefinedFault"
        message="tns:UserDefinedException"/>
    </wsdl:operation>
  </wsdl:portType>

<!-- Binding (SOAP 1.2/HTTP binding) -->
<wsdl:binding name="testJaxWsBinding" type="tns:TestJaxWs">
  <!-- document/literal/wrapped -->
  <soap12:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <!-- Operation -->
    <wsdl:operation name="jaxWsTest1">
      <soap12:operation/>
      <wsdl:input>
        <soap12:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap12:body use="literal"/>
      </wsdl:output>
      <wsdl:fault name="UserDefinedFault">
        <soap12:fault name="UserDefinedFault" use="literal"/>
      </wsdl:fault>
    </wsdl:operation>
  </wsdl:binding>

  <!-- Service -->
  <wsdl:service name="TestJaxWsService">
    <!-- Port -->
    <wsdl:port name="testJaxWs" binding="tns:testJaxWsBinding">
      <soap12:address location="http://webhost:8085/wsrm/TestJaxWsService"/>
    </wsdl:port>
  </wsdl:service>

</wsdl:definitions>
```

(2) Creating a WSDL file by converting the Java source

Create a Web Service Implementation Class and an exception class to be temporarily implemented for the WSDL conversion, and execute the WSDL generation functionality of the `cjwsgen` command to create a WSDL file from the already compiled Java source. Annotate the created class by using the `javax.jws.WebService` annotation. When using SOAP 1.2, annotate with the `javax.xml.ws.BindingType` annotation in which SOAP 1.2 is specified additionally. You need not implement any method.

The source code for the temporarily implemented Web Service Implementation Class and `TestJaxWsService.wsdl` differs depending on whether you use SOAP 1.1 or SOAP 1.2.

The following is an example of the temporarily implemented Web Service Implementation Class when SOAP 1.1 is used.

```
package com.example.sample;

@javax.jws.WebService
public class TestJaxWsImpl {

    public String jaxWsTest1(String information, int count)
        throws UserDefinedException
    {
        // Need not be implemented
        return null;
    }

}
```

The following is an example of the temporarily implemented Web Service Implementation Class when SOAP 1.2 is used.

```
package com.example.sample;

@javax.jws.WebService
@javax.xml.ws.BindingType(javax.xml.ws.soap.SOAPBinding.SOAP12HTTP_BINDING)
public class TestJaxWsImpl {

    public String jaxWsTest1(String information, int count)
        throws UserDefinedException
    {
        // Need not be implemented
        return null;
    }

}
```

The following is an example of the temporarily implemented exception class:

```
package com.example.sample;

public class UserDefinedFault extends Exception{
    // Need not be implemented
    public int additionalInfo;
    public String detail;
    public String message;
}
```

Save and compile the created `TestJaxWsImpl.java` and `UserDefinedFault.java` in the `c:\temp\jaxws\works\wsrm\server\temporary\src\com\example\sample\` directory in the UTF-8 format.

The following is an example of compilation:

```
> cd c:\temp\jaxws\works\wsrm\server\
> mkdir .\temporary
> mkdir .\temporary\classes
> javac -encoding UTF-8 -cp "%COSMINEXUS_HOME%\jaxws\lib\cjjaxws.jar;%COSMINEXUS_HOME%
\CC\client\lib\j2ee-javax.jar" -d .\temporary\classes .\temporary\src\com\example
\sample\TestJaxWsImpl.java .\temporary\src\com\example\sample\UserDefinedFault.java
```

On successful compilation, the `TestJaxWsImpl.class` and `UserDefinedFault.class` files are generated in the `c:\temp\jaxws\works\wsrm\server\temporary\classes\com\example\sample\`

directory. Use these class files and create a WSDL file by using the WSDL generation functionality of the `cjwsgen` command.

The following is an example for executing the `cjwsgen` command:

```
> cd c:\temp\jaxws\works\wsrm\server\
> mkdir .\WEB-INF\wsdl\
> "%COSMINEXUS_HOME%\jaxws\bin\cjwsgen.bat" -r .\WEB-INF\wsdl -d .\temporary\classes -
cp .\temporary\classes com.example.sample.TestJaxWsImpl
```

On successful termination of the `cjwsgen` command, `TestJaxWsService.wsdl` and `TestJaxWsService_schema1.xsd` are generated in the `c:\temp\jaxws\works\wsrm\WEB-INF\wsdl\` directory. Delete the classes from the `c:\temp\jaxws\works\wsrm\temporary\classes\` directory.

You must partially modify the generated `TestJaxWsService.wsdl` and `TestJaxWsService_schema1.xsd`.

The following is an example for modifying `TestJaxWsService.wsdl` when SOAP 1.1 is used. The parts in *Italics* indicate modifications.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<definitions targetNamespace="http://example.com/sample" name="TestJaxWsImplService"
xmlns:tns="http://example.com/sample" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns="http://schemas.xmlsoap.org/
wsdl/">
  <types>
    <xsd:schema targetNamespace="http://example.com/sample">
      <xsd:include schemaLocation="TestJaxWsImplService_schema1.xsd"/>
    </xsd:schema>
  </types>
  <message name="jaxWsTest1">
    <part name="parameters" element="tns:jaxWsTest1"/>
  </message>
  <message name="jaxWsTest1Response">
    <part name="parameters" element="tns:jaxWsTest1Response"/>
  </message>
  <message name="UserDefinedFault">
    <part name="fault" element="tns:UserDefinedFault"/>
  </message>
  <portType name="TestJaxWs">
    <operation name="jaxWsTest1">
      <input message="tns:jaxWsTest1"/>
      <output message="tns:jaxWsTest1Response"/>
      <fault message="tns:UserDefinedFault" name="UserDefinedFault"/>
    </operation>
  </portType>
  <binding name="testJaxWsBinding" type="tns:TestJaxWs">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <operation name="jaxWsTest1">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
      <fault name="UserDefinedFault">
        <soap:fault name="UserDefinedFault" use="literal"/>
      </fault>
    </operation>
  </binding>
  <service name="TestJaxWsService">
    <port name="testJaxWs" binding="tns:testJaxWsBinding">
      <soap:address location="http://webhost:8085/wsrm/TestJaxWsService"/>
    </port>
  </service>
</definitions>
```

The following is an example for modifying `TestJaxWsService.wsdl` when SOAP 1.2 is used. The parts in *Italics* indicate modifications.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<definitions targetNamespace="http://example.com/sample" name="TestJaxWsImplService"
xmlns:tns="http://example.com/sample" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/" xmlns="http://
schemas.xmlsoap.org/wsdl/">
```

```
  <types>
    <xsd:schema targetNamespace="http://example.com/sample">
      <xsd:include schemaLocation="TestJaxWsImplService_schema1.xsd"/>
    </xsd:schema>
  </types>
  <message name="jaxWsTest1">
    <part name="parameters" element="tns:jaxWsTest1"/>
  </message>
  <message name="jaxWsTest1Response">
    <part name="parameters" element="tns:jaxWsTest1Response"/>
  </message>
  <message name="UserDefinedFault">
    <part name="fault" element="tns:UserDefinedFault"/>
  </message>
  <portType name="TestJaxWs">
    <operation name="jaxWsTest1">
      <input message="tns:jaxWsTest1"/>
      <output message="tns:jaxWsTest1Response"/>
      <fault message="tns:UserDefinedFault" name="UserDefinedFault"/>
    </operation>
  </portType>
  <binding name="testJaxWsBinding" type="tns:TestJaxWs">
    <soap12:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document"/>
    <operation name="jaxWsTest1">
      <soap12:operation soapAction=""/>
      <input>
        <soap12:body use="literal"/>
      </input>
      <output>
        <soap12:body use="literal"/>
      </output>
      <fault name="UserDefinedFault">
        <soap12:fault name="UserDefinedFault" use="literal"/>
      </fault>
    </operation>
  </binding>
  <service name="TestJaxWsService">
    <port name="testJaxWs" binding="tns:testJaxWsBinding">
      <soap12:address location="http://webhost:8085/wsrm/TestJaxWsService"/>
    </port>
  </service>
</definitions>
```

The following is an example for modifying the `TestJaxWsService_schema1.xsd`. The parts in Italics indicate modifications.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema version="1.0" targetNamespace="http://example.com/sample"
xmlns:tns="http://example.com/sample" xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="UserDefinedFault" type="tns:UserDefinedFault"/>

  <xs:element name="jaxWsTest1" type="tns:jaxWsTest1"/>

  <xs:element name="jaxWsTest1Response" type="tns:jaxWsTest1Response"/>

  <xs:complexType name="jaxWsTest1">
    <xs:sequence>
      <xs:element name="arg0" type="xs:string" minOccurs="0"/>
      <xs:element name="arg1" type="xs:int"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="jaxWsTest1Response">
    <xs:sequence>
      <xs:element name="return" type="xs:string" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="UserDefinedFault">
    <xs:sequence>
      <xs:element name="message" type="xs:string" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Change the name of the modified `TestJaxWsService.wsdl` file to `input.wsdl`, and save the file in the `c:\temp\jaxws\works\wsrm\server\WEB-INF\wsdl\` directory.

35.3.2 Adding the WS-RM Policy in the WSDL file

Add the WS-RM Policy in the newly added WSDL file. The following are the items to be added:

1. Defining the name space prefix to be used with the WS-RM Policy
2. Defining the policy as the child element of the `wsdl:definitions` element
3. Referencing the policy with the child element of the `wsdl:binding` element

The following is an example for adding the policy:

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="TestJaxWsService"

<!-- omitted -->

<!-- 1.Defining the name space prefix to be used with the WS-RM Policy -->
  xmlns:wsp="http://www.w3.org/ns/ws-policy"
  xmlns:wsrmp="http://docs.oasis-open.org/ws-rx/wsrmp/200702"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd"
  xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl">

<!-- 2.Defining the policy as the child element of the wsdl:definitions element -->
  <wsp:Policy wsu:Id="WSRM_policy">
    <wsp:ExactlyOne>
      <wsp:All>
        <wsrmp:RMAssertion>
          <wsp:Policy>
        </wsrmp:RMAssertion>
        <wsaw:UsingAddressing/>
      </wsp:All>
    </wsp:ExactlyOne>
  </wsp:Policy>

  <wsdl:types>

<!-- omitted -->

<!-- 3.Referencing the policy with the child element of the wsdl:binding element -->
  <wsdl:binding name="testJaxWsBinding" type="tns:TestJaxWs">
    <wsp:PolicyReference URI="#WSRM_policy"/>
    <!-- document/literal/wrapped -->

<!-- rest is omitted -->
```

35.3.3 Creating SEI

Execute the `cjwsimport` command to generate additional Java sources such as SEI required to develop the Web Services. For the `cjwsimport` command, see *14.1 cjwsimport command*.

The following is an example for executing the `cjwsimport` command:

```
> cd c:\temp\jaxws\works\wsrm\server\
> mkdir src\
> mkdir WEB-INF\classes\
> "%COSMINEXUS_HOME%\jaxws\bin\cjwsimport.bat" -generateService -s src -d WEB-INF
\classes WEB-INF\wsdl\input.wsdl
```

On successful termination of the `cjwsimport` command, the Java sources are generated in the `c:\temp\jaxws\works\wsrm\server\src\com\example\sample\` directory. Note that the directory path `com\example\sample\` (the directory path corresponding to the package) changes according to the Namespace Uri coding. For the mapping between Namespace URI and package, see *15.1.1 Mapping a name space to a package name*.

The following table lists the generated products:

Table 35-3: Java sources generated when SEI is created (Starting from WSDL)

File name	Explanation
JaxWsTest1.java	This is a JavaBean class corresponding to the 'type referenced by the wrapper element of the request message' in the WSDL definition.
JaxWsTest1Response.java	This is a JavaBean class corresponding to the 'type referenced by the wrapper element of the response message' in the WSDL definition.
ObjectFactory.java	This is an ObjectFactory class of the JAXB 2.2 specifications.
package-info.java	This is a package-info.java file.
TestJaxWs.java	This is the SEI corresponding to the TestJaxWsPort type.
TestJaxWsImpl.java	This is a skeleton class corresponding to the TestJaxWsPort type.
UserDefinedFault.java	This is a JavaBean class (fault bean) corresponding to the 'type referenced by the wrapper element of the fault message' in the WSDL definition.
UserDefinedException.java	This is the wrapper exception class of the fault bean.

The file names JaxWsTest1, TestJaxWs, and TestJaxWsImpl change according to the coding of the local name for the operation, the local name for the port type, and the local name for the services. For the mapping between the local name for the operation, local name for the port type, and local name for the services and Java sources, see *15. Mapping WSDL to Java*.

35.3.4 Creating the Web Service Implementation Class

Add the Web Service processing to the skeleton class to create a Web Service Implementation Class. Add the process to return the contents of the received request message as the response message along with the date information.

The following is an example for creating a Web Service Implementation Class:

```
package com.example.sample;

import java.util.Calendar;
import javax.jws.WebService;


@WebService(endpointInterface = "com.example.sample.TestJaxWs", targetNamespace =
"http://example.com/sample", serviceName = "TestJaxWsService", portName = "testJaxWs")
public class TestJaxWsImpl {

    public String jaxWsTest1(String information, int count)
        throws UserDefinedException
    {
        Calendar today = Calendar.getInstance();
        StringBuffer result = new StringBuffer( 256 );
        result.append( "We've got your #" );
        result.append( new Integer( count ) );
        result.append( " message \"" );
        result.append( information );
        result.append( "\"! It's " );
    result.append( String.format( "%04d.%02d.%02d %02d:%02d:%02d", new Object[]{
            new Integer( today.get( Calendar.YEAR ) ),
            new Integer( today.get( Calendar.MONTH ) + 1 ),
            new Integer( today.get( Calendar.DAY_OF_MONTH ) ),
            new Integer( today.get( Calendar.HOUR_OF_DAY ) ),
            new Integer( today.get( Calendar.MINUTE ) ),
            new Integer( today.get( Calendar.SECOND ) ) } ) );
        result.append( " now. See ya!" );

        return result.toString();
    }

}
```

The parts in Italics indicate the implementation added in the skeleton.

35.3.5 Compiling the Web Service Implementation Class

Use the javac command to compile the created Web Service Implementation Class.

The following is an example for compiling the Web Service Implementation Class:

```
> cd c:\temp\jaxws\works\wsrm\server\
> javac -encoding UTF-8 -cp "%COSMINEXUS_HOME%\jaxws\lib\cjjaxws.jar;%COSMINEXUS_HOME%
\CC\client\lib\j2ee-javax.jar;.\WEB-INF\classes" -d .\WEB-INF\classes src\com\example
\sample\TestJaxWsImpl.java
```

On successful termination of the javac command, TestJaxWsImpl.class in the c:\temp\jaxws\works\wsrm\server\WEB-INF\classes\com\example\sample\ directory is overwritten.

For the javac command, see the *JDK documentation*.

35.3.6 Creating a web.xml file

Create a web.xml file that is required as a WAR file component.

The following is an example for creating a web.xml file:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/
javaee/web-app_3_0.xsd">
  <description>Sample web service &amp;quot;wsrm&amp;quot;</description>
  <display-name>Sample_web_service_wsrm</display-name>
  <listener>
    <listener-class>
      com.cosminexus.xml.ws.transport.http.servlet.WSServletContextListener
    </listener-class>
  </listener>

  <servlet>
    <description>Endpoint servlet for Cosminexus JAX-WS</description>
    <display-name>Endpoint_servlet_for_Cosminexus_JAX_WS</display-name>
    <servlet-name>CosminexusJaxwsServlet</servlet-name>
    <servlet-class>
      com.cosminexus.xml.ws.transport.http.servlet.WSServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>CosminexusJaxwsServlet</servlet-name>
    <url-pattern>/TestJaxWsService</url-pattern>
  </servlet-mapping>
  <session-config>
    <session-timeout>60</session-timeout>
  </session-config>
</web-app>
```

When creating web.xml of version 2.5, specify 2.5 in the version attribute of the web-app element and specify http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd as the second location information in the xsd:schemaLocation attribute.

The created web.xml file is stored in the c:\temp\jaxws\works\wsrm\server\WEB-INF\ directory in the UTF-8 format. For the web.xml settings, see *3.4 Creating web.xml*.

35.3.7 Creating an application.xml file

Create an application.xml file that is required as an EAR file component.

The following is an example for creating an application.xml file:

```
<?xml version="1.0" encoding="UTF-8"?>
<application version="6" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/
```

```
javaee/application_6.xsd">

  <description>Sample application &amp;quot;wsrm&amp;quot;</description>
  <display-name>Sample_application_wsrm</display-name>
  <module>
    <web>
      <web-uri>wsrm.war</web-uri>
      <context-root>wsrm</context-root>
    </web>
  </module>
</application>
```

When creating `application.xml` of version 5, specify `5` in the `version` attribute of the `application` element and specify `http://java.sun.com/xml/ns/javaee/application_5.xsd` as the second location information in the `xsd:schemaLocation` attribute.

The created `application.xml` file is stored in the `c:\temp\jaxws\works\wsrm\server\META-INF\` directory in the UTF-8 format. For notes on creating an `application.xml` file, see *5.2.2 Notes on editing application.xml* in the *uCosminexus Application Server Application Development Guide.*

35.3.8 Creating EAR files

Use the `jar` command to create EAR files.

The following is an example for creating an EAR file:

```
> cd c:\temp\jaxws\works\wsrm\server\
> jar cvf wsrm.war .\WEB-INF
> jar cvf wsrm.ear .\wsrm.war .\META-INF\application.xml
```

On successful termination of the `jar` command, `wsrm.ear` is created in the `c:\temp\jaxws\works\wsrm\server\`directory.

For the `jar` command, see the *JDK documentation.*

35.4 Example of deploying and starting the service (Starting from WSDL/WS-RM 1.2)

This section describes how to deploy and start services when the development is done starting from WSDL.

35.4.1 Deploying the EAR files

Use the `cjimportapp` command to deploy the created EAR file to the J2EE Server.

The following example describes how to deploy the EAR files:

```
> cd c:\temp\jaxws\works\wsrm\server\
> "%COSMINEXUS_HOME%\CC\admin\bin\cjimportapp" jaxwsserver -nameserver
corbaname::testserver:900 -f wsrm.ear
```

For the `cjimportapp` command, see *cjimportapp (Importing J2EE applications)* in the *uCosminexus Application Server Command Reference Guide.*

For the method to deploy (import) J2EE applications by using the management portal, see*, 12.3.3 Importing J2EE applications* in the *uCosminexus Application Server Management Portal User Guide.*

35.4.2 Starting Web Services

Use the `cjstartapp` command to start the Web Service.

The following is an example for starting the Web Service:

```
> cd c:\temp\jaxws\works\wsrm\server\
> "%COSMINEXUS_HOME%\CC\admin\bin\cjstartapp" jaxwsserver -nameserver
corbaname::testserver:900 -name Sample_application_wsrm
```

For the `cjstartapp` command, see *cjstartapp (Starting J2EE applications)* in the *uCosminexus Application Server Command Reference Guide*.

For the method to start J2EE applications by using the management portal, see, *12.3.1 Starting J2EE applications* in the *uCosminexus Application Server Management Portal User Guide.*

35.5 Example of developing a Web Service client (starting from WSDI/WS-RM 1.2)

This section describes the examples for the development of Web Service clients, starting from WSDL.

35.5.1 Generating a service class

If you execute the `cjwsimport` command, Java source, such as service class that is required for developing a Web Service client is generated. For the `cjwsimport` command, see *14.1 cjwsimport command*

The following is an example when the Web Service client is developed in the same environment as the Web Service:

```
> cd c:\temp\jaxws\works\wsrm\client\
> mkdir src\
> mkdir classes\
> "%COSMINEXUS_HOME%\jaxws\bin\cjwsimport.bat" -s src -d classes ..\server\WEB-INF
\wsdl\input.wsdl
```

The following is an example when the Web Service client is developed in an environment other than the Web Service:

```
> cd c:\temp\jaxws\works\wsrm\client\
> mkdir src\
> mkdir classes\
> "%COSMINEXUS_HOME%\jaxws\bin\cjwsimport.bat" -s src -d classes http://webhost:8085/
wsrm/TestJaxWsService?wsdl
```

On successful termination of the `cjwsimport` command, the Java sources are generated in the `c:\temp\jaxws\works\wsrm\client\src\com\example\sample\` directory. Note that the directory path `com\example\sample\` (the directory path corresponding to the package) changes according to the coding of the namespace URI. For the mapping between Namespace URI and package, see *15.1.1 Mapping a names pace to a package name*.

The following table lists the generated products:

Table 35-4: Products created when the service class is generated (Starting from WSDL)

File name	Explanation
`JaxWsTest1.java`	This is a `JavaBean` class corresponding to the 'type referenced by the `wrapper` element of the request message' in the WSDL definition.
`JaxWsTest1Response.java`	This is a `JavaBean` class corresponding to the 'type referenced by the `wrapper` element of the response message' in the WSDL definition.
`ObjectFactory.java`	This is an `ObjectFactory` class of the JAXB 2.2 specifications.
`package-info.java`	This is a `package-info.java` file.
`TestJaxWs.java`	This is the SEI corresponding to the `TestJaxWsPort` type.
`TestJaxWsService.java`	This is a service class.
`UserDefinedFault.java`	This is a `JavaBean` class corresponding to the 'type referenced by the `wrapper` element of the fault message' in the WSDL definition.
`UserDefinedException.java`	This is the wrapper exception class of the fault bean.

The file names `JaxWsTest1`, `TestJaxWs`, and `TestJaxWsService` change according to the coding of the local name for the operation, the local name for the port type, and the local name for the services. For the mapping between the local name for the operation, the local name for the port type, and the local name for the services and Java sources, see the following sections:

- *15.1.2 Mapping a port type to a SEI name*

- *15.1.3 Mapping an operation to a method name*
- *15.1.4 Mapping a message part to a parameter and return value (For wrapper style)*
- *15.1.5 Mapping a message part to a parameter and return value (For non-wrapper style)*

35.5.2 Creating an implementation class for the Web Service client

Create an Implementation Class for Web Service client that uses the Web Services.

The following is an example for creating a Web Service client that invokes Web Services once:

```
package com.example.sample.client;

import com.example.sample.TestJaxWs;
import com.example.sample.TestJaxWsService;
import com.example.sample.UserDefinedException;

public class TestClient {
    public static void main( String[] args ) {
        try {
            TestJaxWsService service = new TestJaxWsService();
            TestJaxWs port = service.getTestJaxWs();

            String returnValue = port.jaxWsTest1( "Invocation test.", 1003 );

            System.out.println( "[RESULT] " + returnValue );
        }
        catch( UserDefinedException e ){
            e.printStackTrace();
        }
    }

}
```

The created `TestClient.java` is stored in the `c:\temp\jaxws\works\wsrm\client\src\com\example\sample\client\` directory in the UTF-8 format. Note that `com.example.sample`, `TestJaxWs`, `TestJaxWsService`, `TestJaxWs`, and `jaxWsTest1` change according to the package name, the class name, and the method name in the class of the generated Java sources. When you want to develop a Web Service with a different configuration, you must review, and if necessary revise, the coding of the package name, the class name, and the method name in the class.

35.5.3 Adding sequence termination processing in the Implementation Class for Web Service client

Add sequence termination processing in the Implementation Class for Web Service client.

The following is an example for adding the sequence termination processing:

```
package com.example.sample.client;

import com.example.sample.TestJaxWs;
import com.example.sample.TestJaxWsService;
import com.example.sample.UserDefinedException;
import com.sun.xml.ws.Closeable;

public class TestClient {
    public static void main( String[] args ) {
        TestJaxWsService service = null;
        TestJaxWs port = null;
        try {
            service = new TestJaxWsService();
            port = service.getTestJaxWs();

            String returnValue = port.jaxWsTest1( "Invocation test.", 1003 );

            System.out.println( "[RESULT] " + returnValue );
        }
        catch( UserDefinedException e ){
            e.printStackTrace();
        }
        finally {
```

```
            if( port != null ) {
                ((Closeable)port).close();
            }
        }
    }

}
```

35.5.4 Compiling the implementation class for the Web Service client

Use the `javac` command to compile the created Implementation class for Web Service client.

The following is an example for compiling the implementation class:

```
> cd c:\temp\jaxws\works\wsrm\client\
> javac -encoding UTF-8 -cp "%COSMINEXUS_HOME%\jaxws\lib\cjjaxws.jar;%COSMINEXUS_HOME%
\CC\client\lib\j2ee-javax.jar;%COSMINEXUS_HOME%\jaxp\lib\csmjaxb.jar;.\classes" -d .
\classes src\com\example\sample\client\TestClient.java
```

On successful termination of the `javac` command, `TestClient.class` is generated in the `c:\temp\jaxws\works\wsrm\client\classes\com\example\sample\client\` directory. For the `javac` command, see the *JDK documentation*.

35.6 Example of executing the Web Service (Starting from WSDL/WS-RM 1.2)

This section describes the example for the execution of Web Service clients, starting from WSDL.

35.6.1 Creating an option definition file for Java applications

Create an option definition file (`usrconf.cfg`) for Java applications required for executing the Web Service.

The following is an example for creating the option definition file for Java applications:

```
add.class.path=Conminexus-installation-directory\jaxws\lib\cjjaxws.jar
add.class.path=.\classes
ejb.client.log.directory=logs
add.jvm.arg=-Dcosminexus.home=Cosminexus-installation-directory
add.jvm.arg=-Dejbserver.server.prf.PRFID=<PRF ID>
```

For the *Cosminexus-installation-directory* part, use the absolute path to specify the path where Cosminexus is installed.

The created option definition file for Java applications is stored in the `c:\temp\jaxws\works\wsrm\client\` directory in the UTF-8 format. For the option definition file for Java applications, see *14.2 usrconf.cfg (Option Definition file for Java applications)* in the *uCosminexus Application Server Definition Reference Guide*.

35.6.2 Creating a user property file for Java applications

Create a user property file for Java applications required for executing the Web Service.

Because the settings are not particularly changed in this example, create an empty file in the `c:\temp\jaxws\works\wsrm\client\` directory. For the user property file for Java applications, see *14.3 usrconf.properties (User Property file for Java applications)* in the *uCosminexus Application Server Definition Reference Guide.*

35.6.3 Executing the Web Service client

Use the `cjclstartap` command to execute the Web Service client.

The following is an example for executing the Web Service client:

```
> cd c:\temp\jaxws\works\wsrm\client\
> "%COSMINEXUS_HOME%\CC\client\bin\cjclstartap"  com.example.sample.client.TestClient
```

On successful termination of the `cjclstartap` command, the execution results of the Web Service client are displayed. The following is an example for displaying the execution results:

```
KDJE40053-I The cjclstartap command will now start. (directory for the user
definition file = c:\temp\jaxws\works\wsrm\client, PID = 2636)
--------------------
[RESULT] We've got your #1003 message "Invocation test."! It's 2007.11.28 14:50:50
now. See ya!
KDJE40054-I The cjclstartap command was stopped. (PID = 2636, exit status = 0)
```

The parts in *Italics* change according to the execution timing and environment.

For the `cjclstartap` command, see *cjclstartap (Starting Java applications) in the uCosminexus Application Server Command Reference Guide.*

36 Handler Frame Work

In the Cosminexus JAX-WS functionality, you can use the handler framework defined in the JAX-WS 2.2 specifications to extend the Web Service functionality.

This chapter gives an overview of the handler framework and describes the flow of processing and the usage settings.

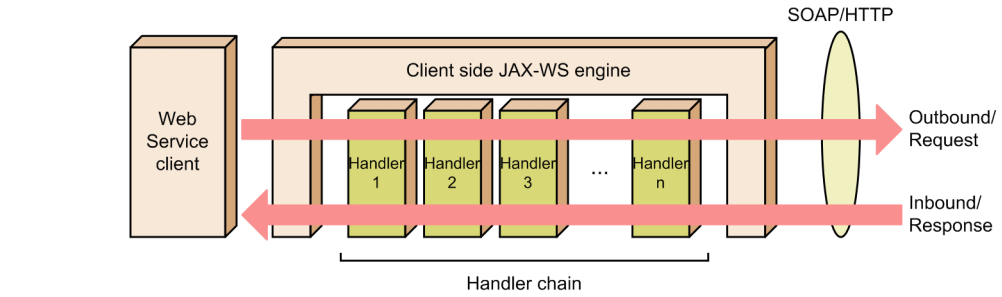
36.1 What is the handler framework

A handler framework is the functionality (framework) for intercepting and adding processing in the JAX-WS engine when a SOAP Message is sent and received. You can add multiple handlers between the Web Service client and Web Service Implementation Class or the Provider Implementation Class, and extend the Web Service functionality.

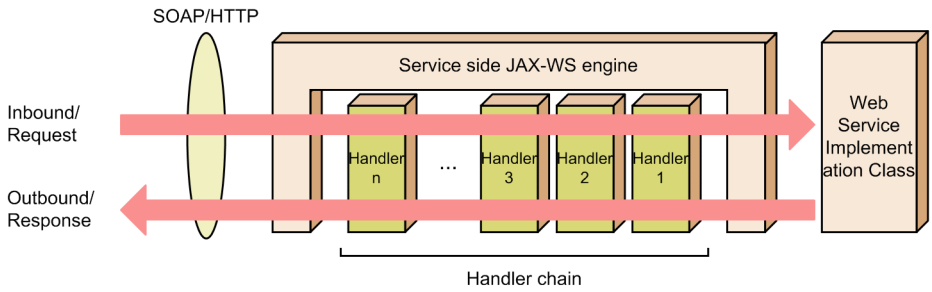
The following figure shows the flow of processing in the handler framework:

Figure 36-1: Flow of processing in the handler framework

• Sending and receiving Web Service client



• Sending and receiving Web Service Implementation Class



The contents of handler processing differ in each combination of *outbound* or *inbound* and request or response. The following table lists the combinations and the processing contents:

Table 36-1: Combinations of outbound or inbound and request or response and the processing contents

Outbound or Inbound	Request or response	Processing contents
Outbound	Request	Indicates that the Web Service client sends the request message. In this processing, *distributing the messages* means sending the request messages to the Web Service.
	Response	Indicates that the Web Service Implementation Class or the Provider Implementation Class sends the response message. In this processing, *distributing the messages* means sending the response messages to the Web Service client. The outbound does not work because no response message is sent in the one-way operations.
Inbound	Request	Indicates that the Web Service Implementation Class or the Provider Implementation Class receives the request message. In this processing, *distributing the messages* means allocating the request messages to the Web Service Implementation Class or the Provider Implementation Class.
	Response	Indicates that the Web Service client receives the response. In this processing, *distributing the messages* means allocating the response messages to the Web Service client.

Outbound or Inbound	Request or response	Processing contents
Inbound	Response	The inbound does not work as no response message is received in the one-way operations.

You can acquire the message context in the handler. For details about the message context, see *19.2.5 Using a message context*.

For details about the architecture of the handler framework, see *JAX-WS 2.2 specifications*.

36.2 Precautions on using the Web Service security functionality

When applying Cosminexus Web Services - Security (Web Service security functionality) to a Web Services client, you cannot use the *Web Service Security Handler* and the handler framework described in this chapter simultaneously.

For Cosminexus Web Services - Security and Web Service security handler, see the *uCosminexus Application Server Web Service Security Users Guide*.

36.3 Notes on applying to the EJB Web Services

When applying the handler framework to the EJB Web Services, you cannot use the functions provided by the EJB container concurrently, because the handler framework applied to the EJB Web Services runs in the Web container. For the functions provided by the EJB container, see *10.19 Invoking EJB Web Service.*

36.4 Types of handlers

Define the logical handler and protocol handler as per the JAX-WS 2.2 specifications. The protocol handler also defines the SOAP handler. The following is a description of each of these handlers:

Logical handler

This handler implements the `javax.xml.ws.handler.LogicalHandler` interface.

Protocol handler

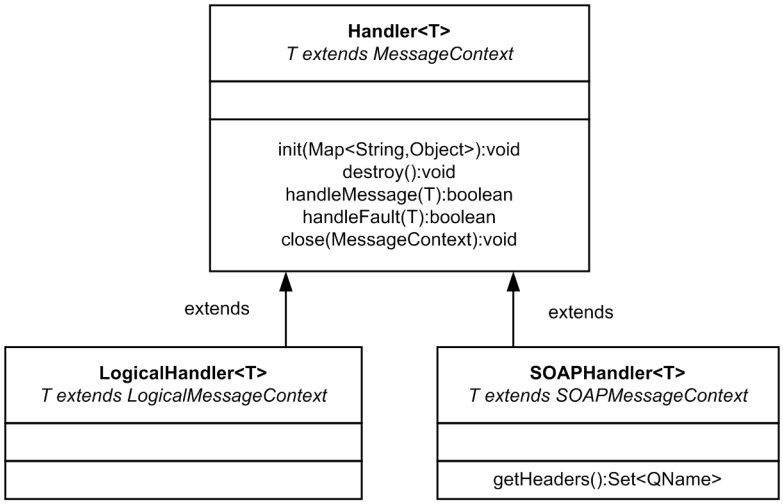
This handler implements all the inherited interfaces of `javax.xml.ws.handler.Handler`, except the `javax.xml.ws.handler.LogicalHandler` interface.

SOAP handler

This handler implements the `javax.xml.ws.handler.soap.SOAPHandler` interface.

The following figure shows the relationship of handlers (class hierarchy):

Figure 36-2: Class hierarchy of handlers



You can use the logical handler and SOAP handler in the Cosminexus JAX-WS functionality.

If a handler that is neither a logical handler nor a SOAP handler is specified in the handler chain, an error message (KDJW00009-E) is output to the log and standard error output when the Web Service is initialized in the JAX-WS engine on the Web Service. In the JAX-WS engine on the Web Service client, `javax.xml.ws.WebServiceException` is thrown when an attempt is made to acquire the port.

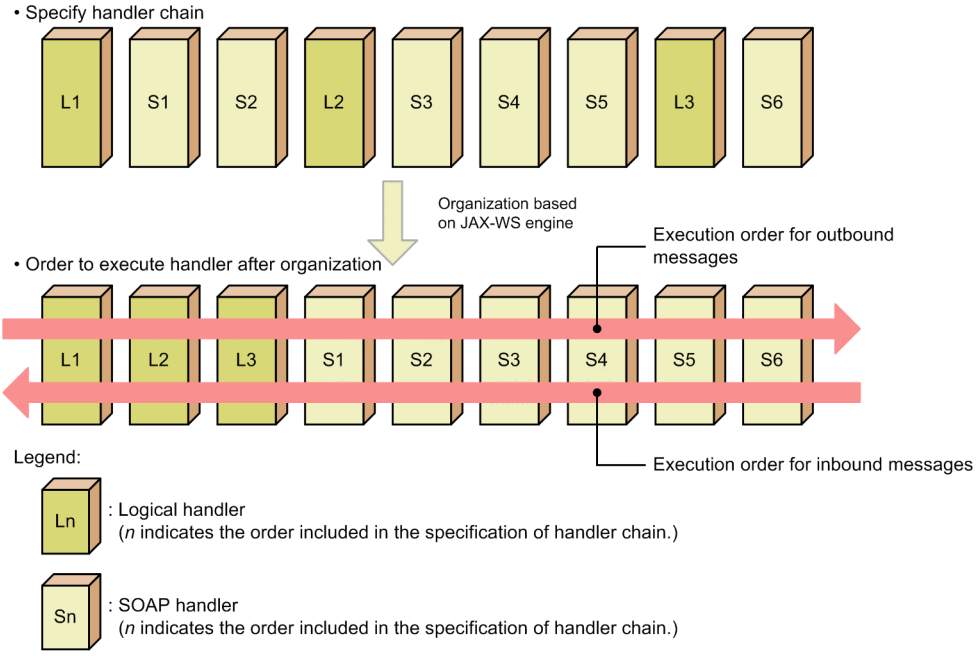
The handler that implements both, the `javax.xml.ws.handler.LogicalHandler` interface and the `javax.xml.ws.handler.soap.SOAPHandler` interface (handler that is logical handler as well as SOAP handler) is considered as the logical handler.

36.5 Execution sequence and organization of the handler chain

The handler chain is organized in such a manner that the logical handler precedes the SOAP handler. The sequence of the logical handlers and SOAP handlers is in accordance with the sequence included in the handler chain settings.

The following figure shows an example of the organization of the handler chain:

Figure 36-3: Example of the organization of the handler chain



As shown in the figure, the handlers are executed in the following sequence in the JAX-WS engine:

In outbound messages:

L1 -> L2 -> L3 -> S1 -> S2 -> S3 -> S4 -> S5 -> S6

In inbound messages:

S6 -> S5 -> S4 -> S3 -> S2 -> S1 -> L3 -> L2 -> L1

For the outbound messages, the handlers are executed in the handler chain sequence from the first handler. For the inbound messages, the handlers are executed in the reverse handler chain sequence from the last handler.

However, if the handler throws an exception or if the `handleMessage` method or `handleFault` method returns `false`, the direction of the handler processing changes.

This section describes the processing in the `handleMessage` method, `handleFault` method, and `close` method respectively.

36.5.1 Processing of the handleMessage method

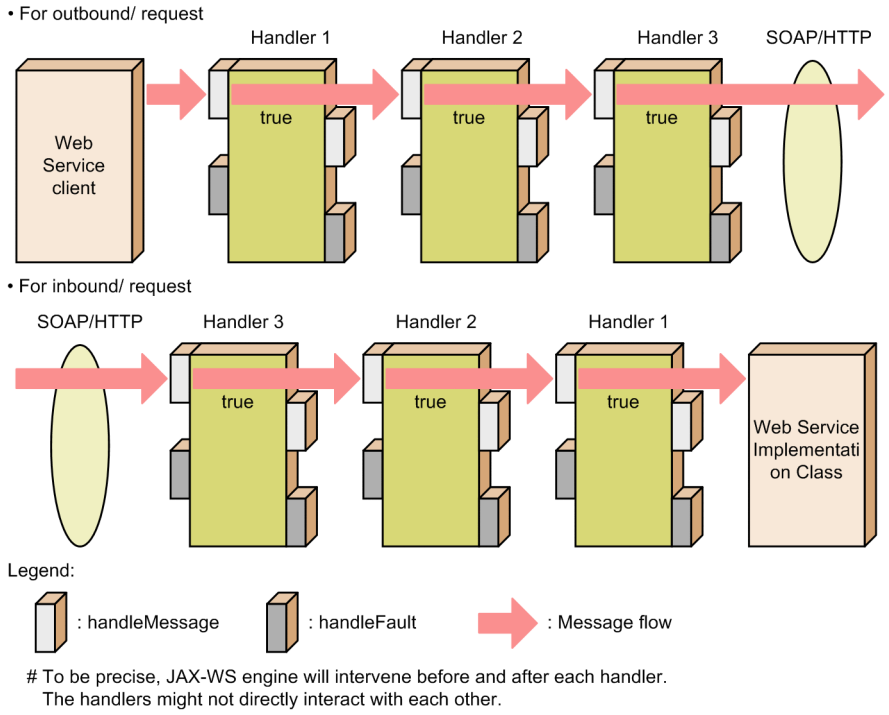
This subsection describes the processing when the `handleMessage` method returns true, returns false, and throws an exception.

(1) When the method returns true

When the `handleMessage` method returns `true`, the JAX-WS engine processes the handler in the same direction as the currently running direction. If there are no more handlers, the messages are distributed.

The following figure shows the flow of request processing:

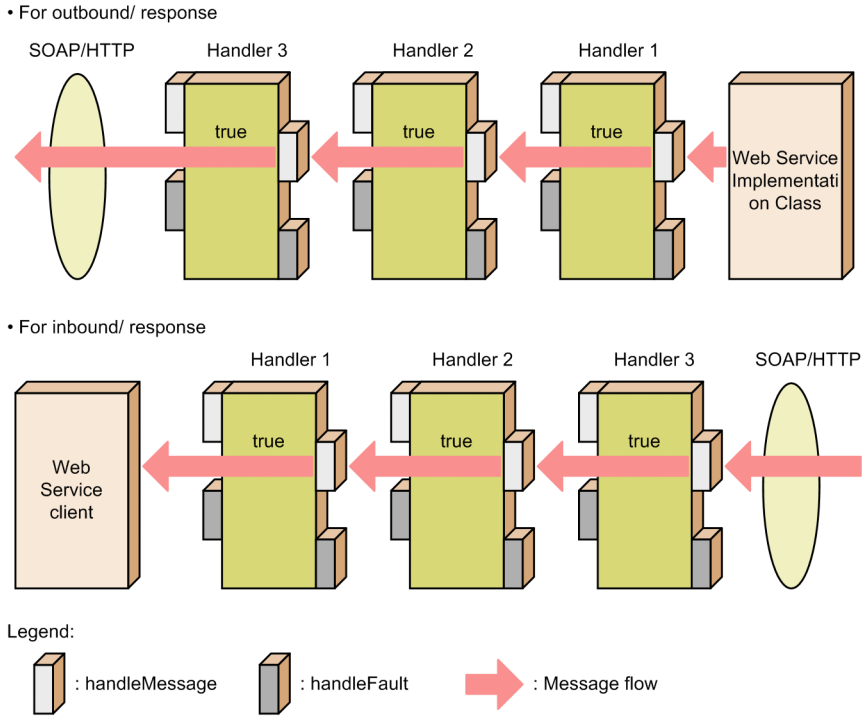
Figure 36-4: Processing when the handleMessage method returns true (request)



To be precise, JAX-WS engine will intervene before and after each handler.
The handlers might not directly interact with each other.

The following figure shows the flow of response processing:

Figure 36-5: Processing when the handleMessage method returns true (response)



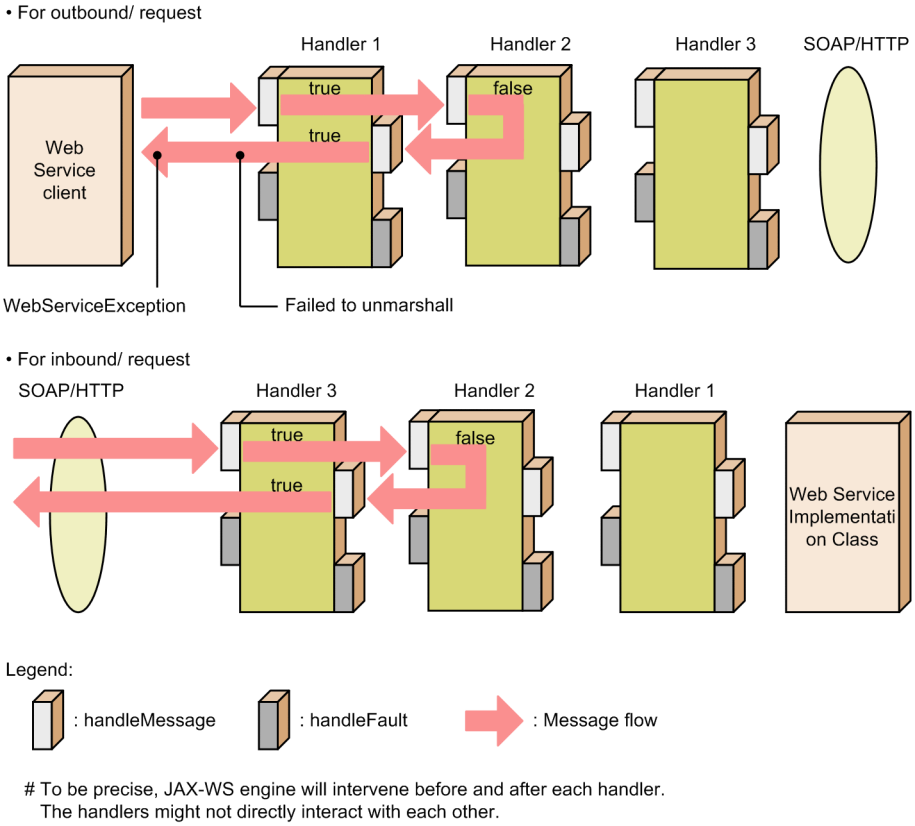
To be precise, JAX-WS engine will intervene before and after each handler.
The handlers might not directly interact with each other.

(2) When the method returns false

When the `handleMessage` method that is processing the request message returns `false`, the JAX-WS engine reverses the direction of the message and handler processing, and invokes the `handleMessage` method of the previous handler. If there are no more handlers, the messages are distributed.

The following figure shows the flow when handler 2 returns `false` during request processing:

Figure 36-6: Processing when the handleMessage method returns false (request)

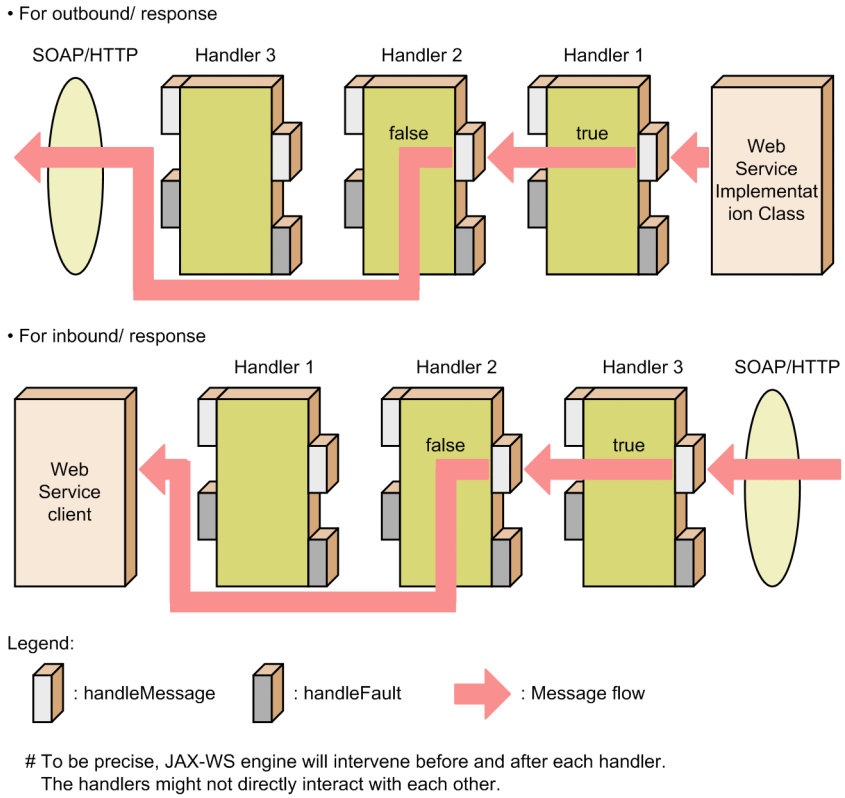


The JAX-WS engine does not change the message set in the message context. Therefore, the request messages are distributed to the Web Service client as is. In the Web Service client where the intended response messages were not distributed, the `javax.xml.ws.WebServiceException` is thrown.

When the `handleMessage` method that processes the response message returns `false`, the direction of the message does not change. However, all the subsequent handlers are omitted and the messages are distributed.

The following figure shows the flow when handler 2 returns `false` during response processing:

Figure 36-7: Processing when the handleMessage method returns false (response)



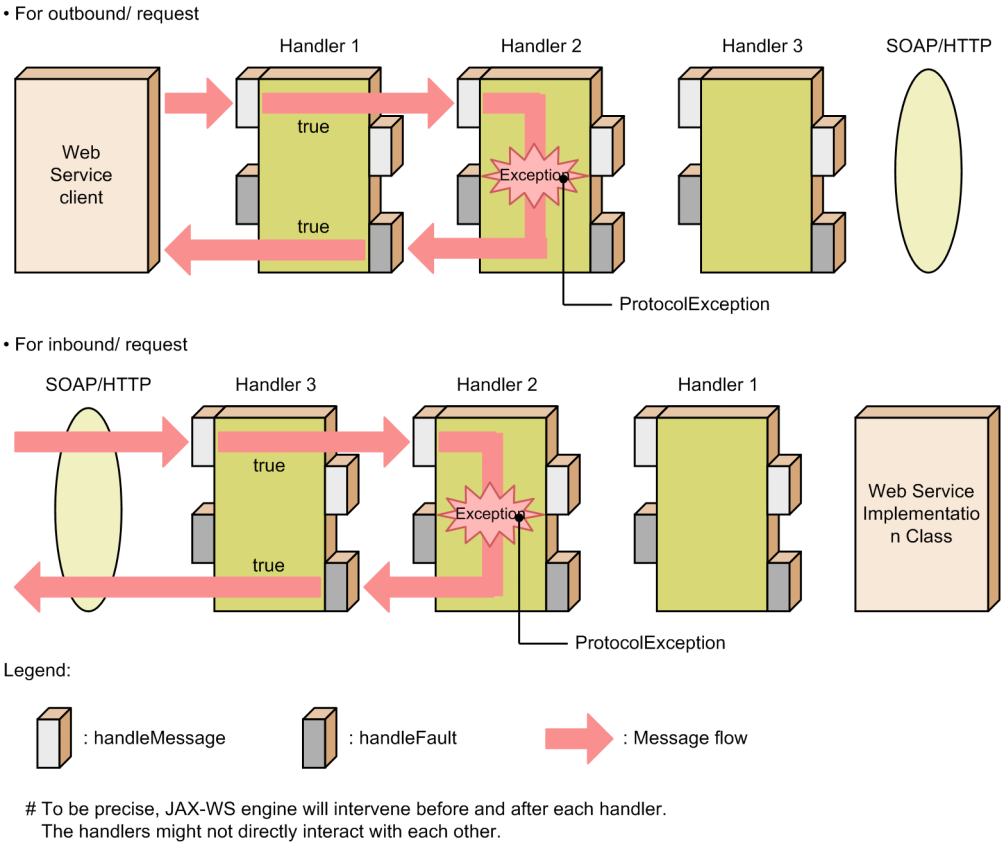
To be precise, JAX-WS engine will intervene before and after each handler.
The handlers might not directly interact with each other.

(3) When the method throws the ProtocolException or a sub-class

When the `handleMessage` method that processes the request message throws the `ProtocolException` or a sub-class, the JAX-WS engine reverses the direction of the message and handler processing, and invokes the `handleFault` method of the previous handler. If there are no more handlers, the messages are distributed.

The following figure shows the flow when handler 2 throws the `ProtocolException` during request processing:

Figure 36-8: Processing when handleMessage throws the ProtocolException (request)



To be precise, JAX-WS engine will intervene before and after each handler.
The handlers might not directly interact with each other.

The JAX-WS engine generates and propagates the fault message. The following table lists and describes the generated messages that differ depending on SOAP versions:

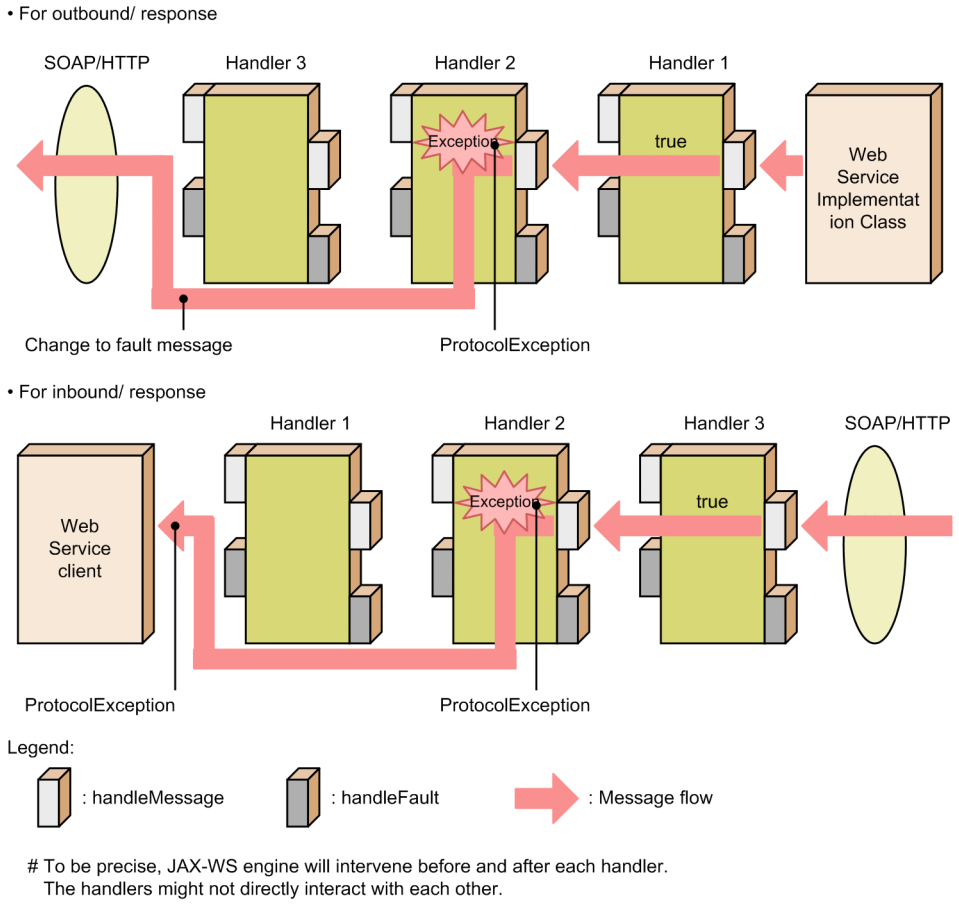
Table 36-2: Faultmessages generated by the JAXWS engine (ProtocolException throw/ request)

Version	Fault message	
SOAP 1.1 specification	In the Web Service client	`faultcode` is `QName soapenv:Client`
	In the Web Service	`faultcode` is `QName soapenv:Server`
SOAP 1.2 specification	In the Web Service client	`soapenv12:Code` is `QName soapenv12:Sender`
	In the Web Service	`soapenv12:Code` is `QName soapenv12:Receiver`

When the `handleMessage` method that is processing the response message throws the `ProtocolException` or a sub-class, the direction of the message does not change. However, all the subsequent handlers are omitted and the exception is thrown as is.

The following figure shows the flow when handler 2 throws the `ProtocolException` during response processing:

Figure 36-9: Processing when handleMessage throws the ProtocolException (response)



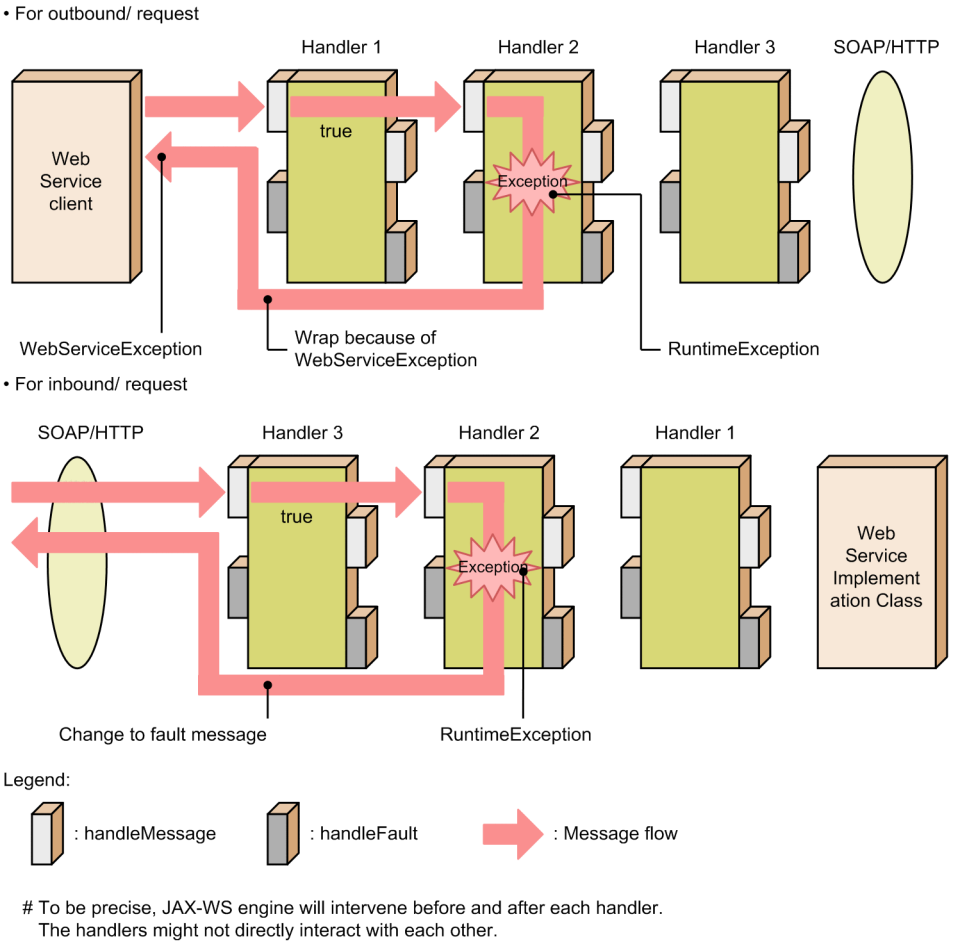
In the Web Service, the JAX-WS engine converts the exception into a SOAP fault and sends the fault. For details about the fault message conversion, see *10.4.1(2) Runtime exception binding*.

(4) When the method throws other runtime exceptions

When the `handleMessage` method that is processing the request message throws other runtime exceptions, the JAX-WS engine reverses the direction of the message and handler processing. The JAX-WS engine also omits all the subsequent handlers and throws the exception as is.

The following figure shows the flow when handler 2 throws the `RuntimeException` during request processing:

Figure 36-10: Processing when handleMessage throws RuntimeException (request)



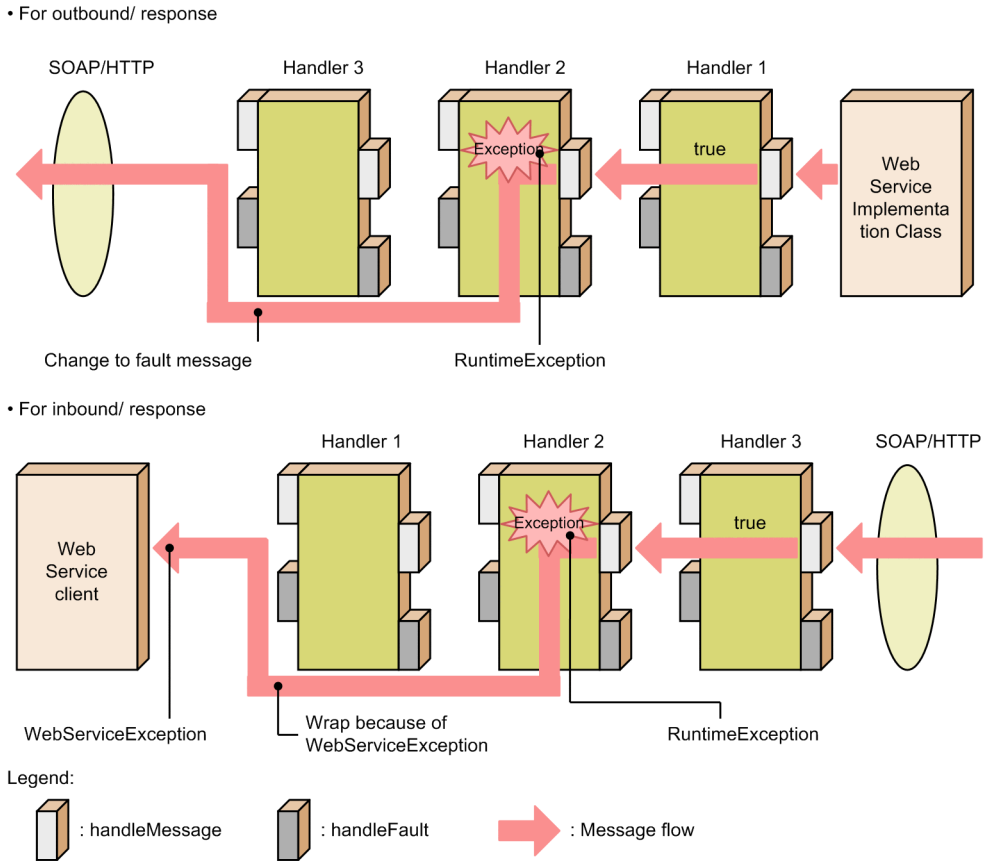
In the Web Service, the JAX-WS engine converts the exception into a SOAP fault and sends the fault. For details about the fault message conversion, see the subsection *10.4.1(2) Runtime exception binding*.

In the Web Service client, the JAX-WS engine wraps the exception with `javax.xml.ws.WebServiceException` and throws `WebServiceException`.

When the `handleMessage` method that is processing the response message throws a runtime exception, the direction of the message does not change. However, all the subsequent handlers are omitted and the messages are distributed.

The following figure shows the flow when handler 2 throws the `RuntimeException` during response processing:

Figure 36-11: Processing when handleMessage throws RuntimeException (response)



In the Web Service, the JAX-WS engine converts the exception into a SOAP fault and sends the fault. For details about the fault message conversion, see the subsection *10.4.1(2) Runtime exception binding*.

In the Web Service client, the JAX-WS engine wraps the exception with `javax.xml.ws.WebServiceException` and throws `WebServiceException`.

36.5.2 Processing of the handleFault method

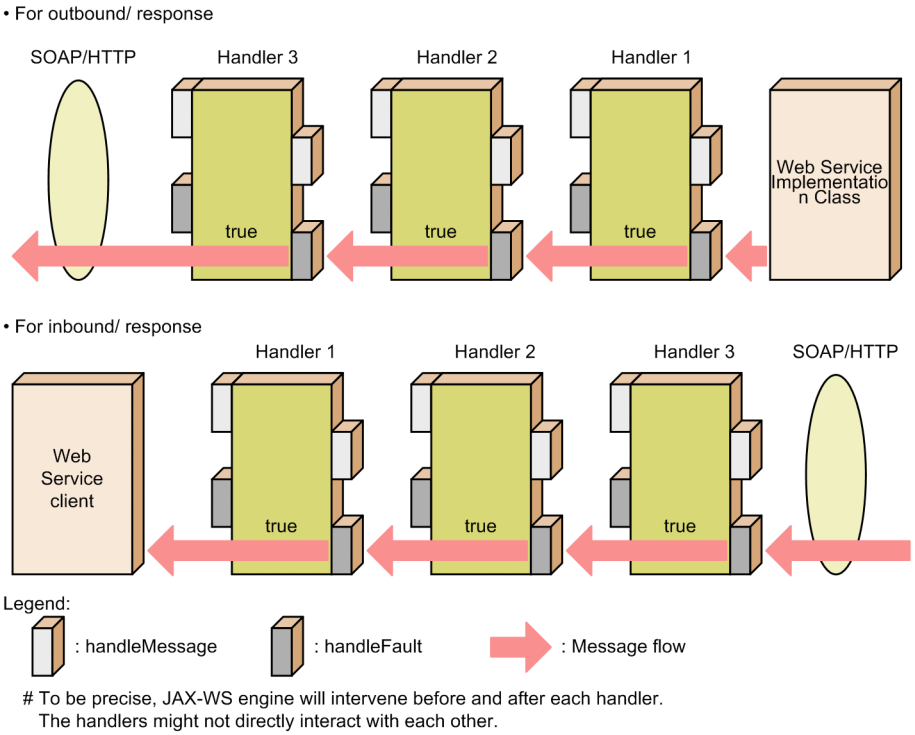
This subsection describes the processing when the `handleFault` method returns true, returns false, and throws an exception.

(1) When the method returns true

When the `handleFault` method returns `true`, the JAX-WS engine processes the handler in the same direction as the currently running direction. If there are no more handlers, the messages are distributed.

The following figure shows the flow of response processing:

Figure 36-12: Processing when the handleFault method returns true (response)

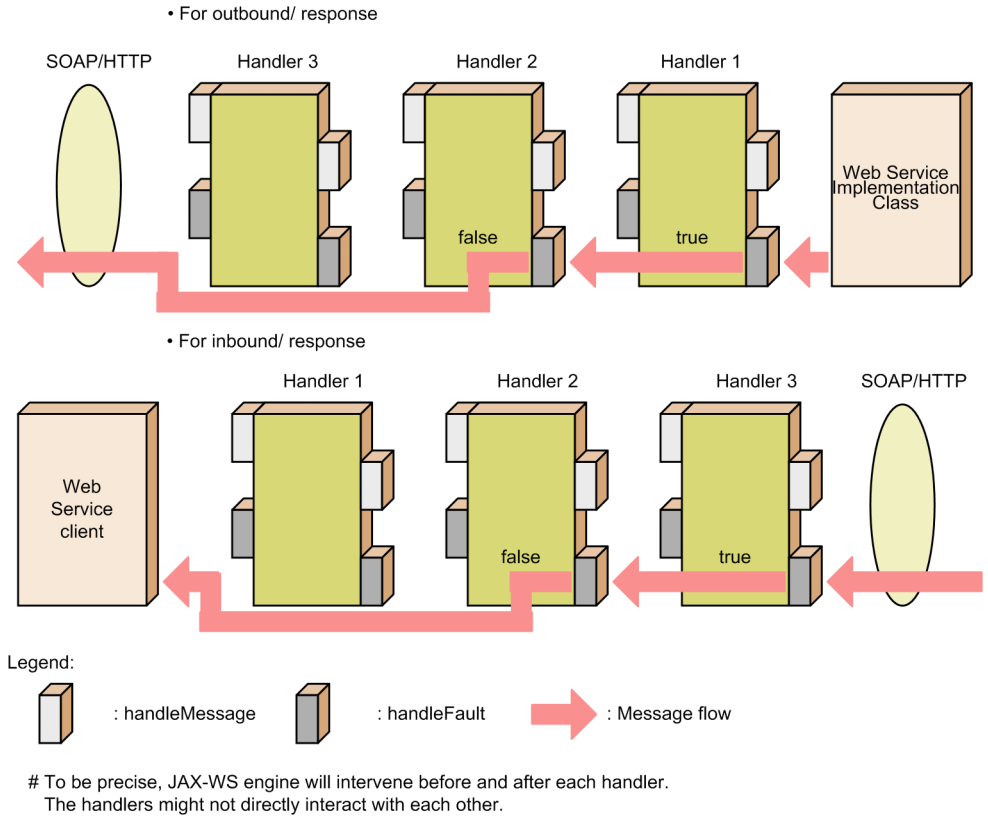


(2) When the method returns false

When the `handleFault` method returns `false`, the JAX-WS engine retains the direction that is currently running, omits all the subsequent handlers and distributes the messages.

The following figure shows the flow when handler 2 returns `false` during response processing:

Figure 36－13: Processing when the handleFault method returns false (response)

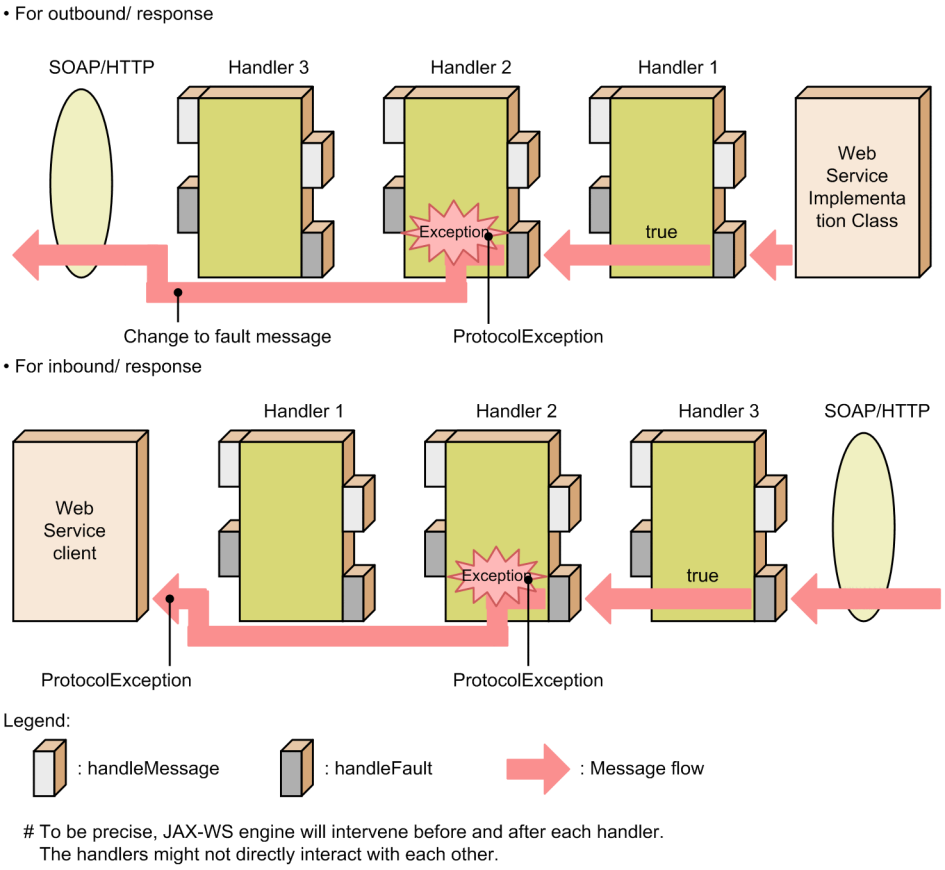


(3) When the method throws the ProtocolException or a sub-class

When the `handleFault` method throws the `ProtocolException` or a sub-class, the JAX-WS engine retains the direction that is currently running, omits all the subsequent handlers, and throws an exception.

The following figure shows the flow when handler 2 throws the `ProtocolException` during response processing:

Figure 36-14: Processing when handleFault returns the ProtocolException (response)



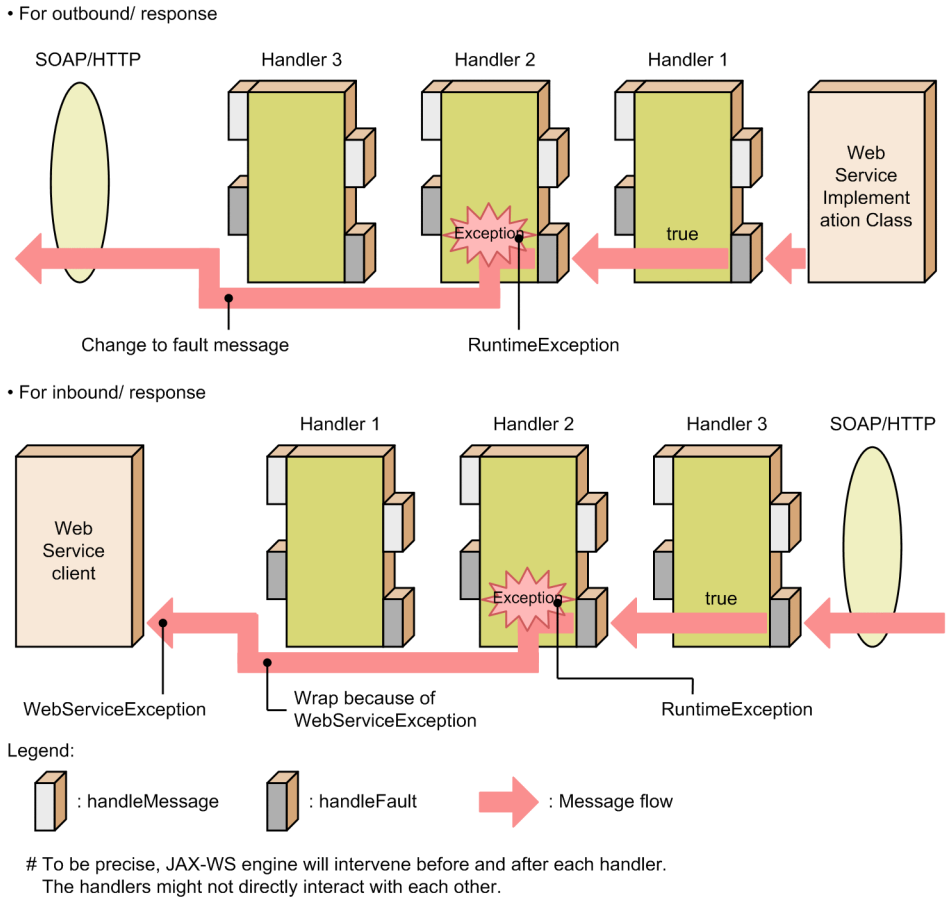
In the Web Service, the JAX-WS engine converts the exception into a SOAP fault and replaces the originally stored SOAP fault. Therefore, the JAX-WS engine actually sends the newly generated fault message. For details about the fault message conversion, see *10.4.1(2) Runtime exception binding*.

(4) When the method throws other runtime exceptions

When the `handleFault` method throws other runtime exceptions, the JAX-WS engine retains the direction that is currently running, omits all the subsequent handlers, and distributes the messages.

The following figure shows the flow when handler 2 throws the `RuntimeException` during response processing:

Figure 36－15: Processing when handleFault returns the RuntimeException (response)



In the Web Service, the JAX-WS engine converts the exception into a SOAP fault and replaces the originally stored SOAP fault. Therefore, the JAX-WS engine actually sends the newly generated fault message. For details about the fault message conversion, see *10.4.1(2) Runtime exception binding*.

In the Web Service client, the JAX-WS engine wraps the exception with `javax.xml.ws.WebServiceException` and throws `WebServiceException`.

36.5.3 Processing of the close method

Just before distributing the response messages to the Web Service client, the JAX-WS engine invokes the `close` method of the handlers that are already invoked.

The `close` method is invoked in the reverse order of the invocation of the handlers. Therefore, if the processing of the handler is reversed while the request message is being processed, the `close` method of the un-invoked handlers is not invoked. However, if all the handlers are executed for processing the request message, even if the invocation of handlers is omitted during the processing of the response message, the `close` method of all the handlers is invoked in the reverse order of the order when the request message was processed.

Also, in a one-way operation, the calling order for Web Services and the Web Client differs as follows:

Web Services

After calling Web Services Implementation Class or Provider Implementation Class, the `close` method of the already called handler is called in the reverse order of the order in which the one-way messages were processed.

Web Client

After receiving an HTTP response, the `close` method of the already called handler is called in the reverse order of the order in which the one-way messages were processed.

36.6 Initializing and destroying the handler

This section describes the operations of the JAX-WS engine when the handler is initialized and when the handler is destroyed.

(1) In the Web Service

When the handler has a method that is annotated using the `javax.annotation.PostConstruct` annotation, that method is invoked to initialize the Web Service.

When the handler has a method that is annotated using the `javax.annotation.PreDestory` annotation, that method is invoked to destroy the Web Service.

(2) In Web Service client

Even if the handler has methods that are annotated using the `javax.annotation.PostConstruct` annotation or `javax.annotation.PreDestory` annotation, those methods are not invoked.

36.7 Operations and settings for the handler when the SOAP Header is included in the SOAP Message

In the handler, Web Service Implementation Class, and stub-based Web Service client, you can set up a SOAP Header that can be processed in the SOAP Message. Note that even if you set up SOAP Header in a Provider Implementation Class and a dispatch-based Web Service client, the SOAP Header cannot be processed.

In this section, the operations of the handler when a SOAP Message containing a SOAP Header is received will be described separately for the Web Service and the Web Service client. This section also describes the settings for the SOAP Header.

36.7.1 Operations of the handler when the SOAP Header is included in the SOAP Message (in the Web Service)

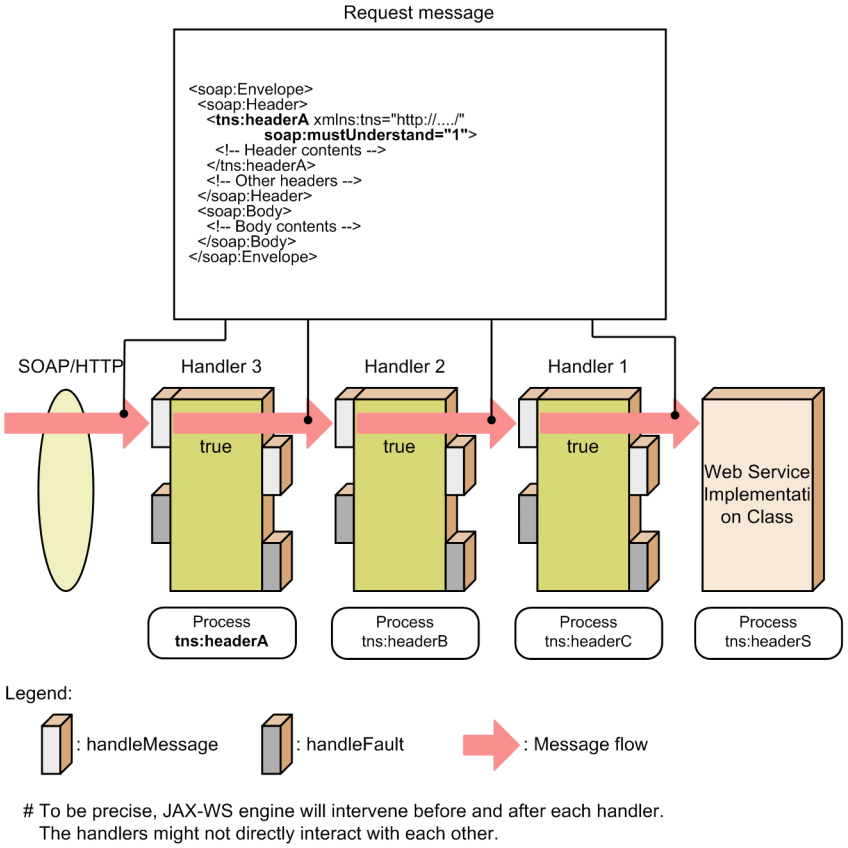
When a SOAP Message containing a SOAP Header is received, the handler processing varies according to the settings in the `soap:mustUnderstand` attribute of the SOAP Header. This subsection describes the processing contents when the `soap:mustUnderstand` attribute is `1` and when the `soap:mustUnderstand` attribute is not `1`.

(1) When the soap:mustUnderstand attribute is '1'

When a SOAP Message with a SOAP Header wherein `soap:mustUnderstand` attribute is `1` is received, all the handlers are invoked if that SOAP Header can be processed using one of the set up handlers or the Web Service Implementation Class.

The following figure shows the handler processing when the SOAP Header can be processed using handlers:

Figure 36-16: Processing when the SOAP Header can be processed using handlers (in the Web Service)

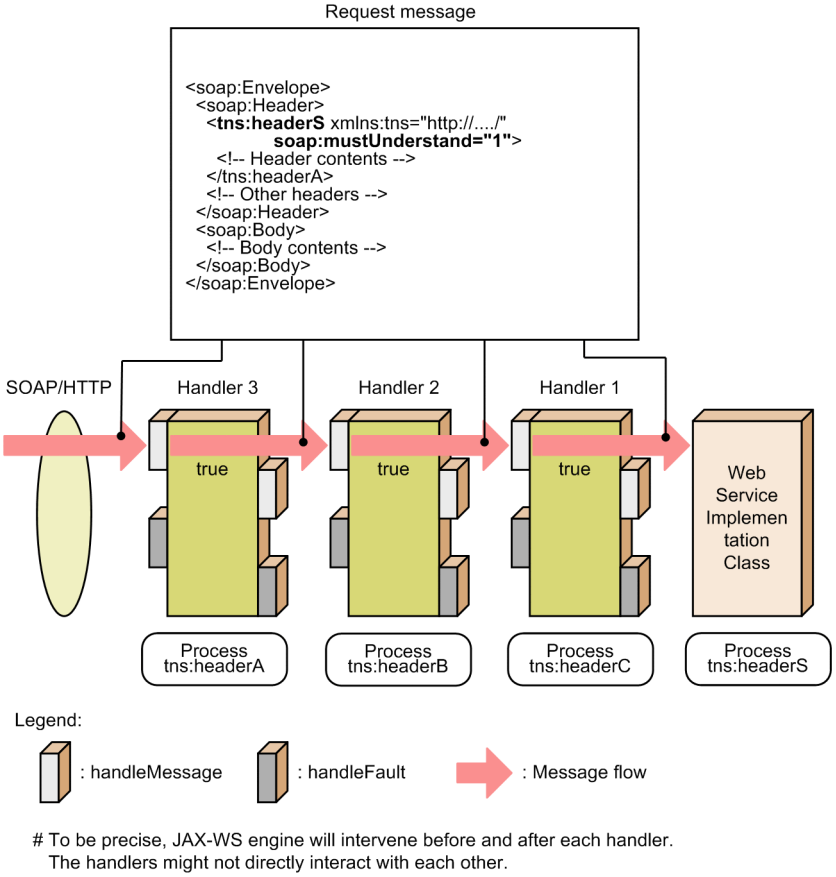


To be precise, JAX-WS engine will intervene before and after each handler.
The handlers might not directly interact with each other.

The example in the figure assumes that handler 3 that can process the element name `tns:headerA` is set up. At this time, if the SOAP Header `tns:headerA` is received, all the handlers - handler 3, handler 2, and handler 1 are invoked.

The following figure shows the handler processing when the SOAP Header can be processed using the Web Service Implementation Class:

Figure 36-17: Processing when the SOAP Header can be processed using the Web Service Implementation Class (in the Web Service)



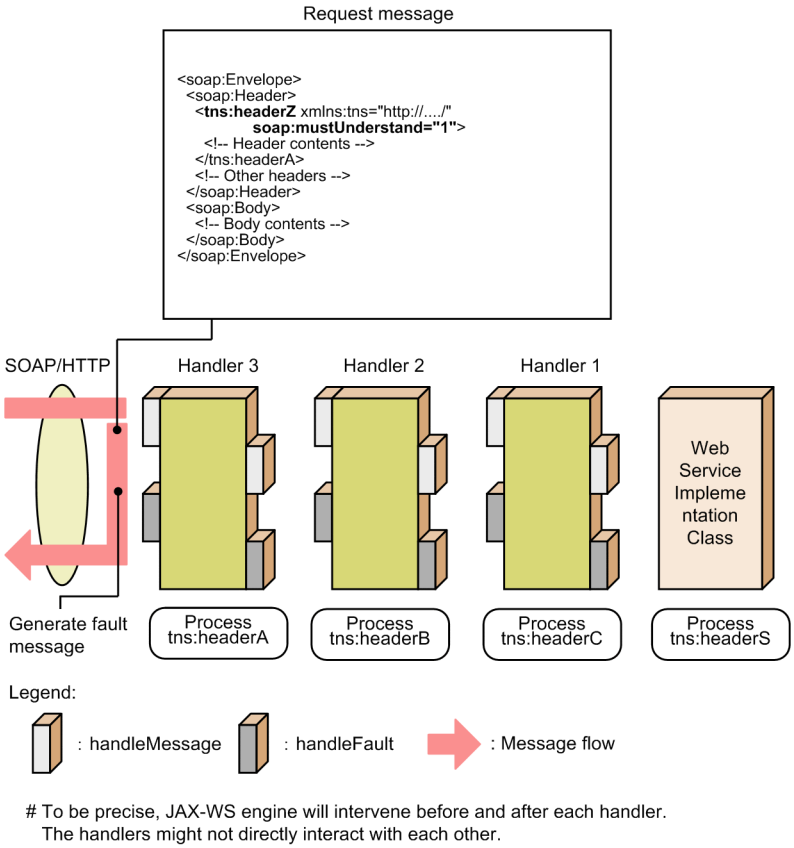
Even assuming that only the Web Service Implementation Class can process the element name `tns:headers`, all the handlers are invoked as for processing using handlers.

When a SOAP Message with a SOAP Header wherein the `soap:mustUnderstand` attribute is `1` is received, a SOAP fault with `soap:MustUnderstand` set in `faultcode` is returned in the following cases:

- One of the set handlers cannot process the SOAP Header.
- The SOAP Header cannot be processed in the Web Service Implementation Class.
- The Provider Implementation Class is used.

The following figure shows handler processing when the SOAP Header cannot be processed:

Figure 36－18: Processing of the handler when the SOAP Header cannot be processed (in the Web Service)



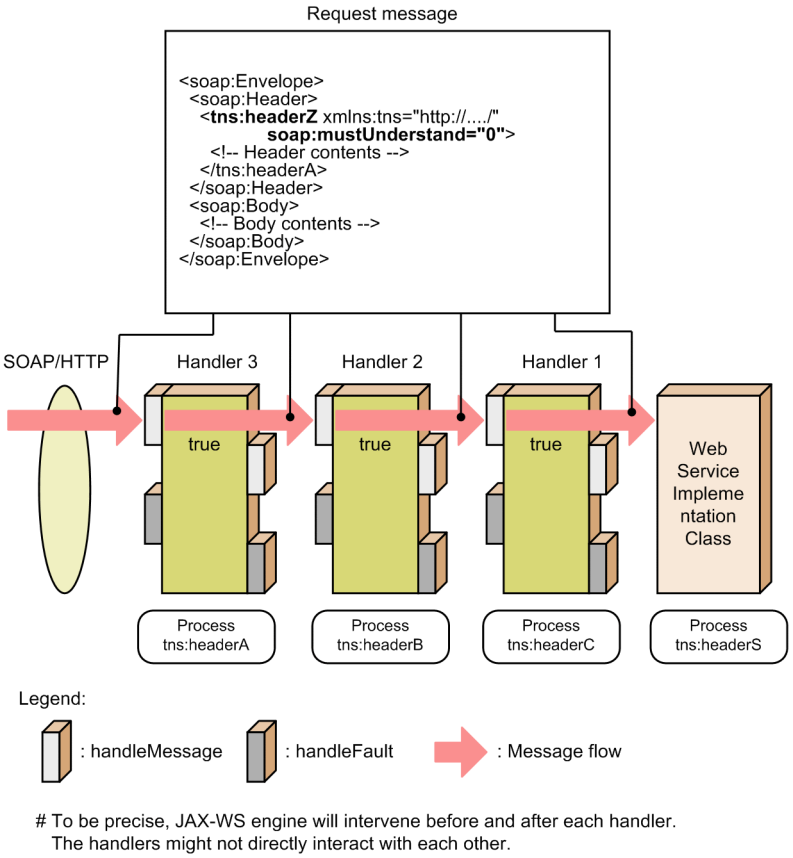
To be precise, JAX-WS engine will intervene before and after each handler.
The handlers might not directly interact with each other.

(2) When the soap:mustUnderstand attribute is not '1'

When the `soap:mustUnderstand` attribute is not `1`, all the set up handlers are invoked.

The following figure shows the handler processing when the `soap:mustUnderstand` attribute is not `1`:

Figure 36-19: Processing of the handler when the soap:mustUnderstand attribute is not '1' (in the Web Service)



To be precise, JAX-WS engine will intervene before and after each handler.
The handlers might not directly interact with each other.

In the above example, when both the handler and the Web Service Implementation Class cannot process the element `tns:headerZ`, but the `soap:mustUnderstand` attribute is not `1`, a SOAP fault does not occur and all the handlers are invoked.

36.7.2 Operations of the handler when the SOAP Header is included in the SOAP Message (in the Web Service client)

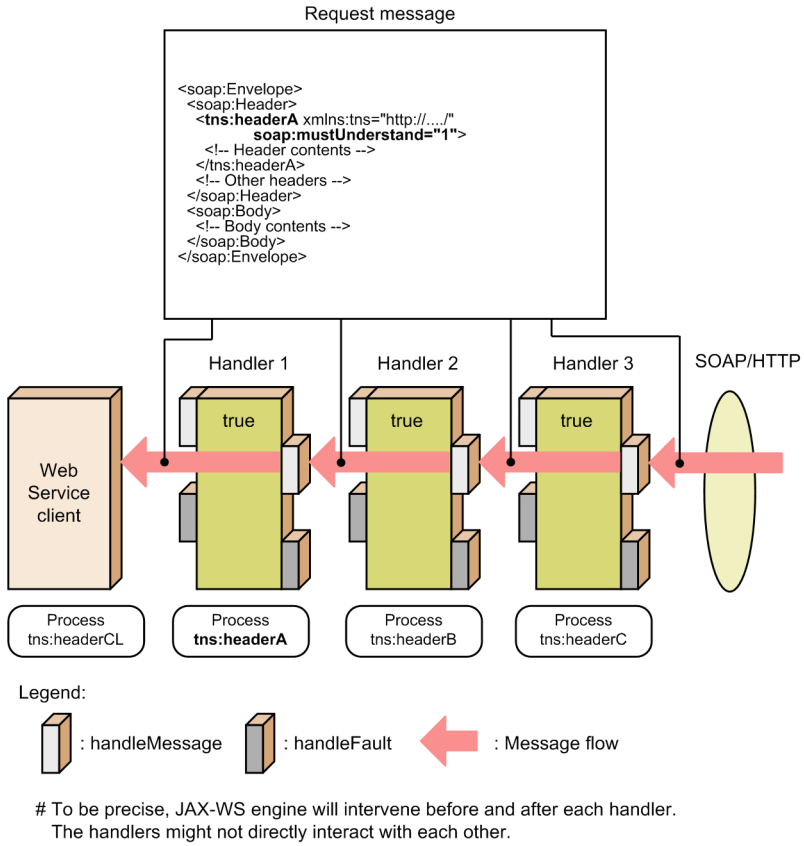
When a SOAP Message containing a SOAP Header is received, the handler processing varies according to the settings in the `soap:mustUnderstand` attribute of the SOAP Header. This subsection describes the processing contents when the `soap:mustUnderstand` attribute is `1` and when the `soap:mustUnderstand` attribute is not `1`.

(1) When the soap:mustUnderstand attribute is '1'

When a SOAP Message with a SOAP Header wherein `soap:mustUnderstand` attribute is `1` is received, all the handlers are invoked if that SOAP Header can be processed using one of the set up handlers or the Web Service client.

The following figure shows the handler processing when the SOAP Header can be processed using handlers:

Figure 36-20: Processing when the SOAP Header can be processed using handlers (in the Web Service client)

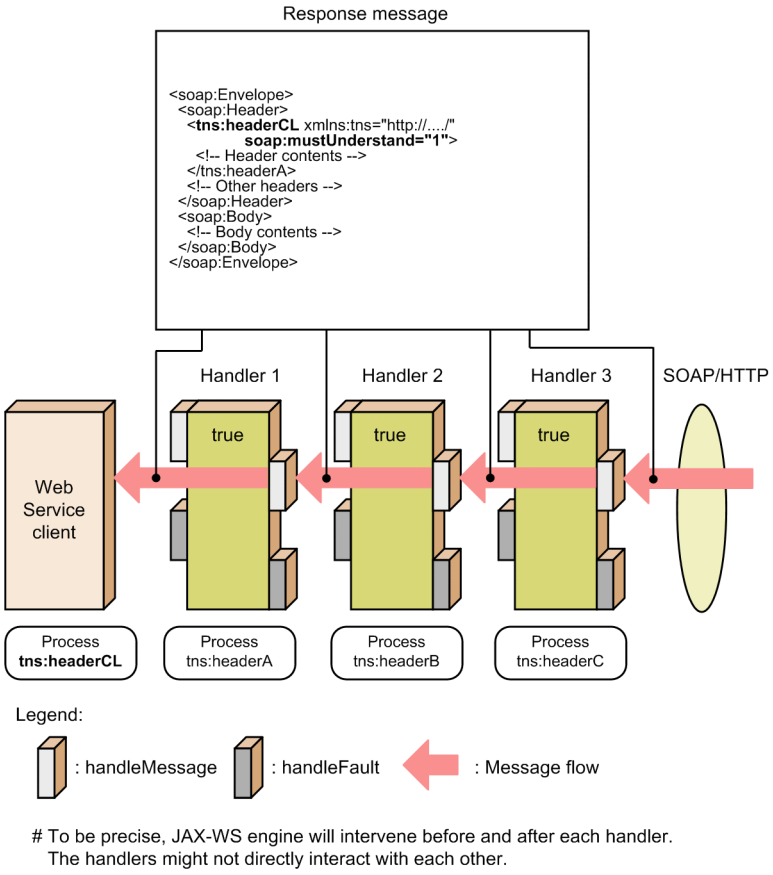


To be precise, JAX-WS engine will intervene before and after each handler.
The handlers might not directly interact with each other.

The example in the figure assumes that handler 3 that can process the element name `tns:headerA` is set up. At this time, if the SOAP Header `tns:headerA` is received, all the handlers - handler 3, handler 2, and handler 1 are invoked.

The following figure shows the handler processing when the SOAP Header can be processed using the Web Service Implementation Class:

Figure 36-21: Processing when the SOAP Header can be processed using the Web Service Implementation Class (in the Web Service client)



To be precise, JAX-WS engine will intervene before and after each handler.
The handlers might not directly interact with each other.

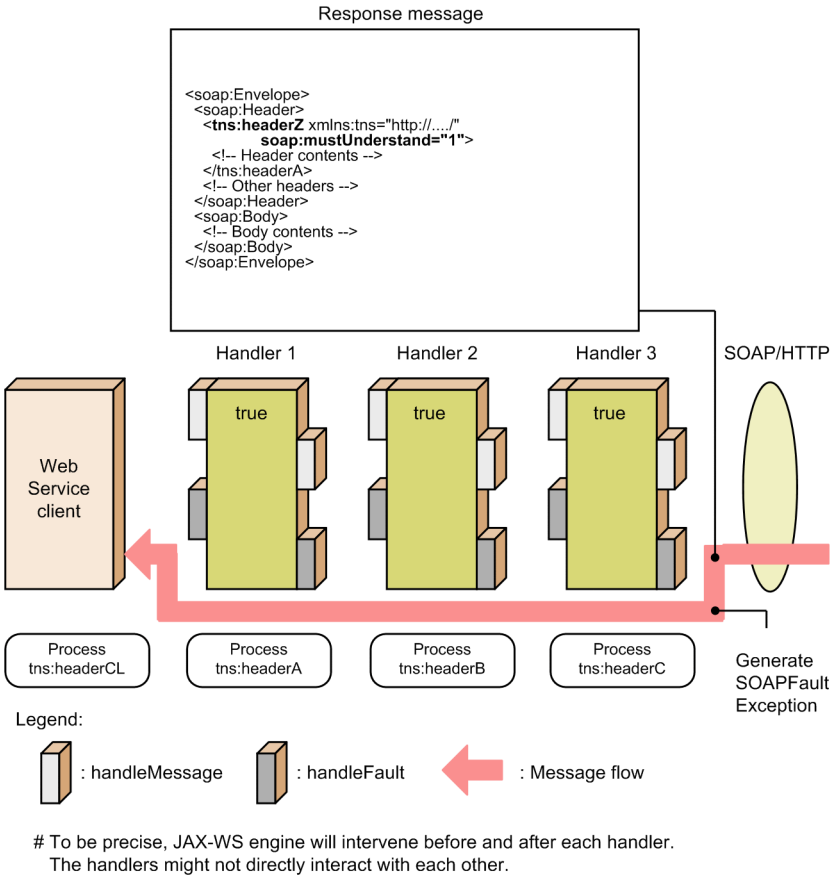
Even assuming that only the Web Service Implementation Class can process the element name `tns:headerCL`, all the handlers are invoked as for processing using handlers.

When a SOAP Message with a SOAP Header wherein `soap:mustUnderstand` attribute is `1` is received, `javax.xml.ws.soap.SOAPFaultException` is returned and an error occurs (KDJW10022-E) in the following cases:

- One of the set handlers cannot process the SOAP Header.
- The SOAP Header cannot be processed in the Web Service client.
- The Web service client is dispatch-based.

The following figure shows handler processing when the SOAP Header cannot be processed:

Figure 36-22: Processing of the handler when the SOAP Header cannot be processed (in the Web Service client)



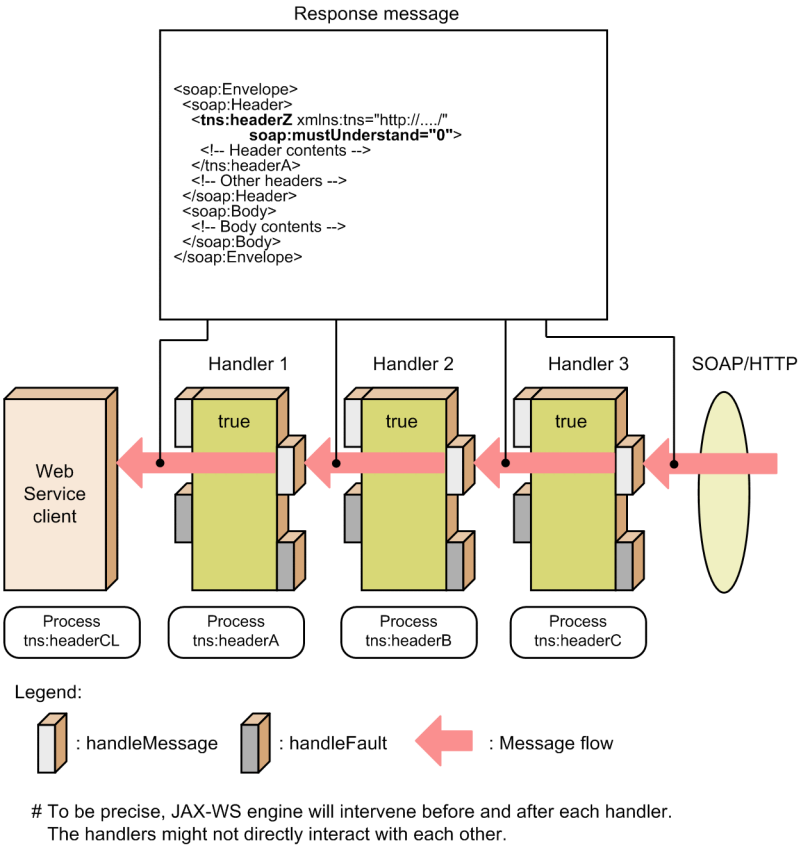
To be precise, JAX-WS engine will intervene before and after each handler.
The handlers might not directly interact with each other.

(2) When the soap:mustUnderstand attribute is not '1'

When the `soap:mustUnderstand` attribute is not `1`, all the set up handlers are invoked.

The following figure shows the handler processing when the `soap:mustUnderstand` attribute is not `1`:

Figure 36-23: Processing of the handler when the soap:mustUnderstand attribute is not '1' (in the Web Service client)



To be precise, JAX-WS engine will intervene before and after each handler.
The handlers might not directly interact with each other.

In the above example, when both the handler and the Web Service Implementation Class cannot process the element `tns:headerCL`, but the `soap:mustUnderstand` attribute is not "1", an error does not occur and all the handlers are invoked.

36.7.3 Setting the SOAP Header that can be processed

In this subsection, the method of setting up a SOAP Header that can be processed will be described with respect to the handler, the Web Service Implementation Class, and the Web Service client.

(1) To set the SOAP Header using the handler

Create the handler class implementing the `javax.xml.ws.handler.soap.SOAPHandler` interface and use the `getHeaders()` method to return `java.util.Set` containing `QName` of the SOAP Header that can be processed. The following is an example of settings in the handler:

```
public class MySOAPHandler implements SOAPHandler<SOAPMessageContext> {
 private final static Set<QName> headers;

 static {
 headers = new HashSet<QName>();
 headers.add(new QName("http://test.org/handler/", "headerA"));
 }

 public Set<QName> getHeaders(){
 return headers;
 }
 (The-rest-of-the-implementation-is-omitted)
}
```

(2) To set the SOAP Header using the Web Service Implementation Class

For the development of Web Services starting from SEI, define a method with an argument annotated using the `javax.jws.WebParam` annotation in the Web Service Implementation Class. Define the following settings in the `javax.jws.WebParam` annotation:

- Specify true in the header attribute.
- Specify `javax.jws.WebParam.Mode.IN` or `javax.jws.WebParam.Mode.INOUT` in the `mode` attribute.

The following is an example of settings in the Web Service Implementation Class:

```
@WebService
@SOAPBinding (parameterStyle= javax.jws.soap.SOAPBinding.ParameterStyle.BARE)
@HandlerChain (file="handlerchainfile.xml")
public class MyWebService{
 @WebMethod(operationName = "webMethod")
 public String webMethod(
 @WebParam(targetNamespace="http://test.org/handler/", name="message")
 String message,
 @WebParam(targetNamespace="http://test.org/handler/", name="headerS", header = true,
WebParam.Mode.IN)
 String headerS
 ){
 (The-rest-of-the-implementation-is-omitted)
}
}
```

For the development of Web Services starting from WSDL, code the `soap:header` element in WSDL within the range supported in WSDL 1.1 specifications and Cosminexus JAX-WS functionality. For details about the coding of the `soap:header` element in the Cosminexus JAX-WS functionality, see *20.1.22 soap:header element*.

(3) To set the SOAP Header using the Web Service client

If WSDL that is the Metadata of the connected Web Service, contains the `soap:header` element, that `soap:header` element can be processed. The following is an example of WSDL:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://test.org/handler/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  targetNamespace="http://test.org/handler/"
  name="HandlerTest01Service">

 <types>
 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  version="1.0"
  targetNamespace="http://test.org/handler/">
 <xs:element name="headerCL" nillable="true" type="xs:string"></xs:element>
 <xs:element name="message" nillable="true" type="xs:string"></xs:element>
 <xs:element name="testResponse" nillable="true" type="xs:string"></xs:element>
 </xs:schema>
 </types>

<message name="test">
 <part name="message" element="tns:message"></part>
 <part name="headerCL" element="tns:headerCL"></part>
 </message>
 <message name="testResponse">
 <part name="testResponse" element="tns:testResponse"></part>
 <part name="headerCL" element="tns:headerCL"></part>
 </message>

 <portType name="HandlerTest01">
 <operation name="test" parameterOrder="message headerCL">
 <input message="tns:test"></input>
 <output message="tns:testResponse"></output>
 </operation>
 </portType>

<binding name="HandlerTest01Binding" type="tns:HandlerTest01">
 <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"></
soap:binding>
```

```
 <operation name="test">
 <soap:operation soapAction=""></soap:operation>
 <input>

<soap:body use="literal" parts="message"></soap:body>
 <soap:header message="tns:test" part="headerCL" use="literal"></soap:header>
 </input>
 <output>

<soap:body use="literal" parts="testResponse"></soap:body>
 <soap:header message="tns:testResponse" part="headerCL" use="literal"></soap:header>
 </output>
 </operation>
 </binding>

<service name="HandlerTest01Service">
 <port name="HandlerTest01Port" binding="tns:HandlerTest01Binding">
 <soap:address location="http://localhost:80/SOAPHeaderTest/HandlerTestService431"></
soap:address>
 </port>
 </service>
</definitions>
```

In this case, the SOAP Header with element name `{http://test.org/handler/}headerCL` can be processed. For details about the `soap:header` element of a WSDL, see *20.1.22 soap:header element*.

36.8 Deploying the handlers

This section describes the deployment of the implementation class for the handler.

(1) In the Web Service

Deploy the handler implementation class to the following locations:

- In POJO Web Services: Deploy to the class path of the WAR file
- In EJB Web Services: Deploy to the class path of the EJB JAR file

If deployed on the class path of the WAR file or the EJB JAR file, the class need not be included in the WAR file or the EJB JAR file.

(2) In the Web Service client

Deploy the implementation class for the handler on the class path of the Web Service client.

36.9 Setting the handler chain

In the Web Service Implementation Class and Web Service client, you can dynamically use the handler chain settings.

This section describes the handler chain settings and the examples of settings.

36.9.1 Setting the handler chain in the Web Service

To set the handler chain in the Web Service, annotate the SEI, Web Service Implementation Class, or the Provider Implementation Class using the `javax.jws.HandlerChain` annotation and specify the handler chain setup file using the file element of the `javax.jws.HandlerChain` annotation.

The following is an example of annotating the Web Service Implementation Class using the `javax.jws.HandlerChain` annotation:

```
package com.sample;

@javax.jws.WebService
@javax.jws.HandlerChain(file="handlers.xml")
public class AddNumbersImpl{

 public int add( int number1, int number2 ) throws AddNumbersFault{
 ...
 }

}
```

In this example, the handler chain defined in the handler chain setup file `handlers.xml` is linked with the Web Service Implementation Class `com.sample.AddNumbersImpl`.

A handler chain setup file is required for each Web Service. A handler chain setup file might also be shared by multiple Web Services, but in that case, the same settings are applied.

For details about the `javax.jws.HandlerChain` annotation, see the subsection *16.2.3 javax.jws.HandlerChain annotation*.

(1) Handler chain setup file

The handler chain setup file defines the handler chain configuration when the handler is used to add processing for the Web Service. The file is specified using the file element of the `javax.jws.HandlerChain` annotation.

For details about the support range of the handler chain setup file, see the subsection *19.4 Support range of the handler chain setup file*. This point describes the syntax and deployment of the handler chain setup file.

(a) Syntax of the handler chain setup file

Code the syntax of the handler chain setup file in the range supported by the standard schema of Java EE 5 specifications (Java EE Web Services Metadata Handler Chain Schema) and the Cosminexus JAX-WS functionality.

You can reference the standard schema by accessing the namespace URI of Java EE 5 (`http://java.sun.com/xml/ns/javaee/`). The following is an example of coding of the handler chain setup file:

```
<?xml version="1.0" encoding="UTF-8"?>
<handler-chains xmlns="http://java.sun.com/xml/ns/javaee">
 <handler-chain>
 <handler>
 <handler-class>org.test.handler.LoggingHandler</handler-class>
 </handler>
 </handler-chain>
</handler-chains>
```

(b) Deployment of the handler chain setup file

Store the handler chain setup file at the following locations:

- In POJO Web Services: Under the WEB-INF directory of WAR file.

- In EJB Web Services: EJB JAR

The deployment of the file complies with the file system limitations for the OS and Java EE 5 specifications, and if the file can be stored in the WAR file or EJB JAR file, there are no limitations on the file name, the directory name below the WEB-INF directory, EJB JAR directory name, and the path length.

(2) SOAP role and actor settings

If the SOAP role and actor settings are omitted or if `null` is specified, the default value is specified. The default value is `http://schemas.xmlsoap.org/soap/actor/next` for a SOAP Messages in the SOAP 1.1 specification and `http://www.w3.org/2003/05/soap-envelope/role/ultimateReceiver` for the SOAP Messages in the SOAP 1.2 specification.

(3) soap:mustUnderstand attribute settings

Set up `0`, `1`, `true`, or `false` in the `soap:mustUnderstand` attribute. If another value is specified, `0` or `false` is assumed.

36.9.2 Setting the handler chain in the Web Service client

To set the handler chain in the Web Services client, use the JAX-WS API. For the details on the available JAX-WS APIs, see *19.2 Support Range of API*.

(1) Add Code that sets the handler chain

The following methods are used to set a handler chain on a Web Services Client:

- Setting the handler chain to a service class
- Setting the handler chain to a port

This section describes the methods of setting the handler chain.

(a) Setting the handler chain to a service class

The following example describes the code added to the Web Services Client to set the handler chain to a service class by using an API:

```
package com.example.sample.client;

import com.example.sample.TestJaxWs;
import com.example.sample.TestJaxWsService;
import com.example.sample.UserDefinedException;

public class TestClient {
 public static void main( String[] args ) {
 try {
  TestJaxWsService service = new TestJaxWsService();
  // Handler-resolver-is-generated-and-set-in-the-service-class
  SampleHandlerResolver handlerResolver = new SampleHandlerResolver();
  service.setHandlerResolver( handlerResolver );
  TestJaxWs port = service.getTestJaxWs();

  String returnValue = port.jaxWsTest1( "Invocation test.", 1003 );

  System.out.println( "[RESULT] " + returnValue );
 }
 catch( UserDefinedException e ){
  e.printStackTrace();
 }
 }
}
```

The following is an example of a handler resolver:

```
package com.example.sample.client;
```

```
import java.util.ArrayList;
import java.util.List;

import javax.xml.ws.handler.HandlerResolver;
import javax.xml.ws.handler.PortInfo;
import javax.xml.ws.soap.SOAPBinding;

public class SampleHandlerResolver implements HandlerResolver{
 //Handler-chain-stored-by-this-handler-resolver
 private List<Handler> handlerChain = new ArrayList<Handler>();

 //Use-a-convenient-method-to-add-the-handler-to-the-handler-chain-in-advance
 //The-constructor-is-used-to-add-the-handler-here
 public SampleHandlerResolver(){
 this.handlerChain.add( new SomeHandler() );
 }

 //Returns-the-handler-chain
 public List<Handler> getHandlerChain( PortInfo portInfo ){
 //Check-the-contents-of-the-portInfo-object
 //If-the-portInfo-object-can-be-processed-by-this-handler-resolver-the-handler-chain-
is-returned
 if( portInfo.getBindingID().equals( SOAPBinding.SOAP11HTTP_BINDING )
  && portInfo.getPortName().equals( ... )
  && portInfo.getServiceName().equals( ... ) ){
  return this.handlerChain;
 }
 else{
  ...
 }
 }
}
```

The changes in the handler resolver set up using the `javax.xml.ws.Service#setHandlerResolver` method do not affect the handler chains of ports previously acquired from the same Service object.

(b) Setting the handler chain to a port

The following example shows a Web Services client to which a code is added to set the handler chain to a port by using an API:

```
package com.example.sample.client;

import java.util.ArrayList;
import java.util.List;

import javax.xml.ws.Binding;
import javax.xml.ws.BindingProvider;
import javax.xml.ws.handler.Handler;

import com.example.sample.TestJaxWs;
import com.example.sample.TestJaxWsService;
import com.example.sample.UserDefinedException;

public class TestClient {
    public static void main( String[] args ) {
        try {
            TestJaxWsService service = new TestJaxWsService();
            TestJaxWs port = service.getTestJaxWs();

// Generate a handler chain to be set to a port

            List<Handler> handlerChain = new ArrayList<Handler>();
            //Add a handler to the handler chain
            handlerChain.add( new SomeHandler() );

            //Acquire javax.xml.ws.Binding
            Binding binding = ( ( BindingProvider )port ).getBinding();

// Set the handler chain to a port by using javax.xml.ws.Binding

            binding.setHandlerChain(handlerChain);

            String returnValue = port.jaxWsTest1( "Invocation test.", 1003 );
            System.out.println( "[RESULT] " + returnValue );
        }
        catch( UserDefinedException e ){
            e.printStackTrace();
        }
```

```
        }
    }
```

If you set the handler chain in both the service classes and ports acquired from the service classes, the handler chain set in the ports is used. Also, the handler chain set by the `javax.xml.ws.Binding#setHandlerChain` method does not affect the handler chain of the `Service` object from which the ports are to be acquired.

(2) SOAP role and actor settings

If the SOAP role and actor settings are omitted or if `null` is specified, the default value is set up. The default value is `http://schemas.xmlsoap.org/soap/actor/next` for a SOAP Message in the SOAP 1.1 specification and `http://www.w3.org/2003/05/soap-envelope/role/ultimateReceiver` for a SOAP Message in the SOAP 1.2 specification.

(3) soap:mustUnderstand attribute settings

Set up `0`, `1`, `true`, or `false` in the `soap:mustUnderstand` attribute. If another value is specified, `0` or `false` is assumed.

37 Addressing Functionality

This chapter describes the communication methods which can be implemented with the Cosminexus addressing functionality, and the settings for using the Cosminexus addressing functionality.

37.1 Addressing functionality

The *Addressing functionality* is used to standardize the information such as server names and port numbers given by the general transfer protocol and communication system, and make the Web Services and messages independent from a specific transport or communication system. The addressing functionality supports two types of communication methods; synchronous communication and asynchronous communication.

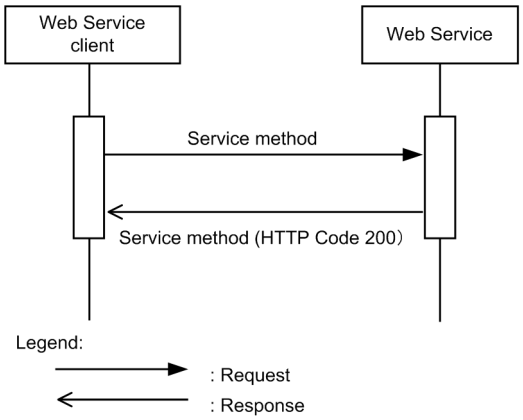
37.1.1 Synchronous communication

(1) Flow of synchronous communication

In the synchronous communication, the Web Service client sends a request message for invoking a service method in a Web Service, and receives the resulting response message from the Web Service.

The following figure shows the flow of synchronous communication:

Figure 37－1: Flow of synchronous communication



(2) Settings for synchronous communication

To use the synchronous communication, set up the anonymous URI `http://www.w3.org/2005/08/addressing/anonymous` defined in the WS-Addressing 1.0 specifications, with the `wsa:ReplyTo/wsa:Address` element of the addressing header.

The following are the examples of the contents specified in the addressing header with the conditions:

- Web Service URI
  `http://localhost/addressing/AddNumbersImplService`
- `Action` value for invoking the Web Service
  `http://sample.com/input`
- Anonymous URI
  `http://www.w3.org/2005/08/addressing/anonymous`
- Unique ID indicating this message
  `uuid:4cfb4248-a552-4e33-a610-6af94f2aad07`

```
<To xmlns="http://www.w3.org/2005/08/addressing">
 http://localhost/addressing/AddNumbersImplService
</To>
<Action xmlns=http://www.w3.org/2005/08/addressing>
 http://sample.com/input
</Action>
<ReplyTo xmlns="http://www.w3.org/2005/08/addressing">
 <Address>http://www.w3.org/2005/08/addressing/anonymous</Address>
</ReplyTo>
<MessageID xmlns="http://www.w3.org/2005/08/addressing">
```

```
  uuid:4cfb4248-a552-4e33-a610-6af94f2aad07
</MessageID>
```

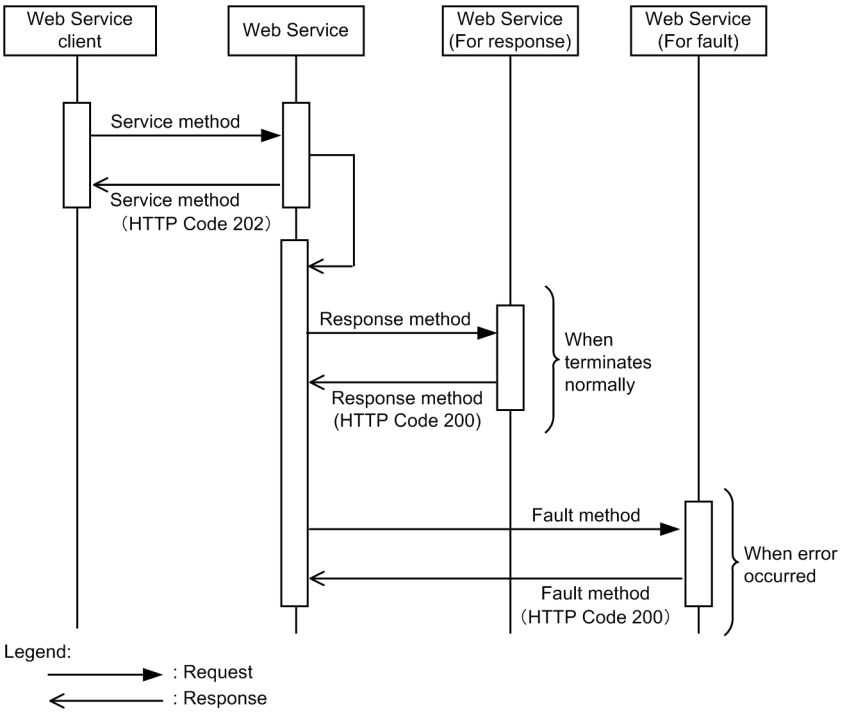
37.1.2 Asynchronous communication

(1) Flow of asynchronous communication

In the asynchronous communication using the addressing functionality, the Web Service client sends a request message for invoking a service method that exists in a Web Service, and then ends the processing without receiving the resulting response message from the Web Service. The Web Service sends the response message to another Web Service.

The following figure shows the flow of asynchronous communication:

Figure 37-2: Flow of asynchronous communication



(2) Settings for asynchronous communication

To use the asynchronous communication, set up the Web Service URL that sends response messages in the `wsa:ReplyTo/wsa:Address` and `wsa:FaultTo/wsa:Address` elements of the addressing header.

The following are the examples of the contents specified in the addressing header with the conditions:

- Web Service URI
  `http://localhost/addressing/AddNumbersImplService`
- `Action` value for invoking the Web Service
  `http://sample.com/input`
- URL for sending the response message when the Web Service terminates successfully
  `http://localhost/responseserver/ResponseServerImplService`
- URL for sending the response message when the Web Service terminates abnormally
  `http://localhost/responseserver/FaultServerImplService`
- Unique ID indicating this message
  `uuid:b19439fa-7a29-4045-93d9-56d6a2183afd`

```
<To xmlns="http://www.w3.org/2005/08/addressing">
 http://localhost/addressing/AddNumbersImplService
</To>
<Action xmlns="http://www.w3.org/2005/08/addressing">
 http://sample.com/input
</Action>
<ReplyTo xmlns="http://www.w3.org/2005/08/addressing">
 <Address>
 http://localhost/responseserver/ResponseServerImplService
 </Address>
</ReplyTo>
<FaultTo xmlns="http://www.w3.org/2005/08/addressing">
 <Address>
 http://localhost/responseserver/FaultServerImplService
 </Address>
</FaultTo>
<MessageID xmlns="http://www.w3.org/2005/08/addressing">
 uuid:b19439fa-7a29-4045-93d9-56d6a2183afd
</MessageID>
```

(3) Precautions for asynchronous communication

This subsection describes the precautions for using the asynchronous communication.

- Even if the Web Service fails to send the response message, no error is returned to the Web Service client.
- In the `wsa:ReplyTo/wsa:Address` and `wsa:FaultTo/wsa:Address` elements of the addressing header, if you set up the same URL as you would set up for the Web Service specified in the `wsa:To` element, the sending source and destination of the response message become the same and the communication might go into an infinite loop.
- If the JAX-WS engine of the Web Service machine receives a fault message as the request message, the fault message is processed as a server error (HTTP code: 500). Therefore, you cannot create a Web Service that receives a fault message as a request message.
- You cannot use the asynchronous communication through a proxy. To use the asynchronous communication, send the response message to another Web Service without passing through a proxy.

37.2 WSDL extension elements and extension attributes

The WS-Addressing 1.0 specifications define two extension elements and one extension attribute for coding in the WSDL. This section describes the WSDL extension elements and the extension attribute.

37.2.1 WSDL extension elements

The following WSDL extension elements of the WS-Addressing 1.0 specifications are available with Cosminexus:

(1) `wsaw:UsingAddressing` element

This element indicates whether the addressing functionality is enabled using Web Services.

You can code the `wsaw:UsingAddressing` element as a child element of the `wsdl:definitions/wsdl:binding` or `wsdl:definitions/wsdl:service/wsdl:port` element. The addressing functionality is enabled in the Web Service used for coding this element.

- `wsdl:required` attribute

  This attribute indicates whether an addressing header is required for a request message.

  If `true` is specified in this attribute, an addressing header is required. If `false` is specified, an addressing header is optional.

Note that you cannot code an attribute belonging to a child element or the name space `http://www.w3.org/2006/05/addressing/wsdl` in the `wsaw:UsingAddressing` element. If such an attribute is coded, a standard error is output and an error message is output to logs during the `cjwsimport` command is executed, and then the processing of the `cjwsimport` command ends (KDJW51029-E).

(2) `wsaw:Anonymous` element

This element indicates whether an anonymous URI can be used in response end points (`wsa:From/wsa:Address`, `wsa:ReplyTo/wsa:Address`, and `wsa:FaultTo/wsa:Address` elements) of an addressing header.

You can code the `wsaw:Anonymous` element as a child element of the `wsdl:definitions/wsdl:binding/wsdl:operation` element. You can specify the following values:

- `optional`

  Using an anonymous URI in a response end point of a request message is optional.
- `required`

  An anonymous URI must always be used as a response end point of a request message.
- `prohibited`

  Do not use an anonymous URI as a response end point of a request message.

You can specify only one of the above values in the `wsaw:Anonymous` element. If another value is specified, a standard error is output and an error message is output to logs when the `cjwsimport` command is executed, and then the processing of the `cjwsimport` command ends (KDJW51029-E). Also, you cannot code an attribute belonging to a child element or the namespace `http://www.w3.org/2006/05/addressing/wsdl` in the `wsaw:Anonymous` element. If such an attribute is coded, a standard error is output and an error message is output to logs when the `cjwsimport` command is executed, and then the processing of the `cjwsimport` command ends (KDJW51029-E).

Note that this element is not coded in the WSDL file issued by the JAX-WS engine of a Web Service machine and the WSDL file generated using the `cjwsgen` command. Prepare a WSDL file coding this element to control the operations, when an anonymous URI is specified.

(3) `wsam:Addressing` element

The `wsam:Adressing` element indicates whether the WS-Addressing functionality is enabled in Web Services. With the JAX-WS 2.2 specifications (WS-Addressing 1.0 meta data), you use this element instead of the

wsaw:UsingAddressing element. When both the wsam:Adressing and wsaw:UsingAddressing elements are specified simultaneously, you indicate that an addressing header is required in a request message by specifying that the addressing header is required in either of the elements.

You can code the wsam:Adressing element as a child element of the wsdl:definitions/wsdl:binding or wsdl:definitions/wsdl:service/wsdl:port element. When you code the wsam:Adressing element, the settings of this element are given priority.

- wsp:Optional attribute

  The wsp:Optional attribute indicates whether an addressing header is required in a request message.

  If true is specified in this attribute, an addressing header is optional. If false is specified or this attribute is omitted, an addressing header is required.

(4) wsam:AnonymousResponses element

The wsam:AnonymousResponse element indicates that you must specify an anonymous URI in response end points (wsa:From/wsa:Address, wsa:ReplyTo/wsa:Address, and wsa:FaultTo/wsa:Address elements) for a request message containing an addressing header.

You can code this element as a child element of wsdl:definitions/wsdl:binding/wsam:Addressing/wsp:Policy or wsdl:definitions/wsdl:service/wsdl:port/wsam:Addressing/wsp:Policy.

(5) wsam:NonAnonymousResponses element

The wsam:NonAnonymousResponses element indicates that you must specify a non-anonymous URI in response end points (wsa:From/wsa:Address, wsa:ReplyTo/wsa:Address, and wsa:FaultTo/wsa:Address elements) for a request message containing an addressing header.

You can code this element as a child element of wsdl:definitions/wsdl:binding/wsam:Addressing/wsp:Policy or wsdl:definitions/wsdl:service/wsdl:port/wsam:Addressing/wsp:Policy.

(6) Notes for WSDL extension elements

This subsection describes the notes for using the following WSDL extension element:

wsaw:Anonymous element

The wsaw:Anonymous element is not indicated in the WSDL file published by the JAX-WS engine of a service machine or the WSDL file generated by the cjwsgen tool. To control the operations when an anonymous URI is specified by using the wsaw:Anonymous element, you must prepare a WSDL file with the coded wsaw:Anonymous element.

37.2.2 WSDL extension attributes

This subsection describes the WSDL extension attribute of the WS-Addressing 1.0 specifications available with Cosminexus:

- **wsaw:Action attribute**

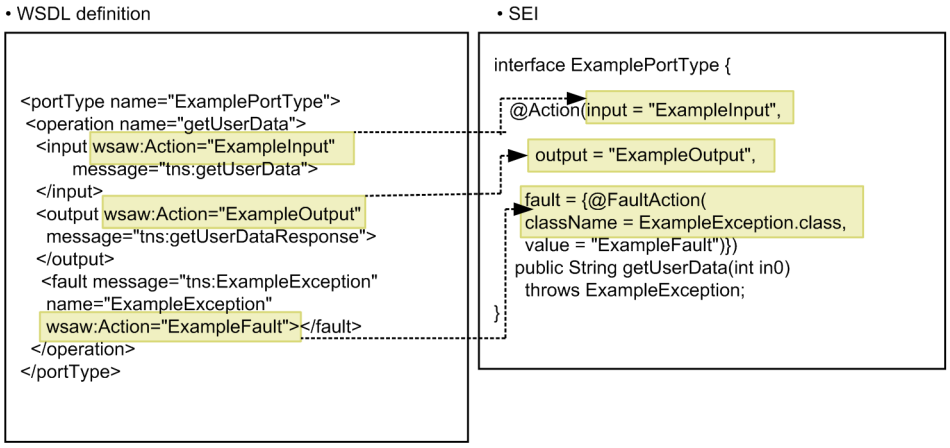
  This extension attribute is used to bind the Action value of the addressing header with the wsdl:input, wsdl:output and wsdl:fault elements.

  You can code the wsaw:Action attribute as the extension attribute of the wsdl:input, wsdl:output, and wsdl:fault elements, the child elements of the wsdl:portType/wsdl:operation element. If you code this attribute as the extension attribute of other elements, the value is ignored. If a space is specified in this attribute, the space is used as the Action value as it is. Also, if null ("") is specified, the wsaw:Action attribute is ignored.

  If this attribute is coded in WSDL, the javax.xml.ws.Action and javax.xml.ws.FaultAction annotations are created in SEI that is generated using the cjwsimport command. For details on the javax.xml.ws.Action annotation, see the section *16.2.13 javax.xml.ws.Action annotation* and for details on the javax.xml.ws.FaultAction annotation, see the section *16.2.15 javax.xml.ws.FaultAction annotation*.

  The following figure shows an example of mapping, when the wsaw:Action attribute is used.

Figure 37-3: Example of mapping when the wsaw:Action attribute is used



37.3 Notes for the annotations used with the addressing functionality

This section describes the precautions for using the addressing functionality.

- Specify the `javax.xml.ws.soap.Addressing` annotation in the service implementation class.
- The `javax.xml.ws.soap.Addressing` annotation is not mapped to the skeleton class of the service implementation class that is generated by executing the `cjwsimport` command. Therefore, to use the skeleton class of the service implementation class, you must specify the `javax.xml.ws.soap.Addressing` annotation.

For details on the annotations available with the addressing functionality, see the section *16.2.1 List of annotations*.

37.4 Fault messages

This section describes the fault messages issued, when using the addressing functionality.

37.4.1 Un-supported sub-sub code

In the WS-Addressing 1.0 specifications, the sub-sub code used in fault messages are defined in advance. Among the sub-sub codes provided beforehand, Cosminexus does not support the following sub-sub codes:

- `wsa:InvalidEPR`
- `wsa:DuplicateMessageID`
- `wsa:ActionMismatch`#

#

Supported with the SOAP 1.1 specifications.

37.4.2 Notes for fault messages

This subsection describes the precautions for using the addressing functionality.

- Fault message containing `wsa:MessageAddressingHeaderRequired` in the sub-code

  If the JAX-WS engine of the Web Service machine sends a fault message #1 containing `wsa:MessageAddressingHeaderRequired` in the sub-code, set up the same value (`wsa:MessageAddressingHeaderRequired`) as the sub-code in the sub-sub code of the fault message.
- Fault message containing `wsa:ActionNotSupported` in the sub-code

  If the Web Service-side JAX-WS engine sends a fault message #2 containing `wsa:ActionNotSupported` in the sub-code, no value is set up in the sub-sub code of the fault message.

#1

Fault message without a required element in the addressing header

#2

Fault message in which the value of the `wsa:Action` element of the addressing header differs from the `wsa:Action` value in the Web Service

37.5 Operations of the JAX-WS engine on a Web Service machine (When using the addressing functionality)

This section describes the operations of the JAX-WS engine of the Web Service machine, when using the addressing functionality.

37.5.1 Operations for receiving request messages

When you use the addressing functionality, the operations for receiving request messages depend on the contents of the elements set up in the `javax.xml.ws.soap.Addressing` annotation specified in the service implementation class. The following table lists and describes the relationship between the `javax.xml.ws.soap.Addressing` annotation and the operations for receiving request messages.

Table 37－1: javax.xml.ws.soap.Addressing annotation and operations for receiving request messages

No.	javax.xml.ws.soap.Addressing annotation		Operation when a request message is received		
			Addressing header		Communication
	`enabled` element	`required` element	Request message	Response message	
1	`true`	`true`	Y	Y	Successful
2			N	N (Fault message)	Failure
3		`false`	Y	Y	Successful
4			N	N	Successful
5	`false`	`true`	Y	N	Successful
6			N	N	Successful
7		`false`	Y	N	Successful
8			N	N	Successful

Legend:

Y: Addressing header exists.

N: Addressing header does not exist.

If the `javax.xml.ws.soap.Addressing` annotation is not specified in the service implementation class, the addressing functionality is disabled. In such a case, irrespective of the presence or absence of the addressing header, the Web Service receives the request message. Also, since the addressing functionality is disabled, the Web Service sends a response message without the addressing header even if the received request message has the addressing header specified. Because the response messages do not exist for one-way operations, the addressing functionality in which the one-way operations are used is not supported. The operation is not guaranteed if you use the addressing functionality with one-way operations.

37.5.2 Response messages

This subsection describes the response messages.

(1) Sending destination for the response messages

If a value other than an anonymous URI is used in the response end point of the addressing header included in the request message, the sending destination of the response message differs depending on the availability of the `wsa:From/wsa:Address`, `wsa:ReplyTo/wsa:Address`, and `wsa:FaultTo/wsa:Address` elements.

The following table lists and describes the relationship between availability of the elements and the sending destination of the response messages:

Table 37-2: Sending destination of the response messages

No.	Addressing header			Type of response message to be sent	Sending destination of the response message
	wsa:From/ wsa:Address element	wsa:ReplyTo/ wsa:Address element	wsa:FaultTo/ wsa:Address element		
1	Does not exist	Does not exist	Does not exist	Normal message	HTTP sending source
2				Abnormal message	HTTP sending source
3			Exists	Normal message	HTTP sending source
4				Abnormal message	`wsa:FaultTo/ wsa:Address` element
5		Exists	Does not exist	Normal message	`wsa:ReplyTo/ wsa:Address` element
6				Abnormal message	`wsa:ReplyTo/ wsa:Address` element
7			Exists	Normal message	`wsa:ReplyTo/ wsa:Address` element
8				Abnormal message	`wsa:FaultTo/ wsa:Address` element
9	Exists	Does not exist	Does not exist	Normal message	HTTP sending source
10				Abnormal message	HTTP sending source
11			Exists	Normal message	HTTP sending source
12				Abnormal message	`wsa:FaultTo/ wsa:Address` element
13		Exists	Does not exist	Normal message	`wsa:ReplyTo/ wsa:Address` element
14				Abnormal message	`wsa:ReplyTo/ wsa:Address` element
15			Exists	Normal message	`wsa:ReplyTo/ wsa:Address` element

No.	Addressing header			Type of response message to be sent	Sending destination of the response message
	wsa:From/ wsa:Address element	wsa:ReplyTo/ wsa:Address element	wsa:FaultTo/ wsa:Address element		
16	Exists	Exists	Exists	Abnormal message	`wsa:FaultTo/ wsa:Address` element

(2) Operations when http://www.w3.org/2005/08/addressing/none is specified

`http://www.w3.org/2005/08/addressing/none` is a URI indicating that the message is not sent using the WS-Addressing 1.0 specifications. If this URI is set up in the `wsa:ReplyTo/wsa:Address` and `wsa:FaultTo/wsa:Address` elements of the addressing header, the response message is not sent.

37.5.3 Operations when the wsaw:Anonymous element is specified

If both the elements; `wsaw:UsingAddressing` and `wsaw:Anonymous` are coded in WSDL, the receipt of the request message might fail and a fault message might be returned depending on the response end point value in the addressing header of the received request message. The following table lists and describes the relationship between the `wsaw:Anonymous` element and the operations of the Web Service-side JAX-WS engine:

Table 37–3: wsaw:Anonymous element and the operations of the Service-side JAX-WS engine

No.	Value of the wsaw:Anonymous element	Response end point value	Operation of the Web Service-side JAX-WS engine
1	`optional`	Anonymous URI	Normal termination
2		Non-anonymous URI	Normal termination
3	`required`	Anonymous URI	Normal termination
4		Non-anonymous URI	Reception failure (fault message)
5	`prohibited`	Anonymous URI	Reception failure (fault message)
6		Non-anonymous URI	Normal termination

For the values specified in the `wsaw:Anonymous` element, see the subsection *37.2.1 WSDL extension elements*.

37.5.4 Operations when an Addressing annotation is specified

If the `javax.xml.ws.soap.Addressing` annotation is specified in the service implementation class, the WSDL file issued by the Web Service-side JAX-WS engine and the WSDL file generated by executing the `cjwsgen` command become as follows:

- In the WSDL file, the `wsaw:Action` attribute is not only given to the `wsaw:UsingAddressing` element, but also to the `wsdl:input` element. The given value is the default `Action` value defined in the WS-Addressing 1.0 specifications. For details about the default `Action` value, reference the WS-Addressing 1.0 specifications.
- If `false` is specified in the `required` element of the `javax.xml.ws.soap.Addressing` annotation, the `wsdl:required` attribute of the `wsaw:UsingAddressing` element is not given in the WSDL file.

37.5.5 Operations when an Action annotation is specified

If you specify the `javax.xml.ws.FaultAction` annotation indicating a class other than the exception class declared using the `throws` clause of the method in the fault element of the `javax.xml.ws.Action` annotation specified in the SEI method, the `javax.xml.ws.FaultAction` annotation is ignored.

37.5.6 Operations when the wsa:Action element is specified

This subsection describes the values of the `wsa:Action` element and the precautions for specifying the `wsa:Action` element.

(1) Values of the wsa:Action element

The values of the `wsa:Action` element of the addressing header change according to the following conditions:

- Is the `javax.xml.ws.Action` annotation specified in the SEI method?
- Is the `wsaw:Action` attribute coded in the WSDL?

The following table lists and describes the relationship between the `javax.xml.ws.Action` annotation and `wsaw:Action` attribute and the values of the `wsa:Action` element:

Table 37-4: Values of the wsa:Action element

No.	Action annotation	wsaw:Action attribute	Value of the wsa:Action element
1	Specified	Coded	Value of the `Action` annotation
2		Not coded	Value of the `Action` annotation
3	Not specified	Coded	Value of the `wsaw:Action` attribute of WSDL
4		Not coded	Default `Action` value defined in the WS-Addressing 1.0 specifications

Specify the value of the `wsa:Action` element of the addressing header used by the Web Service for response or fault in the asynchronous communication as described above.

(2) Precautions for receiving the request message

This point describes the precautions for receiving the request messages.

- If the value of the `wsa:Action` element differs in the addressing header of the received request message and the addressing header used by the Service-side JAX-WS engine, the attempt to invoke the Web Service will fail. An error occurs in the Web Service-side JAX-WS engine and a fault message containing the sub-sub code `wsa:ActionNotSupported`, defined in the WS-Addressing 1.0 specifications, is sent.
- If the value of the `wsa:Action` element of the addressing header in the received request message and the value of `SOAPAction` is different, an error occurs. However, if the value of `SOAPAction` is null (`""`), an error does not occur even if the value of `wsa:Action` element is not null. If an error occurs, the Web Service-side JAX-WS engine sends a fault message containing the sub-sub code `wsa:ActionMismatch`, defined in the WS-Addressing 1.0 specifications.

37.5.7 Operations when the wsa:MessageID element is not specified

An addressing header that does not contain the `wsa:MessageID` element is not considered normal. Therefore, an error occurs in the JAX-WS engine of the Web Service machine and a fault message containing the subcode `wsa:MessageAddressingHeaderRequired`, defined in the WS-Addressing 1.0 specifications is sent.

37.6 Operations of the JAX-WS engine on a Web Service client machine (When using the addressing functionality)

This section describes the operations of the JAX-WS engine of the Web Service client machine, when using the Addressing Function.

37.6.1 Operations for sending and receiving messages

The addressing functionality follows the settings of a WSDL mapped with an acquired SEI. However, if SEI is acquired using the `AddressingFeature` class, the settings of the `AddressingFeature` class are given priority, therefore you can set up the operations for sending and receiving messages with the arguments that are specified when generating the `AddressingFeature` class.

The following table lists and describes the relationship between the `AddressingFeature` class and the operations for sending messages:

Table 37-5: AddressingFeature class and operations for sending messages

No.	AddressingFeature class			Operation for sending request messages
	enabled	required	responses	
1	`true`	`True`	`Responses.ALL`	Y
			`Responses.ANONYMOUS`	
			`Responses.NON_ANONYMOUS`	
2		`False`	`Responses.ALL`	Y
			`Responses.ANONYMOUS`	
			`Responses.NON_ANONYMOUS`	
3	`false`	`True`	`Responses.ALL`	N
			`Responses.ANONYMOUS`	
			`Responses.NON_ANONYMOUS`	
4		`False`	`Responses.ALL`	N
			`Responses.ANONYMOUS`	
			`Responses.NON_ANONYMOUS`	

Legend:

Y: The addressing or header is given to the message.

N: The addressing or header is not given to the message.

The following table lists and describes the relationship between the `AddressingFeature` class and the operations for receiving messages:

Table 37-6: AddressingFeature class and operations for receiving messages

No.	AddressingFeature class			Operation for receiving response message	
	enabled	required	responses	Addressing header	Communication
1	`true`	`true`	`Responses.ALL`	Y	Successful

No.	AddressingFeature class			Operation for receiving response message	
	enabled	required	responses	Addressing header	Communication
1	true	true	Responses.ANONYMOUS	Y	Successful
			Responses.NON_ANONYMOUS		
2			Responses.ALL	N	Failure
			Responses.ANONYMOUS		
			Responses.NON_ANONYMOUS		
3		false	Responses.ALL	Y	Successful
			Responses.ANONYMOUS		
			Responses.NON_ANONYMOUS		
4			Responses.ALL	N	Successful
			Responses.ANONYMOUS		
			Responses.NON_ANONYMOUS		
5	false	true	Responses.ALL	Y	Successful
			Responses.ANONYMOUS		
			Responses.NON_ANONYMOUS		
6			Responses.ALL	N	Successful
			Responses.ANONYMOUS		
			Responses.NON_ANONYMOUS		
7		false	Responses.ALL	Y	Successful
			Responses.ANONYMOUS		
			Responses.NON_ANONYMOUS		
8			Responses.ALL	N	Successful
			Responses.ANONYMOUS		
			Responses.NON_ANONYMOUS		

Legend:

Y: The addressing header exists.

N: The addressing header does not exist.

#

The settings of a `responses` attribute of the `AddressingFeature` class do not impact the operations of the JAX-WS engine on a client machine.

37.6.2 AddressingFeature class and anonymous URI

When you use the `AddressingFeature` class to acquire SEI, the JAX-WS engine of the Web Service client machine sets up an anonymous URI in the `wsa:ReplyTo/wsa:Address` element of the addressing header. At this time, if you invoke the following Web Service, the anonymous URI cannot be used, so `WebServiceException` occurs when invoking a service method:

- a Web Service wherein `prohibited` is specified in the `wsa:Anonymous` element of a WSDL

- a Web Service wherein the `wsam:NonAnonymousResponses` element is specified in the child element of the `wsam:Addressing` element

To communicate with a Web Service wherein `prohibited` is specified in the `wsa:Anonymous` element or the `wsam:NonAnonymousResponses` element is specified in the child element of the `wsam:Addressing` element, create an addressing header specifying a non-anonymous URI.

37.6.3 Notes for the wsaw:Action and wsam:Action attributes

A value of the `wsa:Action` element in the addressing header of a request message sent by the JAX-WS engine on a client machine differs according to the attributes coded in the WSDL. The following values are specified in the `wsa:Action` element:

When coding the `wsaw:Action` attribute

A value of the `wsaw:Action` attribute is specified.

When coding the `wsam:Action` attribute

A value of the `wsam:Action` attribute is specified.

When coding the `wsam:Action` and `wsaw:Action` attributes

A value of the `wsam:Action` attribute is specified.

When no attribute is coded

A default `Action` value, predetermined in the specification, is specified.

For details on the default `Action` value, see the *WS-Addressing 1.0 Metadata* specifications.

37.6.4 Notes for the wsa:Action element

If the value of the `wsa:Action` element in the addressing header and the `SOAPAction` value differ in the response message received by the JAX-WS engine of the Web Service client machine, the operations might not function properly.

37.6.5 Notes related to acquiring SEI

If you use the `getPort(Class<T>, WebServiceFeature...)` method of the `W3CEndpointReference` class, `WebServiceException` occurs, and you cannot acquire SEI. To acquire SEI, use the `getPort(EndpointReference, Class<T>, WebServiceFeature...)` method of the `Service` class.

38 Examples of development from SEI (when addressing functionality used)

This chapter describes the examples for developing Web Services starting from SEI, using the addressing functionality.

38.1 Configuration examples of development (Starting from SEI and addressing)

This chapter describes examples for developing the Web Services starting from SEI.

The following table lists and describes the configuration of the Web Service to be developed:

Table 38-1: Configuration of the Web Service (Starting from SEI and addressing)

No.	Item		Value
1	Name of the J2EE server to be deployed		jaxwsserver
2	Host name and port number of the Web server		webhost:8085
3	Naming Server URL		corbaname::testserver:900
4	Context route		addressing_dynamic_generate
5	Style		document/literal/wrapped
6	Namespace URI		http://sample.com
7	Port type	Number	1
8		Local name	AddNumbersImpl
9	Operation	Number	3
10		Local name 1	add
11		Local name 2	add2
12		Local name 3	add3
13	Service	Number	1
14		Local name	AddNumbersImplService
15	Port	Number	1
16		Local name	AddNumbersImplPort
17	Web Service Implementation Class		com.sample.AddNumbersImpl
18	Methods made public in the Web Service Implementation Class	Number	3
19		Method name 1	add
20		Method name 2	add2
21		Method name 3	add3
22	Exceptions thrown in the Web Service Implementation Class methods	Number	1
23		Class name	com.sample.AddNumbersFault

The following table lists and describes the configuration of the current directory, when the Web Service is developed.

Table 38-2: Configuration of the current directory (Starting from SEI and addressing)

Directory	Explanation
c:\temp\jaxws\works\addressing	This is the current directory.
server\	Used for Web Service development.
META-INF\	Corresponds to the META-INF directory of the EAR file.

Directory	Explanation
application.xml	Created in *38.3.4 Creating application.xml*.
src\	Saves the source file (*.java) for the Web Service. Used in *38.3.1 Creating Web Services Implementation Class*and *38.3.2 Compiling Web Services Implementation Classes*.
WEB-INF\	Corresponds to the WEB-INF directory of the WAR file.
web.xml	Created in *38.3.3 Creating web.xml*.
classes\	Saves the compiled class file (*.class). Used in *38.3.2 Compiling Web Service Implementation Classes*.
addressing_dynamic_generate.ear	Created in *38.3.5 Creating EAR files*.
addressing_dynamic_generate.war	Created in *38.3.5 Creating EAR files*.
client\	Used for the development of the Web Service client.
src\	Saves the source file (*.java) of the Web Service client. Used in *38.5.1 Generating a service class*and *38.5.2 Creating Implementation Class for the Web Services client*.
classes\	Saves the compiled class file (*.class). Used in *38.5.3 Compiling Implementation Class for the Web Services client*.
usrconf.cfg	Created in *38.6.1 Creating an option definition file for Java applications*.
usrconf.properties	Created in *38.6.2 Creating a user property file for Java applications*.

Change the current directory path according to the environment to be developed.

Note that the directory and file names listed in the above table will be used in the description hereafter. The part formatted in *Bold*, in the command execution example and in the Java source, indicates the specified values and generated values that are used in this example. Substitute and read according to the environment you want to build.

Also, in the development examples described in this chapter, the Web Service and Web Service client are developed in the same environment, but you can also develop them in separate environments. For developing the Web Service and Web Service client in different environments, substitute and read the current directory path suitable to the respective environments.

38.2 Flow of development examples (Starting from SEI and addressing)

This section describes the development examples for the development and execution flow.

Developing a Web Service

1. Creating Web Services Implementation Class (*38.3.1*)
2. Executing the `javac` command and compiling the Web Services Implementation Classes (*38.3.2*)
3. Creating `web.xml` (*38.3.3*)
4. Creating `application.xml` (*38.3.4*)
5. Creating EAR files (*38.3.5*)

Deploying and starting

1. Deploying EAR files (*38.4.1*)
2. Starting Web Services (*38.4.2*)

Developing a Web Service client

1. Executing the `cjwsimport` command and generating a service class (*38.5.1*)
2. Creating Implementation Class for the Web Services client (*38.5.2*)
3. Compiling Implementation Class for the Web Services client (*38.5.3*)

Executing a Web Service

1. Creating an option definition file for Java applications (*38.6.1*)
2. Creating a user property file for Java applications (*38.6.2*)
3. Executing the Web Services client (*38.6.3*)

38.3 Examples of Web Service development (Starting from SEI and addressing)

This section describes the examples for developing Web Services starting from SEI (using the addressing functionality).

38.3.1 Creating the Web Service Implementation Class

Create a Web Service Implementation Class to code the processing of Web Services. This subsection describes how to calculate the contents of the received request message and create the Web Service Implementation Class that returns response messages.

The following is an example for creating a Web Service Implementation Class:

```
package com.sample;

import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;
import javax.xml.ws.Action;
import javax.xml.ws.FaultAction;

@WebService(name = "AddNumbers", targetNamespace = "http://sample.com/")
@SOAPBinding(style=SOAPBinding.Style.DOCUMENT, use=SOAPBinding.Use.LITERAL,
parameterStyle=SOAPBinding.ParameterStyle.WRAPPED)
interface AddNumbers {

    @Action(input = "http://sample.com/input",
            output = "http://sample.com/output")
    public int add(int number1, int number2) throws AddNumbersFault;

    public int add2(int number1, int number2) throws AddNumbersFault;

    @Action(input = "http://sample.com/input3",
            output = "http://sample.com/output3",
            fault = {@FaultAction(className = AddNumbersFault.class, value = "http://
sample.com/fault3")})
    public int add3(int number1, int number2) throws AddNumbersFault;
}
```

The created `AddNumbers.java` is saved in the `c:\temp\jaxws\works\addressing\server\src\com\sample\` directory with the UTF-8 format.

Next, create the main Web Service that implements SEI. This subsection describes how to calculate the contents of the received request message and create the Web Service Implementation Class `com.sample.AddNumbersImpl` that is returned as a response message.

The following is an example for creating the main Web Service:

```
package com.sample;

import javax.jws.WebService;
import javax.xml.ws.soap.Addressing;

@Addressing
@WebService(endpointInterface = "com.sample.AddNumbers")
public class AddNumbersImpl implements AddNumbers {
    public int add(int number1, int number2) throws AddNumbersFault {
        return impl(number1, number2);
    }

    public int add2(int number1, int number2) throws AddNumbersFault {
        return impl(number1, number2);
    }

    public int add3(int number1, int number2) throws AddNumbersFault {
        return impl(number1, number2);
    }
    int impl(int number1, int number2) throws AddNumbersFault {
        if (number1 < 0 || number2 < 0) {
            throw new AddNumbersFault("Negative numbers can't be added!",
                                      "Numbers: " + number1 + ", " + number2);
```

```
        }
        return number1 + number2;
    }
}
```

The created AddNumbersImpl.java is saved in the c:\temp\jaxws\works\addressing\server\src\com\sample\ directory with the UTF-8 format.

Also create an exception class com.sample.AddNumbersFault thrown in the com.sample.AddNumbersImpl class.

The following is an example for creating an exception class:

```
package com.sample;

public class AddNumbersFault extends Exception {

    String detail;

    public AddNumbersFault(String message, String detail) {
        super(message);
        this.detail = detail;
    }

    public String getDetail() {
        return detail;
    }
}
```

The created AddNumbersFault.java is saved in the c:\temp\jaxws\works\addressing\server\src\com\sample\ directory with the UTF-8 format.

38.3.2 Compiling Web Services Implementation Classes

Compile Web Services Implementation Classes by executing the javac command. For details on the javac command, see the *JDK documentation*.

The following example describes the execution of javac command:

```
> cd c:\temp\jaxws\works\addressing\server\
> mkdir WEB-INF\classes\
> javac -cp "%COSMINEXUS_HOME%\jaxws\lib\cjjaxws.jar;%COSMINEXUS_HOME%\CC\client\lib
\j2ee-javax.jar;%COSMINEXUS_HOME%\jaxp\lib\csmjaxb.jar;%COSMINEXUS_HOME%\jaxp\lib
\csmjaxp.jar;%COSMINEXUS_HOME%\jaxp\lib\csmstax.jar" -d WEB-INF\classes\ -s src src
\com\sample\AddNumbers.java src\com\sample\AddNumbersImpl.java src\com\sample
\AddNumbersFault. java
```

On successful execution of the javac command, the compiled classes are output to the following path:

c:\temp\jaxws\works\addressing\server\WEB-INF\classes\com\sample\directory

Executing the cjwsgen command for the compiled Web Services Implementation Classes enables you to check errors in advance. For details on the cjwsgen command see *14.3 cjwsgen command* and for details on error checking see *10.23 (1) Using the cjwsgen command to check errors*.

- In Windows (x64)

```
> set HNTRLIB2_HOME=HNTRLib2-Installation-directory
> cd c:\temp\jaxws\works\addressing\server\
> mkdir \WEB-INF\classes\
> apt -factory com.cosminexus.istack.ws.AnnotationProcessorFactoryImpl -factory
com.cosminexus.istack.ws.AnnotationProcessorFactoryImpl -J-
Dcosminexus.home="%COSMINEXUS_HOME%" -cp "%COSMINEXUS_HOME%\jaxws\lib\cjjaxws.jar;
%COSMINEXUS_HOME%\CC\client\lib\j2ee-javax.jar;%COSMINEXUS_HOME%\jaxp\lib
\csmjaxb.jar;%COSMINEXUS_HOME%\jaxp\lib\csmjaxp.jar;%COSMINEXUS_HOME%\jaxp\lib
\csmstax.jar;%HNTRLIB2_HOME%\classes\hntrlib2j64.jar;%HNTRLIB2_HOME%\classes
\hntrlibMj64.jar" -d WEB-INF\classes\ -s src src\com\sample\AddNumbers.java src\com
\sample\AddNumbersImpl.java src\com\sample\AddNumbersFault.java
```

38.3.3 Creating web.xml

Create `web.xml` that is required as a WAR file component.

The following is an example for creating `web.xml`:

```
<<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/
javaee/web-app_3_0.xsd">
  <description>Sample web service &quot;addressing&quot;</description>
&quot;addressing_dynamic_generate&quot;</description>
<display-name>Sample_web_service_addressing_dynamic_generate</display-name>
  <listener>
    <listener-class>


com.cosminexus.xml.ws.transport.http.servlet.WSServletContextListener
    </listener-class>
  </listener>
  <servlet>
    <description>Endpoint servlet for Cosminexus JAX-WS</description>
    <display-name>Endpoint_servlet_for_Cosminexus_JAX_WS</display-name>
    <servlet-name>CosminexusJaxwsServlet</servlet-name>
    <servlet-class>
      com.cosminexus.xml.ws.transport.http.servlet.WSServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>CosminexusJaxwsServlet</servlet-name>
    <url-pattern>/AddNumbersImplService</url-pattern>
  </servlet-mapping>
  <session-config>
    <session-timeout>60</session-timeout>
  </session-config>
</web-app>
```

When creating `web.xml` of version 2.5, specify `2.5` in the `version` attribute of the `web-app` element and specify `http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd` as the second location information in the `xsd:schemaLocation` attribute.

The created `web.xml` is saved in the `c:\temp\jaxws\works\addressing\server\WEB-INF\` directory with the UTF-8 format. For details about the `web.xml` settings, see the section *3.4 Creating web.xml*.

38.3.4 Creating application.xml

Create `application.xml` that is required as an EAR file component.

The following is an example for creating `application.xml`. Note that no items are set up in `application.xml` as the Web Service.

```
<?xml version="1.0" encoding="UTF-8"?>
<application version="6" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/
javaee/application_6.xsd">

<description>Sample application
 &quot;addressing_dynamic_generate &quot;</description>
<display-name>Sample_application_addressing_dynamic_generate</display-name>
 <module>
  <web>
  <web-uri>addressing_dynamic_generate.war</web-uri>
  <context-root>addressing_dynamic_generate</context-root>
  </web>
  </module>
  </application>
```

When creating `web.xml` of version 5, specify `5` in the `version` attribute of the `application` element and specify `http://java.sun.com/xml/ns/javaee/application_5.xsd` as the second location information in the `xsd:schemaLocation` attribute.

The created `application.xml` is saved in the `c:\temp\jaxws\works\addressing\server\META-INF\` directory with the UTF-8 format. For details about the precautions for creating `application.xml`, see *5.2.2 Notes on editing application.xml* in the *uCosminexus Application Server Application Development Guide*.

38.3.5 Creating EAR files

Use the `jar` command to create an EAR file containing the created files.

The following is an example for creating an EAR file:

```
> cd c:\temp\jaxws\works\addressing\server\
> jar cvf addressing_dynamic_generate.war .\WEB-INF
> jar cvf
addressing_dynamic_generate.ear .\addressing_dynamic_generate.war .\META-INF
\application.xml
```

If the `jar` command is terminated successfully, `addressing_dynamic_generate.ear` is created in the `c:\temp\jaxws\works\addressing\server\` directory.

For the `jar` command, reference the *JDK documentation*.

38.4 Examples of deployment and startup (Starting from SEI and addressing)

This section describes the examples for the deployment and the startup starting from SEI, using the addressing functionality.

38.4.1 Deploying EAR files

Use the `cjimportapp` command to deploy the created EAR file on the J2EE server.

The following is an example of deployment:

```
> cd c:\temp\jaxws\works\addressing\server\
> "%COSMINEXUS_HOME%\CC\admin\bin\cjimportapp" jaxwsserver -nameserver
corbaname::testserver:900 -f addressing_dynamic_generate.ear
```

For the `cjimportapp` command, see *cjimportapp (Importing J2EE applications)* in the *uCosminexus Application Server Command Reference Guide*.

For the method to deploy (import) J2EE applications by using the management portal, see *12.3.3 Importing J2EE applications* in the *uCosminexus Applications Server Management Portal User Guide.*

38.4.2 Starting Web Services

Use the `cjstartapp` command to start Web Services.

The following is an example for starting a Web Service:

```
> cd c:\temp\jaxws\works\addressing\ \
> "%COSMINEXUS_HOME%\CC\admin\bin\cjstartapp" jaxwsserver corbaname::testserver:900 -
name Sample_application_addressing_dynamic_generate
```

For the `cjstartapp` command, see *cjstartapp (Starting J2EE applications*) in the *uCosminexus Application Server Command Reference Guide*.

For the method to start J2EE applications by using the management portal, see *12.3.1 Starting J2EE applications* in the *uCosminexus Application Server Management Portal User Guide.*

38.5 Examples of Web Service client development (Starting from SEI and addressing)

This section describes the examples for the development of Web Service clients (using the addressing functionality) starting from SEI.

38.5.1 Generating a service class

If you execute the `cjwsimport` command, the Java source such as a service class that is required for developing a Web Service client is generated. For details about the `cjwsimport` command, see the section *14.1 cjwsimport command*.

The following is an example of the execution of the `cjwsimport` command:

```
> cd c:\temp\jaxws\works\addressing\client\
> mkdir src\
> mkdir classes\
> "%COSMINEXUS_HOME%\jaxws\bin\cjwsimport.bat" -s src -d classes http://webhost:8085/
addressing_dynamic_generate/AddNumbersImplService?wsdl
```

If the `cjwsimport` command is terminated successfully, the Java source is generated in the `c:\temp\jaxws\works\addressing\client\src\com\sample\` directory.

The following table lists the products:

Table 38-3: Products during service class generation (Starting from SEI and addressing)

File name	Explanation
`Add.java`	This is the `JavaBean` class corresponding to the type referenced by the `wrapper` element of the request message of `operation` in the WSDL definition.
`AddResponse.java`	This is the `JavaBean` class corresponding to the type referenced by the `wrapper` element of the response message of `operation` in the WSDL definition.
`Add2.java`	This is the `JavaBean` class corresponding to the type referenced by the `wrapper` element of the request message of `operation` in the WSDL definition.
`Add2Response.java`	This is the `JavaBean` class corresponding to the type referenced by the `wrapper` element of the response message of `operation` in the WSDL definition.
`Add3.java`	This is the `JavaBean` class corresponding to the type referenced by the `wrapper` element of the request message of `operation` in the WSDL definition.
`Add3Response.java`	This is the `JavaBean` class corresponding to the type referenced by the `wrapper` element of the response message of `operation` in the WSDL definition.
`ObjectFactory.java`	This is the `ObjectFactory` class of the JAXB 2.2 specifications.
`package-info.java`	This is the `package-info.java` file.
`AddNumbers.java`	This is the Service Endpoint Interface (SEI) corresponding to `service` in the WSDL definition.
`AddNumbersImplService.java`	This is a service class.
`AddNumbersFault.java`	This is the `JavaBean` class corresponding to `AddNumbersFault` in the WSDL definition.

File name	Explanation
AddNumbersFault_Exception.java	This is the wrapper exception class of the fault bean.

The file names Add, AddNumbersImpl, and AddNumbersImplService change as per the coding of the local names in operation, port type, and service. For details about mapping of the local names in operation, port type, and service, see the following sections:

- *15.1.2 Mapping a port type to a SEI name*
- *15.1.3 Mapping an operation to a method name*
- *15.1.4 Mapping a message part to a parameter and return value (For wrapper style)*
- *15.1.5 Mapping a message part to a parameter and return value (For non-wrapper style)*

38.5.2 Creating an implementation class for the Web Service client

Create an implementation class for the Web Service client that uses the Web Services.

The following is an example for creating a Web Service client that invokes the Web Service thrice:

```
package com.sample.client;

import javax.xml.namespace.QName;
import javax.xml.ws.soap.AddressingFeature;
import javax.xml.ws.wsaddressing.W3CEndpointReference;
import javax.xml.ws.wsaddressing.W3CEndpointReferenceBuilder;

import com.sample.AddNumbers;
import com.sample.AddNumbersImplService;
import com.sample.AddNumbersFault_Exception;

public class TestClient {
    int number1 = 10;
    int number2 = 10;
    int negativeNumber = -10;

    public static void main(String[] args) {
        TestClient client = new TestClient();

        client.existActionAnnotation1();
        client.existActionAnnotation2();
        client.notExistActionAnnotation();
        client.existFaultActionAnnotation();
        client.notExistFaultActionAnnotation();
    }

    public void existActionAnnotation1() {
        System.out.println("existActionAnnotation1");
        try {
            AddressingFeature feature = new AddressingFeature();
            AddNumbersImplService service = new AddNumbersImplService();
            AddNumbers stub = service.getAddNumbersImplPort(feature);
            int result = stub.add(number1, number2);
            assert result == 20;
        } catch (Exception ex) {
            ex.printStackTrace();
            assert false;
        }
    }

    public void existActionAnnotation2() {
        System.out.println("existActionAnnotation2");
        try {
            AddressingFeature feature = new AddressingFeature();
            W3CEndpointReferenceBuilder eprBuilder = new
W3CEndpointReferenceBuilder();
            eprBuilder.address("http://webhost:8085/addressing_dynamic_generate/
AddNumbersImplService");
            eprBuilder.serviceName(new QName("http://sample.com/",
"AddNumbersImplService"));
            eprBuilder.endpointName(new QName("http://sample.com/",
"AddNumbersImplPort"));
            W3CEndpointReference epr = eprBuilder.build();
```

```
            AddNumbersImplService service = new AddNumbersImplService();
            AddNumbers stub = service.getPort(epr, AddNumbers.class, feature);
            int result = stub.add(number1, number2);
            assert result == 20;
        } catch (Exception ex) {
            ex.printStackTrace();
            assert false;
        }
    }

    public void notExistActionAnnotation() {
        System.out.println("notExistActionAnnotation");
        try {
            AddressingFeature feature = new AddressingFeature();
            AddNumbersImplService service = new AddNumbersImplService();
            AddNumbers stub = service.getAddNumbersImplPort(feature);
            int result = stub.add2(number1, number2);
            assert result == 20;
        } catch (Exception ex) {
            ex.printStackTrace();
            assert false;
        }
    }

    public void existFaultActionAnnotation() {
        System.out.println("existFaultActionAnnotation");
        try {
            AddressingFeature feature = new AddressingFeature();

            AddNumbersImplService service = new AddNumbersImplService();
            AddNumbers stub = service.getAddNumbersImplPort(feature);
            stub.add3(negativeNumber, number2);
            assert false;
        } catch (AddNumbersFault_Exception e) {
            System.out.println("This is expected exception");
        } catch (Exception e) {
            e.printStackTrace();
            assert false;
        }
    }
    public void notExistFaultActionAnnotation() {
        System.out.println("notExistFaultActionAnnotation");
        try {
            AddressingFeature feature = new AddressingFeature();

            AddNumbersImplService service = new AddNumbersImplService();
            AddNumbers stub = service.getAddNumbersImplPort(feature);
            stub.add(negativeNumber, number2);
            assert false;
        } catch (AddNumbersFault_Exception ex) {
            System.out.println("This is expected exception");
        } catch (Exception e) {
            e.printStackTrace();
            assert false;
        }
    }
}
```

The created `TestClient.java` is saved in the `c:\temp\jaxws\works\addressing\client\src\com\sample\client\` directory with the UTF-8 format.

Note that `com.sample`, `AddNumbers`, `AddNumbersImplService`, `AddNumbersImplPort`, `add`, `add2`, and `add3` differ as per the package names, class names, and the method names in the classes of the generated Java source. For developing Web Services with different configurations, you must review the coding of the package names, class names, and method names in the classes.

38.5.3 Compiling the implementation class for the Web Service client

Use the `javac` command to compile the created Web Service client.

The following is an example of compilation:

```
> cd c:\temp\jaxws\works\addressing\client\
> javac -encoding UTF-8 -cp "%COSMINEXUS_HOME%\jaxws\lib\cjjaxws.jar;%COSMINEXUS_HOME%
```

```
\CC\client\lib\j2ee-javax.jar;%COSMINEXUS_HOME%\jaxp\lib\csmjaxb.jar;.\classes" -d .
\classes src\com\sample\client\TestClient.java
```

If the `javac` command is terminated successfully, `TestClient.class` is generated in the `c:\temp\jaxws\works\addressing\client\classes\com\sample\client\` directory.

For the `javac` command, reference the JDK documentation.

38.6 Examples of Web Service execution (Starting from SEI and addressing)

This section describes the examples for executing Web Service clients (when using the addressing functionality) starting from SEI.

38.6.1 Creating option definition files for Java applications

Create option definition files for Java applications (`usrconf.cfg`) required for executing a Web Service.

The following is an example for creating option definition files for Java applications:

```
add.class.path= Cosminexus -installation-directory\jaxws\lib\cjjaxws.jar
add.class.path=.\classes
ejb.client.log.directory=logs
add.jvm.arg=-Dcosminexus.home= Cosminexus -installation-directory
add.jvm.arg=-Dejbserver.server.prf.PRFID=<PRF ID>
```

For *Cosminexus-installation-directory*, use the absolute path to specify the path where Cosminexus is installed.

The created option definition file for Java applications is saved in the `c:\temp\jaxws\works\addressing\client\` directory. For details about the option definition files for Java applications, see *14.2 usrconf.cfg (Option definition file for Java applications)* in the *uCosminexus Application Server Definition Reference Guide*.

38.6.2 Creating user property files for Java applications

Create user property files for Java applications required for executing Web Services.

Because the settings are not specially changed, create an empty file named `usrconf.properties` in the `c:\temp\jaxws\works\addressing\client\` directory. For details about the user property files for Java applications, see *14.3 usrconf.properties (User property file for Java applications)* in the *uCosminexus Application Server Definition Reference Guide*.

38.6.3 Executing the Web Service client

Use the `cjclstartap` command to execute Web Service clients.

The following is an example for executing a Web Service client:

```
> cd c:\temp\jaxws\works\addressing\client\
> "%COSMINEXUS_HOME%\CC\client\bin\cjclstartap" com.sample.client.TestClient
```

If the `cjclstartap` command is terminated successfully, the result of Web Service client execution is displayed. The following is an example for displaying the execution results:

```
KDJE40053-I The cjclstartap command will now start. (directory for the user
definition file = c:\temp\jaxws\works\addressing\client, PID = 2636)
existActionAnnotation1
existActionAnnotation2
notExistActionAnnotation
existFaultActionAnnotation
This is expected exception
notExistFaultActionAnnotation
This is expected exception
KDJE40054-I The cjclstartap command was stopped. (PID = 2636, exit status = 0)
```

The part in italics changes according to the execution timing and the environment.

For the `cjclstartap` command, see *cjclstartap (Starting Java applications)* in the *uCosminexus Application Server Command Reference Guide*.

39 Troubleshooting

This chapter describes the action for failures that occurred during the operations of Web Services or Web resources.

39.1 Types of failure and actions

The following failures might occur when operating Web Services or Web resources:

When a Web Service or a Web resource does not operate normally

- Running program ends abnormally
- Program does not operate as intended

When a Web Service or a Web resource operates normally

- The performance is not as expected

If the program does not operate normally, an error message or warning message is displayed on the log or console and the program ends abnormally. Even if the program ends normally, the operations might not be as intended.

Furthermore, even if the program operates normally, the processing capacity of the program or component might not reach the expected performance.

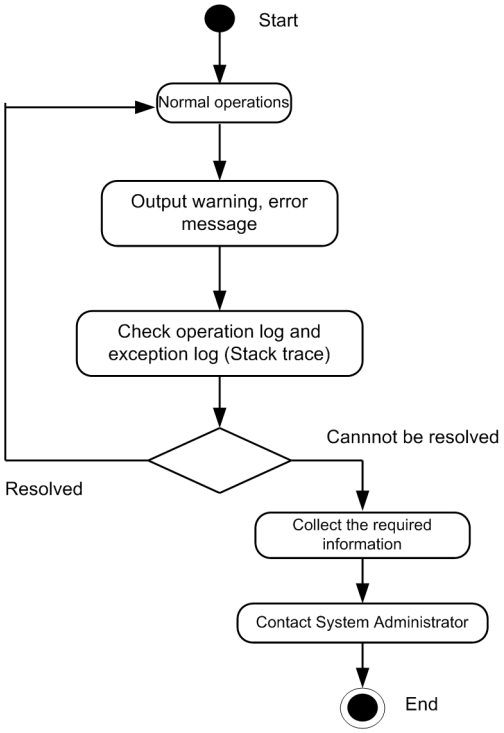
The following subsections describe each of the above cases. Also, when invoking an EJB Web Service, an error might occur in an EJB container. For details on the errors that occur in EJB containers, see *8.11 Trace collection points of EJB containers* in the *uCosminexus Application Server Maintenance and Migration Guide*.

Note that you cannot define a Web Service or a Web resource message in a System Log Messages Mapping file for JP1/IM integration. Consequently, you cannot convert a Web Service or a Web resource message to a JP1 event and issue that JP1 event. For details on the System Log Messages Mapping file for JP1/IM integration, see *10.19 System Log Messages Mapping files for JPI/IM integration* in the *uCosminexus Application Server Definition Reference Guide.*

39.1.1 When a running program ends abnormally

The following figure shows the flow of actions when a running program ends abnormally:

Figure 39－1: Flow of checks when a warning, error message is output



When a created program does not operate normally and issues an error message or warning message, check the cause from the contents of the message output in the log. The log that you will check is the operation log and the exception

log. Check the cause of the failure on the basis of the information output in each log and the stack trace information of the exception.

Note that the warning and error messages, that occur during the execution of a command, are output in the command operation or exception logs. For details about the logs, see *39.3 Log*.

If the occurred failure is considered to have a cause apart from the program and if the cause cannot be resolved, collect the failure information and contact the system administrator. For details about the failure information, see *39.2 Material to be acquired when a failure occurs*.

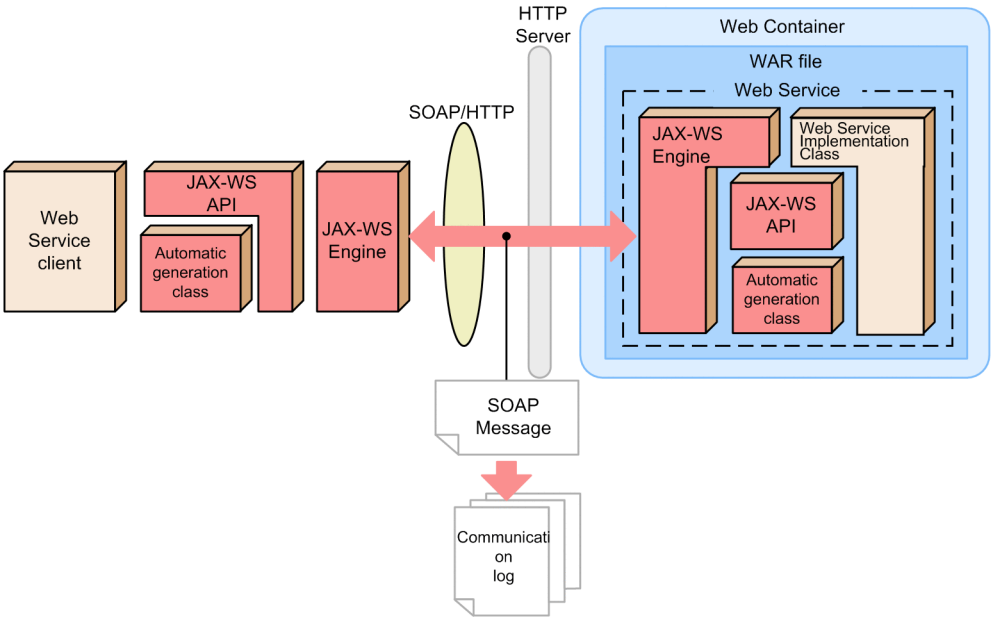
39.1.2 When a program does not operate as intended

Even if error or warning messages are not issued, a created program does not operate as intended due to causes such as defects. In this case, the program might not send an intended SOAP Message (in the case of Web Services) or an HTTP message (in the case of Web resources) for implementing Web Services or Web resources. If the program does not operate as intended, analyze the contents of the sent and received SOAP Messages and check whether the intended message is being sent.

For details about the contents output to the communication logs, see *39.3.5 Log format*.

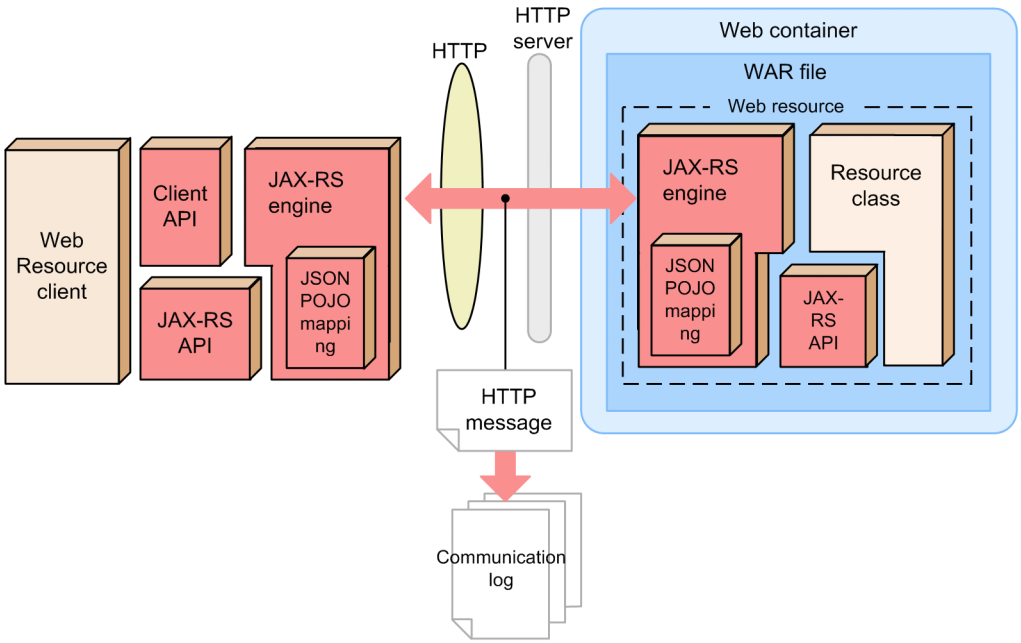
The following figure shows the flow of communication for SOAP Messages.

Figure 39–2: Output of communication log (for Web Services)



The following figure shows an HTTP message communication flow.

Figure 39－3: Output of the communication log (In the case of Web resources)



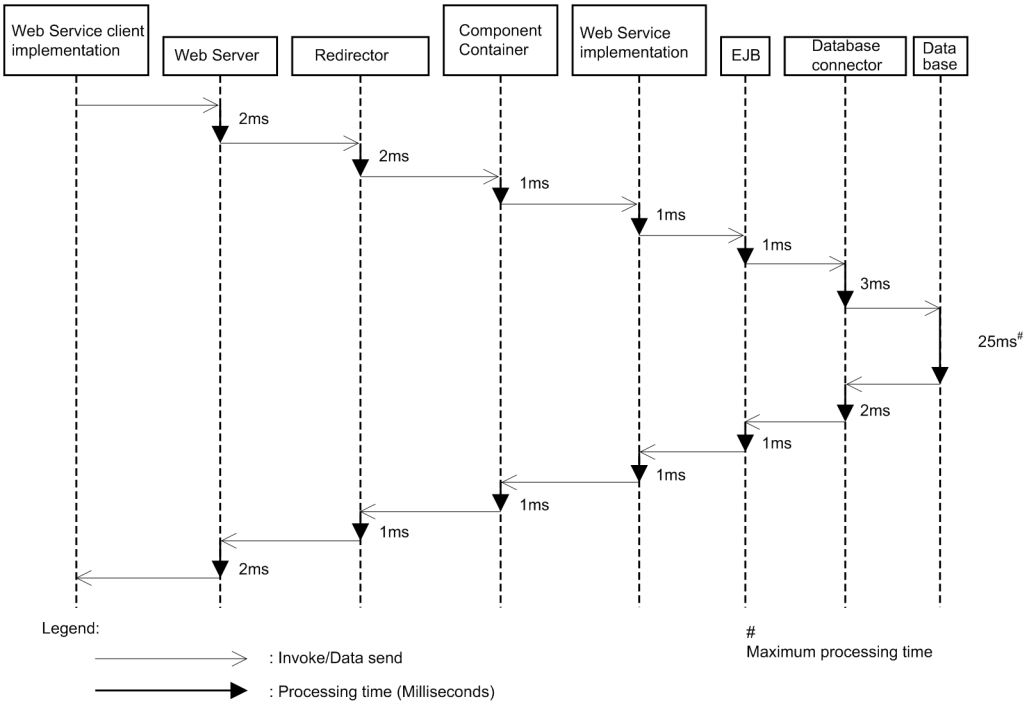
39.1.3 When the performance is not as expected

Even if the program is operating normally, the runtime performance might not be as expected. Apart from problems in the program, the cause might be the problems in components such as the Web server and database. If there are problems in the components, the check using the Trace based performance analysis (PRF) is more effective. If you use the Trace based performance analysis, you can analyze the processing time of each layer and the status of requests. Follow the flow of requests from the program in the Web Service client and check where the delay is occurring.

For details about the checking method using the Trace based performance analysis, see *39.4.3 Method of performance analysis based on Trace based performance analysis*.

The following figure shows an example of the processing time in each functionality layer.

Figure 39-4: Example of the processing time in each functionality layer



39.2 Material to be acquired when a failure occurs

For a failure for which you need to contact the system administrator or a failure for which you do not understand the measures you need to take, collect the following failure information and contact the system administrator:

Note that '?' in the following file names indicates the number of files:

- Operation log (`cjwmessage?.log` or `cjrmessage?.log`)
- Exception log (`cjwexception?_?.log` or `cjrexception?_?.log`)
- Maintenance log (`cjwmessage?.log` or `cjrmessage?.log`)
- Communication log (`cjwtransport?_?.log` or `cjrtransport?_?.log`)
- Command operation log (`cjwsimport?.log`, `cjwapt?.log`)
- Command exception log (`cjwsimportex?_?.log`, `cjwaptex?_?.log`)
- Command maintenance log (`cjwsimport?.log`, `cjwapt?.log`)
- Trace based performance analysis
- User-program specific log (only when present)
- Common definition file (`cjwconf.properties` or `cjrconf.properties`)
- Process-wise definition file
- DD file for the Web Service (`web.xml`, cosminexus `-jaxws.xml`)
- System properties and system class path set for the server and client
- Standard output and standard error output of the server and client
- Cosminexus Component Container and the Web server log
- Information collected for failure, defined in Cosminexus Component Container:
  - User-defined file for the J2EE server
  - Maintenance information of the J2EE server
  - User-defined file of the server management commands
  - Maintenance information of the server management commands
  - Standard output and standard error output of the J2EE server
  - Cosminexus TPBroker log in the J2EE server
  - System properties and the system class path set in the EJB client application
  - Standard output and standard error output of the EJB client application

To obtain the collection information as specified in Cosminexus Component Container when a failure occurs, see *3.8.9 How to collect data* in the *uCosminexus Application Server Maintenance and Migration Guide*.

39.3 Log

This section describes the logs output during the development and execution of Web Services or Web resources.

39.3.1 Types of log

The following table lists and describes the output log types and provides an overview.

Table 39-1: Types of log

No.	Classification	Overview
1	operation log	This log outputs the operation status. This log is used to check the problem that occurred and to remove the problem.
2	exception log	This log outputs the exception stack trace. This log is used to check the details of the exception that occurred.
3	maintenance log	This log outputs the maintenance information.
4	communication log#	This log outputs the sent and received SOAP Messages. This log is used to check the contents sent and received during application development and during failure investigation.

#

In the case of a JAX-WS functionality, for enabling the communication log, specify a value in the `com.cosminexus.jaxws.logger.runtime.transport.client_dump` property (communication log output (in the Web Service client)), and `com.cosminexus.jaxws.logger.runtime.transport.server_dump` property (communication log output (in the Web Service)) of the action definition file.

For details about the properties, see *10.1.2 Settings for the common definition file*.

In the case of a JAX-RS functionality, for enabling the communication log, specify a value in the `com.cosminexus.jaxrs.logger.runtime.transport.server.level` property (communication log output level) of the action definition file. For details on the properties, see *13.1.2 Common definition file settings*.

The following table lists and describes the relationship between the functionality that outputs the log and the output log.

Table 39-2: Relationship between the functionality and the output log

No.	Functionality	Output log			
		Operation log	Exception log	Maintenance log	Communication log
1	Web Service and Web resource	Y	Y	Y	Y
2	Web Service client	Y	Y	Y	Y
3	`cjwsimport` command	Y	Y	Y	--
4	`apt` command	Y	Y	Y	--
5	`cjwsgen` command	Y	Y	Y	--

Legend:

Y: Log is output.

--: Log is not output.

39.3.2 Log file rotation

With the Cosminexus JAX-WS functionality and JAX-RS functionality, logs are rotated to multiple files, and then the logs are output. This subsection describes the log file rotation.

(1) How to rotate

With the Cosminexus JAX-WS functionality and JAX-RS functionality, if Web resources or Web Services run on a J2EE server, you can specify the rotation switching for types of the output logs. The following table lists the log types and describes whether the settings to switch the log types can be specified.

Table 39-3: Log types and whether the rotation switching can be specified

No.	Log types	Whether the rotation switching can be specified
1	Operation log	Y
2	Exception log	N
3	Maintenance log	Y
4	Communication log	N

Legends:

Y: You can specify the settings for rotation switching.

N: You cannot specify the settings for rotation switching.

Note

If the output destination file has the same size as that of the specified file, the output destination is switched to the next file (`wraparound` mode), by default. For details, see *3. Preparation for Troubleshooting* in the *uCosminexus Application Server Maintenance and Migration Guide*.

You cannot specify a rotation switching in the logs of the Web Service client that starts by the Java application startup command (`cjclstartap` command) or in the logs output when implementing the command line interface.

39.3.3 Log output destination

This subsection describes the log output destination for each usage situation.

(1) When using the Web Service on the J2EE server and the Web Service client

The following table lists and describes the log output destination of the Web Service and Web Service client (such as servlet and EJB) running on the J2EE server:

Table 39-4: Log output destination (for a Web Service or Web resource client running on the J2EE)

No.	Log types	Output destination directory	File names
1	Operation log	`<ejb.server.log.directory>/CJW`	For the round up method `cjwmessage?.log` For the shift method Current log file: `cjwmessage.log` Back up file: `cjwmessage?.log`
2	Exception log	`<ejb.server.log.directory>/CJW`	`cjwexception?_?.log`

3	Maintenance log	`<ejb.server.log.directory>/CJW/maintenance`	For the round up method `cjwmessage?.log`
			For the shift method
			Current Log File:
			`cjwmessage.log`
			Back up file:
			`cjwmessage?.log`

4	Communication log	`<ejb.server.log.directory>/CJW`	`cjwtransport?_?.log`

Notes

- <ejb.server.log.directory> becomes the log output destination for the J2EE server. For details, see *3.3.6 Settings for collecting the J2EE server logs* in the *uCosminexus Application Server Maintenance and Migration Guide*.
- '?' in the file name indicates the number of files.
- If a problem occurs before the log initialization processing, the log is output to the operation log (cjmessage?.log) of Cosminexus Component Container.
- A *current log file* is the log file of the output destination.
- A *back up file* is the log file that is set up as a back up by rotation.

(2) When using the Web Service client in the command line interface

The following table lists and describes the log output destination when the Web Service client is invoked using the Java application start command (cjclstartap):

Table 39-5: Log output destination (when using a Web Service from the command line interface)

No.	Log type	Output destination directory	File name
1	Operation log	<ejbserver.client.log.directory>/ <ejbserver.client.ejb.log>/ <ejbserver.client.log.appid>/CJW	cjwmessage?.log
2	Exception log	<ejbserver.client.log.directory>/ <ejbserver.client.ejb.log>/ <ejbserver.client.log.appid>/CJW	cjwexception?_?.log
3	Maintenance log	<ejbserver.client.log.directory>/ <ejbserver.client.ejb.log>/ <ejbserver.client.log.appid>/CJW/maintenance	cjwmessage?.log
4	Communication log	<ejbserver.client.log.directory>/ <ejbserver.client.ejb.log>/ <ejbserver.client.log.appid>/CJW	cjwtransport?_?.log

Notes

- <ejbserver.client.log.directory>, <ejbserver.client.ejb.log>, and <ejbserver.client.log.appid> become the output destination for the system log of the EJB client application. For details, see *4.5.2 Output destination for system logs of EJB client applications* in the manual *uCosminexus Application Server Maintenance and Migration Guide*.
- '?' in the file name indicates the number of files.
- If a problem occurs before the log initialization processing, the log is output to the operation log (cjclmessage?.log) of Cosminexus Component Container.

(3) When a command is executed

The following table lists and describes the log output destination when a command is executed.

Table 39-6: Log output destination (when a command is executed)

No.	Log types	Output destination directory	File names
1	Operation log	*Command-log-output-destination-directory* (com.cosminexus.jaxws.tool.log.directory)	For the cjwsimport command: cjwsimport?.log For the apt command: cjwapt?.log For the cjwsgen command: cjwsgen?.log
2	Exception log	*Command-log-output-destination-directory* (com.cosminexus.jaxws.tool.log.directory)	For the cjwsimport command: cjwsimportex?_?.log For the apt command: cjwaptex?_?.log For the cjwsgen command: cjwsgenex?_?.log
3	Maintenance log	*Command-log-output-destination-directory* (com.cosminexus.jaxws.tool.log.directory)/maintenance	For the cjwsimport command: cjwsimport?.log For the apt command: cjwapt?.log For the cjwsgen command: cjwsgen?.log

Notes

- Set the *Command-log-output-destination-directory* (com.cosminexus.jaxws.tool.log.directory) using the properties of the action definition file. The default output destination is *Cosminexus-installation-directory*/jaxws/logs.

  For details about the properties, see *10.1.2 Settings for the common definition file*.
- '?' in the file name indicates the number of files.

(4) When using a Web resource and a Web resource client operating on the J2EE server

The following table describes the output destinations for logs when using a Web resource and a Web resource client operating on the J2EE server.

Table 39-7: Log output destination (for a Web Service and a Web resource client running on the J2EE server)

No.	Log type	Output destination directory	File name
1	Operation log	<*ejb.server.log.directory*>/CJR	For a Round up method cjrmessage?.log For a Shift method Current Log files: cjrmessage.log Back up files: cjrmessage?.log
2	Exception log	<*ejb.server.log.directory*>/CJR	cjrexception?_?.log
3	Maintenance log	<*ejb.server.log.directory*>/CJR/maintenance	For a Round up method cjrmessage?.log For a Shift method Current Log files: cjrmessage.log Back up files: cjrmessage?.log
4	Communication log	<*ejb.server.log.directory*>/CJR	cjrtransport?_?.log

Notes

- The <*ejb.server.log.directory*> serves as the log output destination of the J2EE server. For details, see *3.3.6 Settings for collecting the J2EE server logs* in the *uCosminexus Application Server Maintenance and Migration Guide*.
- '?' in the file name indicates the number of files.
- If a problem occurs before the log initialization processing, the log is output to the operation log (`cjrmessage?.log`) of Cosminexus Component Container.
- A current log file is the log files of the output destination.
- A backup file is a log file that is backed up by rotation.

(5) When using Web resources from the command line interface

The following table lists log output destinations when starting a Web resource client by using a startup command (`cjclstartap`) of Java applications.

Table 39-8: Log output destination (when using a Web resource from the command line interface)

No.	Log type	Output destination directory	File name
1	Operation log	<*ejbserver.client.log.directory*>/ <*ejbserver.client.ejb.log*>/ <*ejbserver.client.log.appid*>/CJR	`cjrmessage?.log`
2	Exception log	<*ejbserver.client.log.directory*>/ <*ejbserver.client.ejb.log*>/ <*ejbserver.client.log.appid*>/CJR	`cjrexception?_?.log`
3	Maintenance log	<*ejbserver.client.log.directory*>/ <*ejbserver.client.ejb.log*>/ <*ejbserver.client.log.appid*>/CJR/maintenance	`cjrmessage?.log`
4	Communication log	<*ejbserver.client.log.directory*>/ <*ejbserver.client.ejb.log*>/ <*ejbserver.client.log.appid*>/CJR	`cjrtransport?_?.log`

Notes

- <*ejbserver.client.log.directory*>, <*ejbserver.client.ejb.log*>, and <*ejbserver.client.log.appid*> serve as the system log output destination of an EJB client application. For details, see *4.5.2 Output destination of system logs for EJB client applications* in the *uCosminexus Application Server Maintenance and Migration Guide.*
- '?' in the file name indicates the number of files.
- If a problem occurs before the log initialization processing, the log is output to the operation log (`cjrmessage?.log`) of Cosminexus Component Container.

39.3.4 Importance level and output conditions of logs

This subsection describes the importance level and output conditions of log.

(1) Importance level of log

The following table lists and describes the importance level of the log and the output contents of the message.

Table 39-9: Meaning of importance level and the output contents

No.	Importance level	Meaning of the message	Output contents of the message
1	ERROR	This message level indicates an important failure. This message is output when a problem occurs and the processing cannot continue. To continue processing, action needs to be taken.	From the message ID indicators, a message with -E is output. The failure information such as the occurred exception and the action is output in the message.
2	WARN	This message level indicates a potential problem. This message is output when a problem occurs, but the processing can continue. This potential problem does not need immediate action, but it is desirable that action be taken.	From the message ID indicators, a message with -W is output. The failure information such as the occurred exception and the action contents are output in the message. (Example) The definition is wrong, so the default value is used.
3	INFO	This message level provides information. Though action is not required, the information that must be notified is output.	A message with message ID indicator -I is output. The contents to be notified are output in the message. (Example) The command processing has started.
4	DEBUG	This is a DEBUG level message. This information is used for investigating the cause of failure.	A message with the message ID KDJW99999-I (for Web Services) or KDJJ99999-I (for Web resources) is output. (Example) Such as the runtime environment, method trace.

For details about the importance levels of the logs output when the log levels are defined in the configuration file, see *39.3.6 Setting logs*.

(2) Operation log/ Exception log/ Maintenance log output conditions

The contents output in the operation log, exception log, and maintenance log change depending on the log definition specified in the action definition file.

The following table lists and describes the output conditions for the operation log, exception log, and maintenance log.

Table 39-10: Output conditions for the operation log/ exception log/ maintenance log (Importance level of the event)

Log types	Log definition	Importance level of the occurred event			
		ERROR	WARN	INFO	DEBUG
Operation log Exception log #	NONE	--	--	--	--
	ERROR	Y	--	--	--
	WARN	Y	Y	--	--
	INFO	Y	Y	Y	--
	DEBUG	Y	Y	Y	Y
Maintenance log	NONE	--	--	--	--
	ALL	Y	Y	Y	Y

Legend:

Y: Log is output.

--: Log is not output.

#

The exception log is not output if there is no exception information. Note that the exception information specifies the stack trace of the exception that causes the event to occur.

The default log definition is `INFO` for the output level of the operation log and exception log and `ALL` for the output level of the maintenance log. Alternatively, if an event with importance level 'ERROR' occurs in the default settings, log is output in the operation log, maintenance log, and the exception log.

The number of operation and exception logs might increase whenever you change the log definition to `NONE`, `ERROR`, `WARN`, `INFO`, or `DEBUG`. Changing the log definition in such a way so that more logs are output is called *raising the output level*. If the number of logs increases, the overall processing speed might decrease, so when you raise the output level, consider the estimation of the processing speed and the number of files.

(3) Communication log output conditions

The contents output for the communication log change depending on the log definition specified in the action definition file. However, the properties to be specified differ depending on Web Service clients, Web Services, or Web resources. The properties to be set are as follows:

- `com.cosminexus.jaxws.logger.runtime.transport.client_dump` (on the Web Service client machine)
- `com.cosminexus.jaxws.logger.runtime.transport.server_dump` (on the Web Service machine)
- `com.cosminexus.jaxrs.logger.runtime.transport.server.level` (on the Web resource machine)

The following table lists and describes the definition of the communication log and the output contents:

Table 39-11: Communication log output conditions (in the Web Service client)

Log definition	Output contents
`ALL`	The sent and received messages are always output.
`HEADER`	The HTTP header of the received message is always output.
`ERROR_HEADER`	Output only when the HTTP header of the received message returns an error.
`NONE`	The sent and received messages are not output.

Table 39-12: Communication log output conditions (On a Web Service or Web resource machine)

Log definition	Output contents
`ALL`	The sent and received messages of Web Services or Web resources are always output. For a received message, the HTTP request information is also output.
`HEADER`	The HTTP header of the received message and the HTTP request information is always output.
`ERROR_HEADER`	The HTTP header of the received message and the HTTP request information are output only when an error occurs (you can specify for Web Services).
`NONE`	The sent and received messages of Web Services or Web resources are not output.

The number of communication logs increases in the sequence of the log definitions `NONE`, `ERROR_HEADER`, `HEADER`, and `ALL`. If the number of log increases, the overall processing speed might decrease, so when you change the default log definition, consider the estimation of the processing speed and the number of files.

39.3.5 Log format

This subsection describes the format of the operation log, maintenance log, exception log, and communication log.

(1) Operation log and maintenance log format

The following table lists and describes the output items and contents of the operation log and maintenance log.

Table 39-13: Operation log and maintenance log format

Items	Output contents
Number	This is the serial number of a trace code (4 digits). Starts from 0000 and returns to 0000 when the number goes to 9999.
Date	Date of output (yyyy/mm/dd format)
Time	Time of output (hh:mm:ss.nnn format)
Application name	• For Web Services and Web Service clients: `cjw` • For the `cjwsimport` command: `cjwsimport` • For the `apt` command: `cjwapt` • For the `cjwsgen` command: `cjwsgen` • For the WS-RM 1.2functionality: `wsrm` • For Web resources: `cjr`
Process identifier	Process identifier (hexadecimal)
Thread identifier	Thread identifier (hexadecimal)
Message ID	Message ID
Message types	Message types • `ER`: Indicates that an error message was displayed. • `EC`: Indicates that an exception was caught. • None: Indicates trace information other than above.
Message text	Main message body
CRLF	End of record sign

(2) Exception log and communication log format

The following table lists and describes the output items and contents of the exception log and communication log.

Table 39-14: Exception log and communication log format

Items	Output contents
Date	Date of output (yyyy/mm/dd format)
Time	Time of output (hh:mm:ss format)
Source class name	Class name that issued the log
Level	Log importance level
Message	Main message body

The following example describes a communication log output for the JAX-WS functionality:

```
2008/10/14 13:09:44 com.cosminexus.xml.ws.transport.http.client.HttpTransportPipe
process
```

```
Information: KDJW30011-I http client message
---[HTTP request]---
SOAPAction: ""
Content-Type: text/xml;charset="utf-8"
X-hitachi-rootAp: MTgxNDczMTYyLzE2ODgvMC84MDI=
X-hitachi-clientAp: MTgxNDczMTYyLzE2ODgvMC84MDI=
Accept: text/xml, multipart/related, text/html, image/gif, image/jpeg, *; q=.2, */*;
q=.2
<?xml version="1.0" ?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/
envelope/"><S:Body><ns2:jaxWsTest1 xmlns:ns2="http://example.com/
sample"><information>Invocation test.</information><count>1003</count></
ns2:jaxWsTest1></S:Body></S:Envelope>
2008/10/14 13:09:45 com.cosminexus.xml.ws.transport.http.client.HttpTransportPipe
process
Information: KDJW30012-I http client message
---[HTTP response 200]---
HTTP/1.1 200 OK
Keep-alive: timeout=3, max=100
Date: Tue, 14 Oct 2008 04:09:44 GMT
Content-type: text/xml;charset=utf-8
Connection: Keep-Alive
Transfer-encoding: chunked
Server: Cosminexus HTTP Server
<?xml version="1.0" ?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/
envelope/"><S:Body><ns2:jaxWsTest1Response xmlns:ns2="http://example.com/
sample"><return>We've got your #1003 message &amp;quot;Invocation test.&amp;quot;! It's
2008.10.14 13:09:45 now. See ya!</return></ns2:jaxWsTest1Response></S:Body></
S:Envelope>
```

The following example describes a communication log output for the JAX-RS functionality:

- Server side

```
Sep 10, 2011 7:57:48 PM com.cosminexus.jersey.api.container.filter.LoggingFilter
filter
INFO: KDJJ30013-I 1 * Server in-bound request
1 > POST http://sample.com/example/root/path%20value;matrix=matrix%20value?
query=query%20value
1 > user-agent: Java/1.5.0_11
1 > host: sample.com
1 > accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
1 > connection: keep-alive
1 > content-type: application/x-www-form-urlencoded
1 > content-length: 19
1 >
form=form%2520value

Sep 10, 2011 7:57:48 PM com.cosminexus.jersey.api.container.filter.LoggingFilter
$Adapter finish
INFO: KDJJ30015-I 1 * Server out-bound response
1 < 200
1 < Content-Type: text/html
1 <
matrix%20value path%20value form%20value query%20value
```

- Client side

```
Aug 27, 2012 10:30:06 AM com.cosminexus.jersey.api.client.filter.LoggingFilter
printRequest
INFO: KDJJ30026-I 1 * Client out-bound request
1 > POST http://sample.com/example
1 > Content-Type: text/plain
1 >
requestEntity

Aug 27, 2012 10:30:06 AM com.cosminexus.jersey.api.client.filter.LoggingFilter
printResponse
INFO: KDJJ30027-I 1 * Client in-bound response
1 < 200
1 < Content-Type: text/html
1 <
responseEntity
```

In the case of a JAX-RS functionality, even if the property is not specified in the action definition file, you can set up whether to output communication logs for the Web applications having Web resources, by including the following servlet initialization parameters and values in `web.xml`.

No.	Servlet initialization parameter	Value	Operation
1	com.sun.jersey.spi.container.ContainerRequestFilters	com.cosminexus.jersey.api.container.filter.LoggingFilter	The HTTP request information is output to a communication log.
2	com.sun.jersey.spi.container.ContainerResponseFilters	com.cosminexus.jersey.api.container.filter.LoggingFilter	The HTTP response information is output to a communication log.

#

If you specify a value other than the values mentioned in the above table, the settings will be ignored.

If a property is specified in the action definition file, you cannot set up whether to output communication logs for the Web applications having Web resources.

You can also set up whether to control the output of the entity information for the Web applications having Web resources. The following table describes the servlet initialization parameters and values.

No.	Servlet initialization parameter	Value	Operation
1	com.sun.jersey.config.feature.logging.DisableEntityLogging	True	The entity information cannot be output to a communication log.
		False	The entity information cannot be output to a communication log.

#

If you specify a value other than true or false, the value of the property acquired from the common definition file (cjrconf.properties) is used and the entity information is output to the communication log. Note that the value specified by using web.xml is ignored.

(3) Character encoding in communication logs

Regardless of the character encoding in a SOAP Message (for Web Services) and an HTTP message (for Web resources), the character encoding specified in the property of the following action definition files is applied to the communication log. By default, the platform dependent encoding is used in the communication log.

com.cosminexus.jaxws.logger.runtime.transport.encoding property (for Web Services)

com.cosminexus.jaxrs.logger.runtime.transport.encoding property (for Web resources)

If the character encoding in a message and a communication log differs, some of the characters in the communication log might become invalid depending on the combination. For example, if the character encoding in a message is UTF-8 and the character encoding in the communication log is MS932, and if the message contains the characters that do not exist in MS932, such characters become invalid in the communication log. Therefore, when developing Web Services or Web resources, we recommend that you define the available character codes or character sets, and report the same to a Web Service client or a Web resource client developer (WSDL public destination).

Points to note depending upon the functionality used:

- When sending or receiving an attachment (wsi:swaRef format, MTOM/XOP specification format), the characters in the communication log and the SOAP envelope of the root part might become invalid because the attachments might include a binary data or a character string in a different character encoding format.

(4) Names of the HTTP header output to the communication log

The name of the HTTP header that is output to the communication log always has only the first character in upper case, regardless of the HTTP messages actually sent and received (HTTP messages including SOAP messages).

Example: Content-type

39.3.6 Setting logs

Specify the output level, size, and number of files in the action definition file.

The following is an example of settings in the action definition file:

```
com.cosminexus.jaxws.logger.runtime.message.level=INFO
com.cosminexus.jaxws.logger.runtime.message.file_num=2
com.cosminexus.jaxws.logger.runtime.message.file_size=2097152

com.cosminexus.jaxws.logger.runtime.maintenance.level=ALL
com.cosminexus.jaxws.logger.runtime.maintenance.file_num=2
com.cosminexus.jaxws.logger.runtime.maintenance.file_size=16777216

com.cosminexus.jaxws.logger.runtime.exception.level=INFO
com.cosminexus.jaxws.logger.runtime.exception.file_num=2
com.cosminexus.jaxws.logger.runtime.exception.file_size=16777216

com.cosminexus.jaxws.logger.runtime.transport.client_dump=NONE
com.cosminexus.jaxws.logger.runtime.transport.server_dump=NONE
com.cosminexus.jaxws.logger.runtime.transport.encoding=DEFAULT

com.cosminexus.jaxws.logger.cjwsimport.message.level=INFO
com.cosminexus.jaxws.logger.cjwsimport.message.file_num=2
com.cosminexus.jaxws.logger.cjwsimport.message.file_size=2097152

com.cosminexus.jaxws.logger.cjwsimport.exception.level=INFO
com.cosminexus.jaxws.logger.cjwsimport.exception.file_num=2
com.cosminexus.jaxws.logger.cjwsimport.exception.file_size=16777216

com.cosminexus.jaxws.logger.cjwsimport.maintenance.level=ALL
com.cosminexus.jaxws.logger.cjwsimport.maintenance.file_num=2
com.cosminexus.jaxws.logger.cjwsimport.maintenance.file_size=16777216

com.cosminexus.jaxws.logger.apt.message.level=INFO
com.cosminexus.jaxws.logger.apt.message.file_num=2
com.cosminexus.jaxws.logger.apt.message.file_size=2097152

com.cosminexus.jaxws.logger.apt.exception.level=INFO
com.cosminexus.jaxws.logger.apt.exception.file_num=2
com.cosminexus.jaxws.logger.apt.exception.file_size=16777216

com.cosminexus.jaxws.logger.apt.maintenance.level=ALL
com.cosminexus.jaxws.logger.apt.maintenance.file_num=2
com.cosminexus.jaxws.logger.apt.maintenance.file_size=16777216

com.cosminexus.jaxws.logger.cjwsgen.message.level=INFO
com.cosminexus.jaxws.logger.cjwsgen.message.file_num=2
com.cosminexus.jaxws.logger.cjwsgen.message.file_size=2097152

com.cosminexus.jaxws.logger.cjwsgen.exception.level=INFO
com.cosminexus.jaxws.logger.cjwsgen.exception.file_num=2
com.cosminexus.jaxws.logger.cjwsgen.exception.file_size=16777216

com.cosminexus.jaxws.logger.cjwsgen.maintenance.level=ALL
com.cosminexus.jaxws.logger.cjwsgen.maintenance.file_num=2
com.cosminexus.jaxws.logger.cjwsgen.maintenance.file_size=16777216
```

39.3.7 Estimating the log

This subsection describes how to estimate each log file.

(1) Setting the model and log to be used

This point describes the method of estimating the log using the following model 1 and model 2 as examples:

Model 1: Normal system

A model when a request is sent from the Web Service and SEI returns the response normally.

Model 2: Abnormal system

A model when `RuntimeException` is thrown in SEI of the Web Service.

The log settings used in these models are as follows:

- For the operation log/ exception log/ maintenance log settings, default values are not changed.
- For the communication log settings, specify 'ALL' in each of the following properties:
  For the JAX-WS functionality

- com.cosminexus.jaxws.logger.runtime.transport.client_dump
- com.cosminexus.jaxws.logger.runtime.transport.server_dump

For the JAX-RS functionality

- com.cosminexus.jaxrs.logger.runtime.transport.server.level

For details about the log settings and default values, see *10.1.2 Settings for the common definition file*.

(2) Method of estimating the log

The following table lists and describes the log output results for model 1 and model 2.

In this table, 'At start' indicates when the application starts, 'At end' indicates when the application stops, and 'when 1 request is processed' indicates the processing of one request by SEI. The output amount shows the amount of log output in these periods.

Table 39‑15: Example of log output at start, when 1 request is processed, and at stop (model 1/ model 2)

No.	Log types	Output amount		
		At start	When 1 request is processed	At end
1	Operation log	0.7KB	Model 1: 0.0 KB Model 2: 0.3 KB	0.3KB
2	Exception log	0KB	Model 1: 0 KB Model 2#: 5 KB	0KB
3	Communication log	0KB	0.3 KB + HTTP request + HTTP responses	0KB
4	Maintenance log	2.4KB	Model 1: 3.2 KB Model 2: 3.6 KB	0.4KB

#

Becomes the amount of stack trace of the exception.

The formula for estimating the log is as follows. Based on this result, each log file obtains one file size and number of files.

```
Amount-of-log-output-per-unit-time =
 Amount-of-output-at-start
 + Amount-of-output-when-one-request-is-processed-in-model-1 × Total-number-of-
requests × Percentage-of-normal-case-requests
 + Amount-of-output-when-one-request-is-processed-in-model-2 × Total-number-of-
requests × Percentage-of-abnormal-case-requests
 + Amount-of-output-at-end
```

The following points describe how to estimate each file using examples based on this formula:

The example assumes a system in which there are 100 requests on an average in 1 minute and the log file size and number are estimated such that log is not wrapped around for 3 hours. Among all the requests, if you assume that requests ending normally are 80% and requests ending abnormally are 20%, because of 50-requests/ minute, total 18,000 requests (100 requests/ minute × 180 minutes) arrive in 3 hours.

In this case, if you calculate the amount of operation log output, the result will be as follows:

```
0.7 KB (Amount-of-output-at-start)
+ {0.0 KB × 18000}(Amount-of-output-when-all-requests-are-processed-in-model-1) ×
0.8 (Percentage-of-normal-case-requests)
```

```
+ {0.3 KB × 18000}(Amount-of-output-when-all-requests-are-processed-in-model-2) ×
0.2 (Percentage-of-abnormal-case-requests)
+ 0.3 KB (Amount-of-output-at-end)
=1081 KB
```

As a result of calculation, the file size and number of files meets the default value.

Similarly, if you calculate the amount of maintenance log, the result will be as follows:

```
2.4 KB (Amount-of-output-at-start)
+{3.2 KB × 18000}(Amount-of-output-when-all-requests-are-processed-in-model-1) ×
0.8 (Percentage-of-normal-case-requests)
+{3.6 KB × 18000}(Amount-of-output-when-all-requests-are-processed-in-model-2) ×
0.2 (Percentage-of-abnormal-case-requests)
+0.4 KB (Amount-of-output-at-end)
=59043 KB ≒ 57.6 MB
```

If you assume 15 MB per log file as calculation result, the files count becomes four.

Notes

In the communication log, the amount of output changes according to the HTTP request and HTTP response. When you use attachments for communication, the attachment contents are recorded in the communication log as well, so when you use a particularly large attachment, you need to consider the size and number of files of the communication log file.

39.4 Performance analysis trace (PRF)

The Trace based performance analysis indicates trace information for checking the performance analysis and failure. By referencing the Trace based performance analysis, you can analyze the degree of achievement of request processing when a system performance bottleneck or failure occurs.

For details about the performance analysis trace provided by Cosminexus system, *see 4.6 Performance analysis trace* in the *uCosminexus Application Server Maintenance and Migration Guide*.

39.4.1 Collection level of Trace based performance analysis

The following table lists and describes the collection levels of the performance analysis trace that can be used with the Cosminexus JAX-WS functionality or the JAX-RS functionality.

Table 39–16: List of collection levels of Trace based performance analysis

No.	Collection level	Purpose
1	Standard level (Default)	The trace information that can identify the boundary (entry and exit) between the Cosminexus JAX-WS functionality or the JAX-RS functionality and other functionality is output.
2	Detailed level	The trace information about the internal processing of the Cosminexus JAX-WS functionality or the JAX-RS functionality is output in addition to the contents of the standard level.
3	Maintenance level	This is a level used for collecting the maintenance information when failure occurs and is not used normally.

39.4.2 Trace output information of Trace based performance analysis

The following table lists and describes the trace output information of the performance analysis trace that is output with the Cosminexus JAX-WS functionality or the JAX-RS functionality.

Table 39–17: Trace output information of Trace based performance analysis

No.	File item	Contents
1	Event ID	This is the event ID of the trace collection point.
2	Client application information	This is unique information between the Web Services client and Web Services within the valid range of the root application information. However, this is not unique for the WS-RM 1.2 functionality. See *Option information*
3	Root application information	This is unique information in the processing for a series of requests. However, this is not unique for the WS-RM 1.2 functionality. See *Option information*
4	Return code	This is the type of trace collection points (normal end or abnormal end).
5	Interface name	This is the class name of the trace collection point.
6	Operation name	This is the method name of the trace collection point.
7	Optional information	This is the option information.

This subsection describes the valid range of client application information and root application information and the trace collection points.

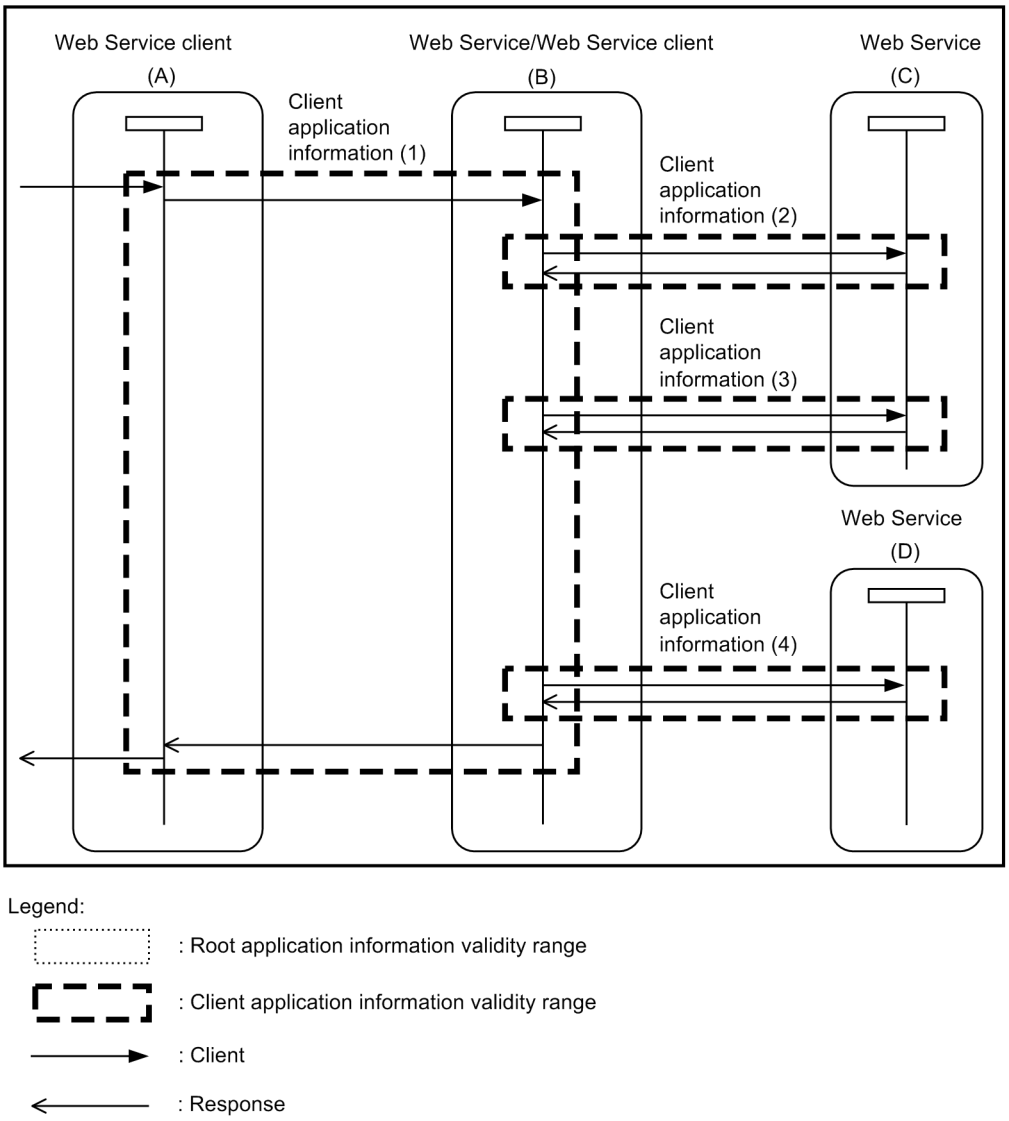
(1) Valid range of client application information and root application information

This subsection describes the valid range of client application information and root application information separately for a Web Service and a Web resource.

(a) For a Web Service

The following figure describes the valid range of the client application and root application information for Web Services.

Figure 39–5: Valid range of client application information and root application information (for a Web Service)



The root application information (consists of the communication number, the process ID, and the IP address) is unique in the processing for a series of requests.

If a request is sent from the Web Services client (A) that is started by running the startup command (`cjclstartap`) of Java applications to the Web Service (B), the Web Service (B) becomes the Web Service client. Further, if Web Service (B) that becomes the Web Service client sends requests to the other Web Services(C) and (D), the value from A to D is a unique value.

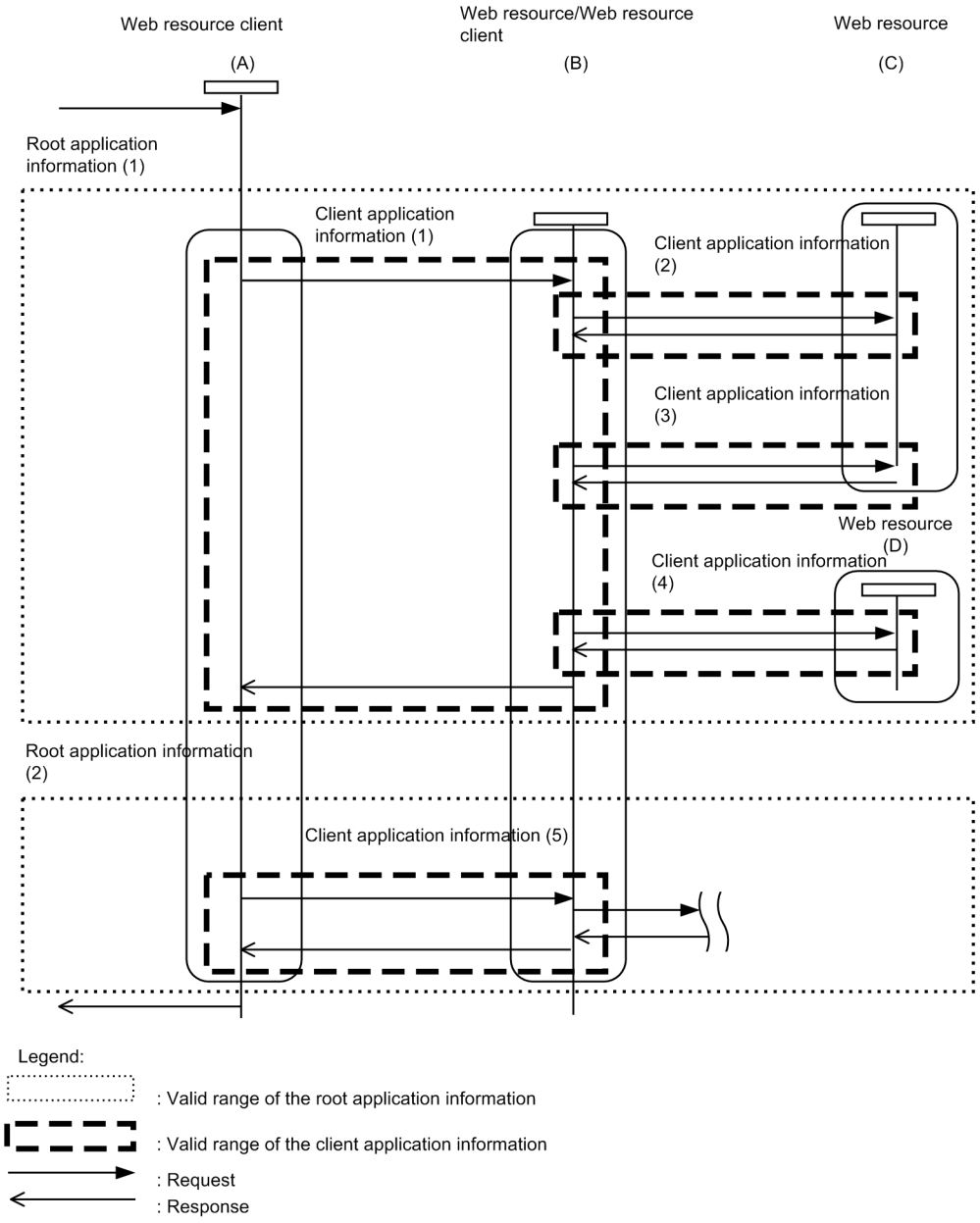
The client application information (consists of the communication number, the process ID, and the IP address) is unique in the processing for a series of requests of the Web Services client and Web Service within the valid range of the root application information.

The client application information (1) to (4) form different client application information.

(b) For a Web resource

This subsection describes the valid range of client application information and root application information for Web resources, based on the following figure.

Figure 39−6: Valid range of client application information and root application information (for a Web resource)



In the above figure, the Web resource client (A) and the Web resource or Web resource client (B) use a client API for RESTful Web Services. The Web resource or Web resource client (B), the Web resource (C), and the Web resource (D) are the Web resources operating on the JAX-RS engine.

The root application information will be unique throughout processing of all the requests made after the client calls a client API for RESTful Web Services until the client receives an HTTP response.

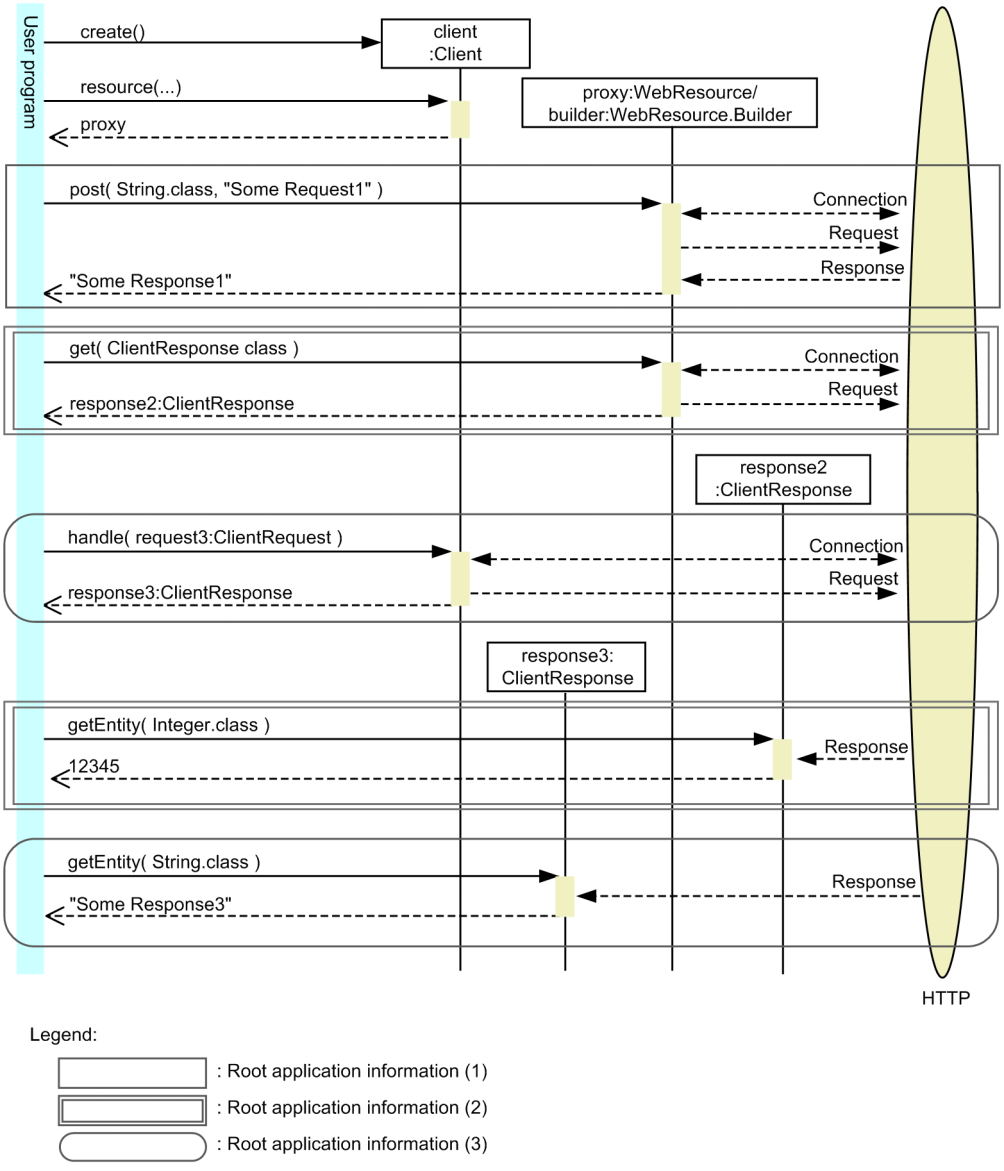
On the other hand, each call will have unique client application information. Each of the client application information from (1) to (5) in the figure has a unique value.

Even when the client retrieves an entity of the HTTP response from the `ClientResponse` object returned by either of the following operations, inconsecutively to the respective operations, the root application information and the client application information will be unique across the series of calls.

- The `handle` method of the `Client` class
- The `method` method of the `WebResource` class or the `WebResource.Builder` class or all the methods compliant with the HTTP methods.

The following figure shows the concept. Note that the client application information is omitted in the figure because the scope of the client application information and the root application information is the same.

Figure 39-7: Inconsecutive calling and application information



Accordingly, the root application information (root application information (2) in the figure) used in inconsecutive calling of the `get(Class)` method of the `WebResource` class and the `getEntity(Class)` method of the `ClientResponse` class is unique, similar to the root application information (root application information (1) in the figure) of the `post(Class, Object)` method of the `WebResource` class that is executed throughout after the HTTP communication is established until the requests and responses are received.

Furthermore, the root application information (root application information (3) in the figure), which is used when calling the `handle(Class)` method of the `Client` class and the `getEntity(Class)` method of the `ClientResponse` class inconsecutively, is also unique.

(2) Trace collection point of Trace based performance analysis

The following figure shows the trace collection points of the trace based performance analysis output with the Cosminexus JAX-WS functionality. Note that when you use the WS-RM 1.2 functionality, other Trace based performance analysis that is not coded will also output. For details, see the section *39.4.2(2)(c) Trace collection points when the WS-RM 1.2 functionality is used*.

(a) Trace collection point when the POJO Web service is invoked

*Figure 39-8*and *Figure 39-9* show the request-response operation and the one-way operation for the trace collection points when the POJO Web Service is invoked. For the trace collection points when the `Provider Implementation` class returns `null` in the one-way operation, see *Figure 39-8*.

Figure 39-8: Trace collection point when the POJO Web Service is invoked (request-response operation)

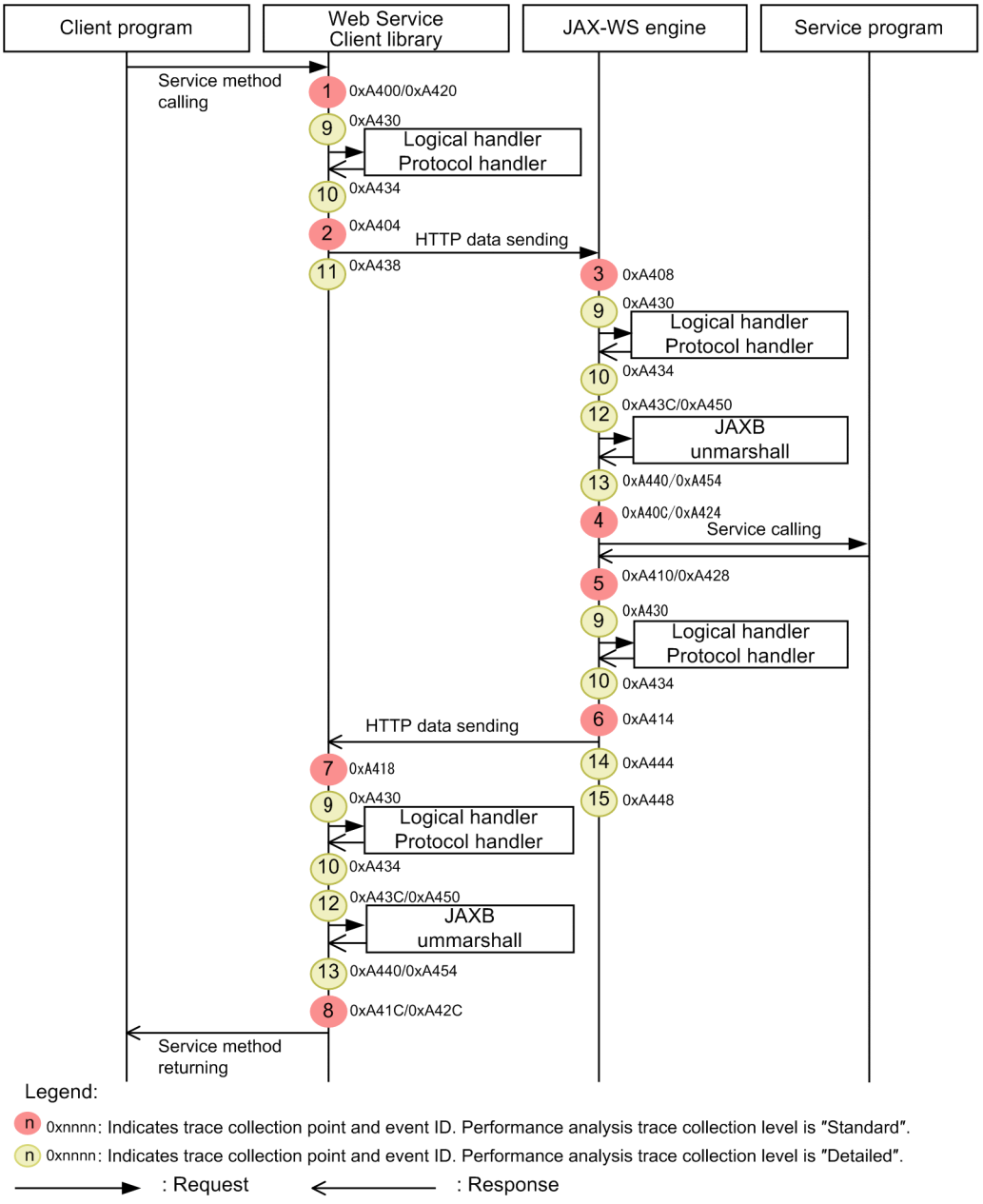
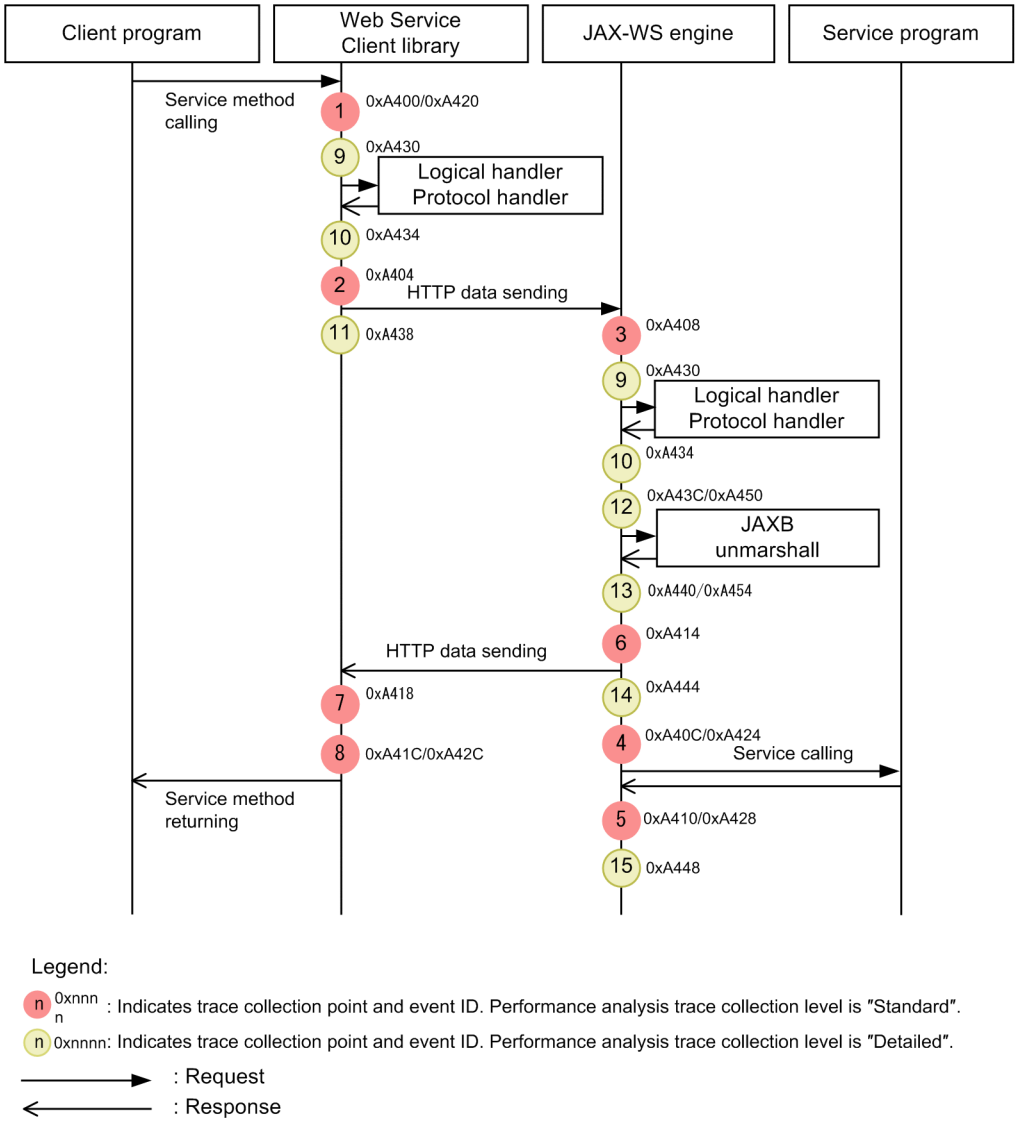


Figure 39-9: Trace collection points when the POJO Web Service is invoked (one-way operation)



The following table shows the relationship between each trace collection point and event ID:

Table 39-18: Trace collection point and output information in the standard level (JAX-WS functionality and POJO Web Service)

No. in figure	Collection point		Output information	
			Event ID	Optional information
1	When the Web Service client library starts	For stub-based Web Service clients	0xA400	--
		For dispatch-based Web Service clients	0xA420	
2	Before the HTTP message of the Web Service client library is sent		0xA404	End point URL
3	When the servlet of the JAX-WS engine starts		0xA408	Context root
4	Before the Web Service is invoked	For Web Service Implementation Classes	0xA40C	Service method name

No. in figure	Collection point		Output information	
			Event ID	Optional information
4	Before the Web Service is invoked	For Provider Implementation Classes	0xA424	--
5	After the Web Service is invoked	For Web Service Implementation Classes	0xA410	Exception name for abnormal end.
		For Provider Implementation Classes	0xA428	
6	Before the HTTP message of the JAX-WS engine is sent		0xA414#	--
7	After the HTTP message of the Web Service client library is received		0xA418	Exception name for abnormal end.
8	When the Web Service client library ends	For stub-based Web Service clients	0xA41C	Exception name for abnormal end.
		For dispatch-based Web Service clients	0xA42C	

Legend:

--: There is no optional information.

#

The trace is not collected when the `Provider Implementation` class returns `null` after a Web Service is called.

Table 39-19: Trace collection point and output information in the detail level (JAX-WS functionality and POJO Web service)

No.in figure	Collection point		Output information	
			Event ID	Optional information
9	Before the handler is invoked		0xA430#	Handler position and handler class name
10	After the handler is invoked		0xA434#	Exception name for abnormal end.
11	After the HTTP message of the Web Service client library is sent		0xA438	Exception name for abnormal end.
12	Before un-marshalling	For stub-based Web Service clients and Web Service Implementation Classes	0xA43C	--
		For dispatch-based Web Service clients and Provider Implementation Classes	0xA450#	
13	After un-marshalling	For stub-based Web Service clients and Web Service Implementation Classes	0xA440	--
		For dispatch-based Web Service clients and Provider Implementation Classes	0xA454#	
14	After the HTTP message of the JAX-WS engine is sent		0xA444#	Exception name for abnormal end.
15	When the servlet of the JAX-WS engine ends		0xA448	Exception name for abnormal end.

Legend:

--: There is no optional information.

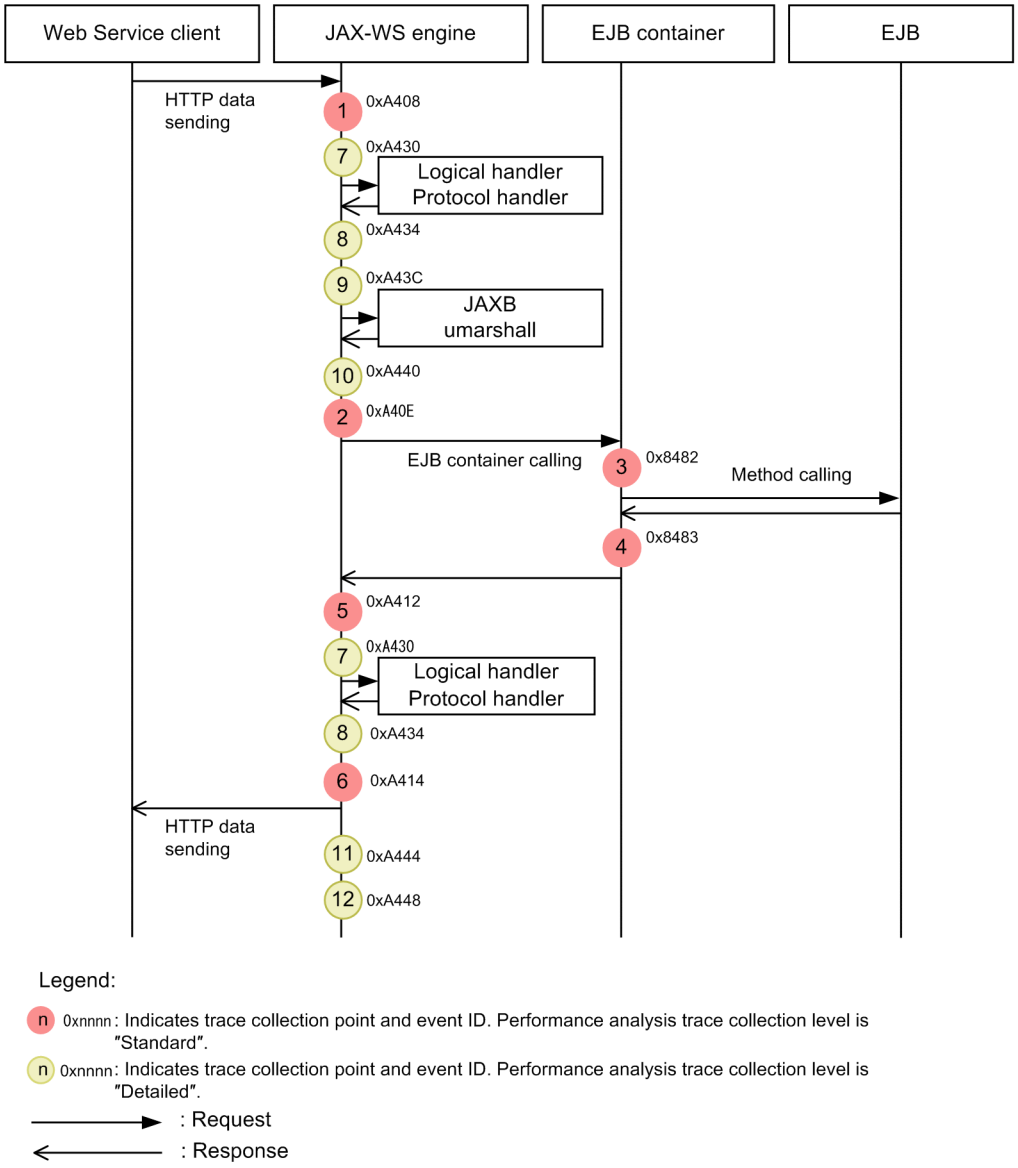
#

The trace is not collected when the `Provider Implementation` class returns `null` after a Web Service is called.

(b) Trace collection point when invoking the EJB Web Service

The following figure shows the trace collection point for a request-response operation when invoking the EJB Web Service. Note that the trace collection point of the Web Service client is same as when the POJO Web Service is invoked. Furthermore, for the trace collection points for Web Services in a one-way operation, substitute invoking a service by invoking an EJB container when referencing *Figure 39-9 Trace collection points when the POJO Web Service is invoked (a one-way operation)*.

Figure 39－10: Trace collection point when invoking the EJB Web Service (request-response operations)



Legend:

n 0xnnnn: Indicates trace collection point and event ID. Performance analysis trace collection level is "Standard".

n 0xnnnn: Indicates trace collection point and event ID. Performance analysis trace collection level is "Detailed".

: Request

: Response

The following table lists and describes the relationship between each trace collection point and event ID:

Table 39-20: Trace collection point and output information in the standard level (JAX-WS functionality and POJO Web service)

No. in Figure	Collection point		Output information	
			Event ID	Optional information
1	When the servlet of a JAX-WS engine starts		0xA408	Context root
2	Before the EJB container is invoked	For Web Service Implementation Classes	0xA40E	Service method name
3	Before the EJB method is invoked		0x8482	--
4	After the EJB method is invoked		0x8483	--
5	After the EJB container is invoked	For Web Service Implementation Classes	0xA412	Exception name for abnormal end
6	After the HTTP message of the JAX-WS engine is sent		0xA414	Exception name for abnormal end

Legend:

--: There is no optional information.

Table 39-21: Trace collection point and output information in the detail level (JAX-WS functionality and POJO Web Service)

No. in Figure	Collection point		Output information	
			Event ID	Optional information
7	Before the handler is invoked		0xA430	Handler position and handler class name
8	After the handler is invoked		0xA434	Exception name for abnormal end.
9	Before un-marshaling	For Web Service Implementation Classes	0xA43C	--
10	After un-marshaling	For Web Service Implementation Classes	0xA440	--
11	After the HTTP message of the JAX-WS engine is sent		0xA444	Exception name for abnormal end
12	When the servlet of the JAX-WS engine ends		0xA448	Exception name for abnormal end

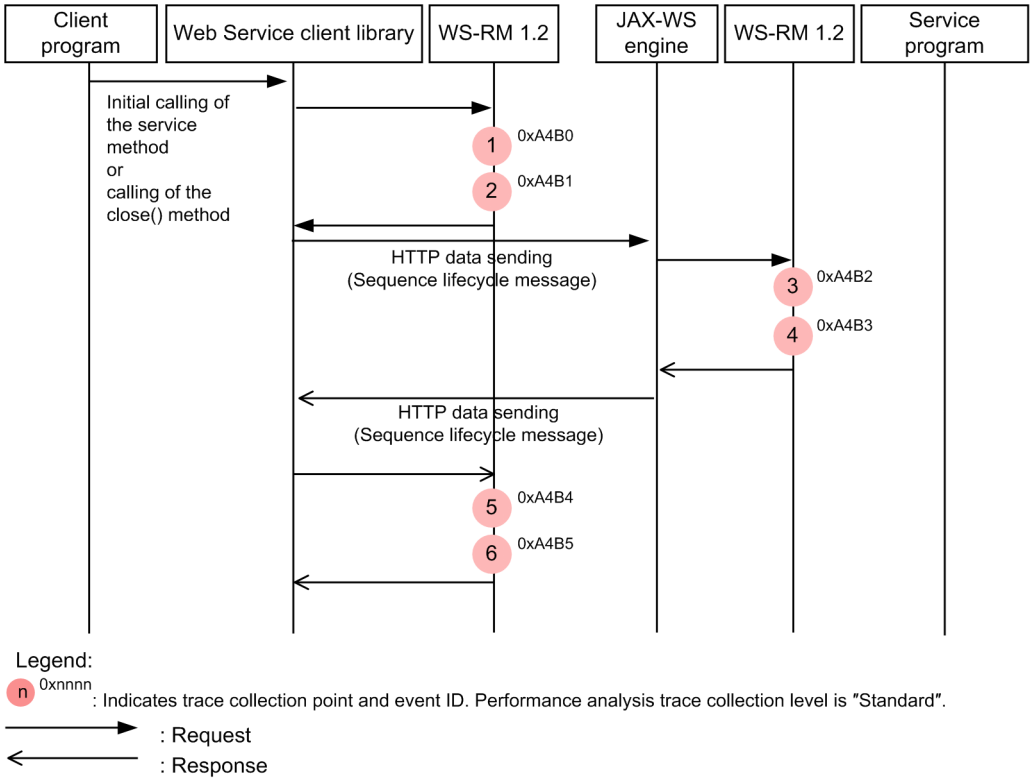
Legend:

--: There is no optional information.

(c) Trace collection points when the WS-RM 1.2 functionality is used

The following figure shows the trace collection points of the Trace based performance analysis output, when the WS-RM 1.2functionality is used to send and receive the sequence life cycle messages:

Figure 39-11: Trace collection points when the sequence life cycle messages are sent and received



The following table lists and describes the relationship between each trace collection point and event ID:

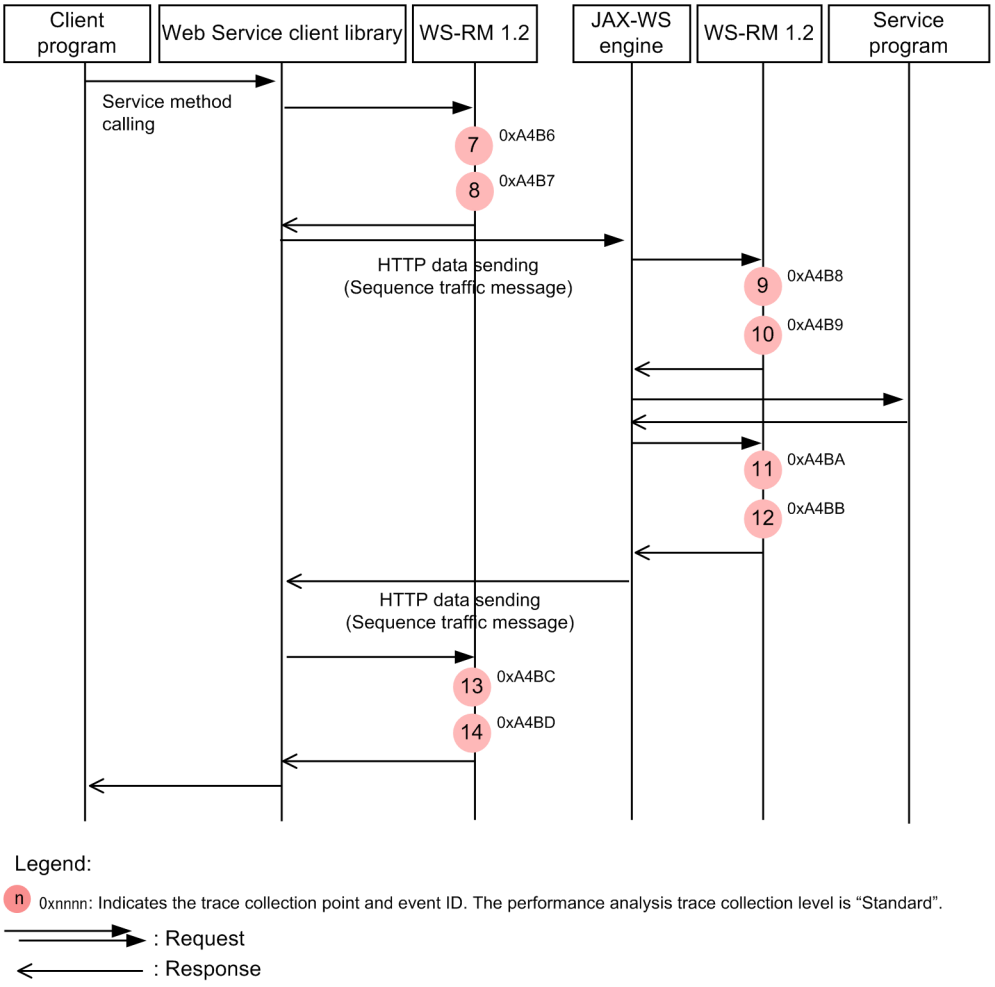
Table 39-22: Trace collection points and output information in the standard level (Life cycle messages of the WS-RM 1.2 functionality)

Number in figure	Collection point	Output information	
		Event ID	Optional information
1	When the WS-RM 1.2 in the Web Service client starts	0xA4B0	One of the following is output. • CreateSequence • CloseSequence, SequenceID • TerminateSequence, SequenceID
2	Before the message in the Web Service client sequence life cycle is sent	0xA4B1	One of the following is output. • CreateSequence • CloseSequence, SequenceID • TerminateSequence, SequenceID
3	When the WS-RM 1.2functionality in the Web Service client starts	0xA4B2	One of the following is output • CreateSequence • CloseSequence, SequenceID • TerminateSequence, SequenceID
4	When the WS-RM 1.2 functionality in the Web Service ends	0xA4B3	One of the following is output CreateSequence, SequenceID • CloseSequence, SequenceID • TerminateSequence, SequenceID • Exception name for abnormal end

Number in figure	Collection point	Output information	
		Event ID	Optional information
5	After the sequence life cycle message in the Web Service client is received.	0xA4B4	One of the following is output. • CreateSequence • CloseSequence, SequenceID • TerminateSequence, SequenceID • Exception name for abnormal end
6	When the WS-RM 1.2 functionality in the Web Service client ends	0xA4B5	One of the following is output. • CreateSequence, SequenceID • CloseSequence, SequenceID • TerminateSequence, SequenceID • Exception name for abnormal end

The following figure shows the trace collection points of the Trace based performance analysis for the sequence traffic messages of the WS-RM 1.2functionality:

Figure 39–12: Trace collection points for the sequence traffic messages



The following table lists and describes the relationship between each trace collection point and event ID:

Table 39-23: Trace collection points and output information in the standard level(Sequence traffic messages of the WS-RM 1.2 functionality)

Number in figure	Collection point	Output information	
		Event ID	Optional information
7	When the WS-RM 1.2 functionality in the Web Service client starts in the request	0xA4B6	SequenceID, MessageNumber
8	When the WS-RM 1.2 functionality in the Web Service client ends in the request	0xA4B7	One of the following is output. • SequenceID, MessageNumber • Exception name for abnormal end
9	When the WS-RM 1.2 functionality in the Web Service starts in the request	0xA4B8	SequenceID,MessageNumber
10	When the WS-RM 1.2 functionality in the Web Service ends in the request	0xA4B9	One of the following is output • SequenceID, MessageNumber • Exception name for abnormal end
11	When the WS-RM 1.2 functionality in the Web Service starts in the response .#1	0xA4BA	SequenceID, MessageNumber
12	When the WS-RM 1.2 functionality in the Web Service ends in the response #2	0xA4BB	One of the following is output • SequenceID, MessageNumber • Exception name for abnormal end
13	When the WS-RM 1.2 functionality in the Web Service client starts in the response	0xA4BC	SequenceID, MessageNumber
14	When the WS-RM 1.2 functionality in the Web Service client ends in the response	0xA4BD	One of the following is output. • SequenceID • Exception name for abnormal end.

#1

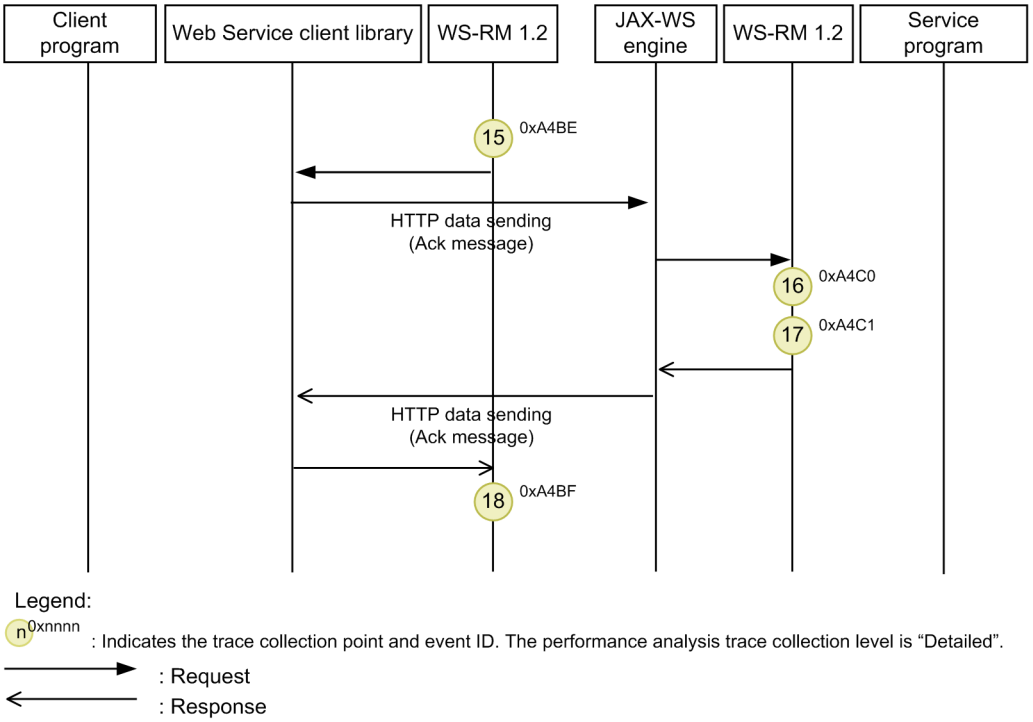
Not output when duplicate message is received.

#2

Not output when HTTP status code 202 is returned on receiving the duplicate message.

The following figure shows the trace collection points of the Trace based performance analysis for the Ack messages of the WS-RM 1.2functionality:

Figure 39-13: Trace collection point of Ack message



The following table lists and describes the relationship between each trace collection point and event ID:

Table 39-24: Trace collection points and output information in the detailed level (Ack message of the WS-RM 1.2 functionality)

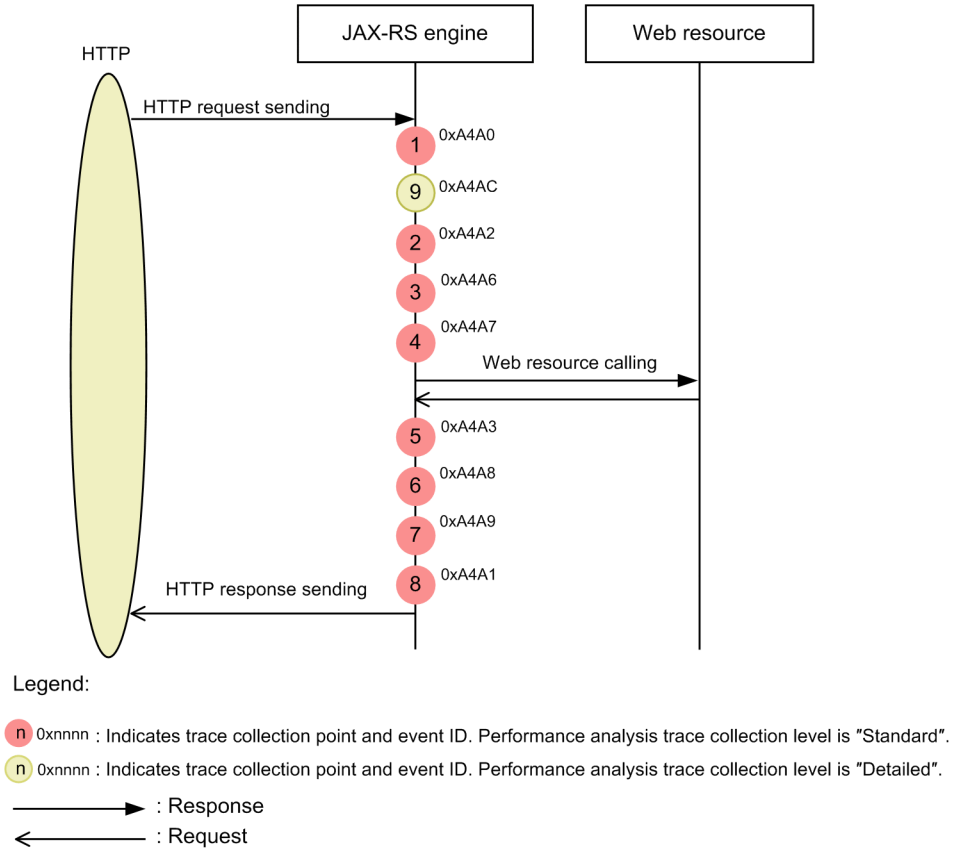
Number in figure	Collection point	Output information	
		Event ID	Optional information
15	When the WS-RM 1.2 functionality in the Web Service client starts	0xA4BE	SequenceID
16	When the WS-RM 1.2 functionality in the Web Service starts	0xA4C0	SequenceID
17	When the WS-RM 1.2 functionality in the Web Service ends	0xA4C1	One of the following is output. • SequenceID • Exception name for abnormal end.
18	When the WS-RM 1.2 functionality in the Web Service client ends	0xA4BF	One of the following is output. • SequenceID • Exception name for abnormal end.

(d) Trace collection points when calling RESTful Web Services

The following figure shows the trace collection points when calling RESTful Web Services (Web Services) separately for resources and clients.

For resources

Figure 39-14: Trace collection points when calling RESTful Web Services(**for resources)**



The following table lists the trace collection points and their corresponding event IDs.

Table 39-25: Trace collection points and the information output at the standard level (JAX-RS functionality and RESTful Web Services)

Number in figure	Collection point	Output information	
		Event ID	Optional information
1	When starting a servlet of the JAX-RS engine	0xA4A0	--
2	Before calling a Web resource	0xA4A1	One of the following is output: • JResponseOutInvoker • ObjectOutInvoker • ResponseOutInvoker • TypeOutInvoker • VoidOutInvoker • VoidVoidMethodInvoker HttpReqResDispatcher
3	Before unmarshalling	0xA4A6	--
4	After unmarshalling	0xA4A7	The exception name when an exception is thrown
5	After calling a Web resource	0xA4A3	One of the following is output.# • JResponseOutInvoker • ObjectOutInvoker

Number in figure	Collection point	Output information	
		Event ID	Optional information
5	After calling a Web resource	0xA4A3	• ResponseOutInvoker • TypeOutInvoker • VoidOutInvoker • VoidVoidMethodInvoker • HttpReqResDispatcher
6	Before marshalling	0xA4A8	--
7	After marshalling	0xA4A9	The exception name when an exception is thrown
8	Before the JAX-RS engine sends an HTTP message	0xA4A1	The exception name when an exception is thrown

Legend
--: No optional information is available.

Table 39–26: Trace collection points and the information that is output at the standard level (JAX-RS functionality and RESTful Web Services)

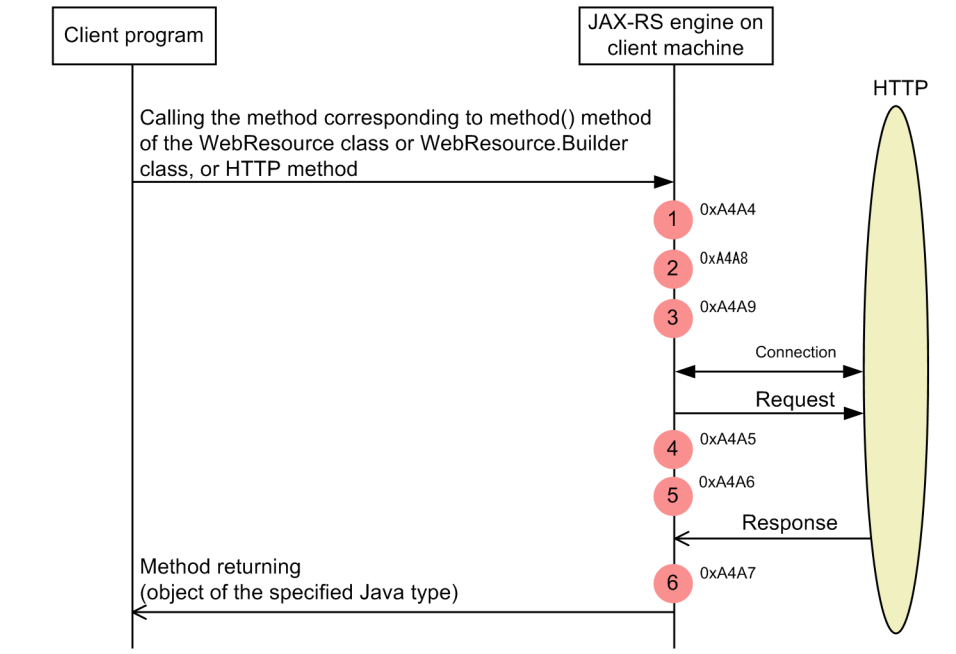
Number in figure	Collection point	Output information	
		Event ID	Optional Information
9	When matching an HTTP request and a resource	0xA4AC	One of the following is output: • accept root resource classes • forwarding view to JSP page • accept implicit view • match path • accept sub-resource methods • accept resource methods • matched sub-resource method • matched resource method • accept resource message in ResourceClassRule • accept resource message in ResourceOjectRule • accept right hand path redirect • accept right hand path • accept sub-resource locator • accept termination • matched message body reader • matched exception mapper • mapped exception to response • matched message body writer

For clients

The following figures show each use case described in *11.4.1 Use case of a Web resource client*.

Figure 39－15: Trace collection points when invoking RESTful Web Services (for a client)

●When sending and receiving the HTTP request and HTTP response by specifying the Java type



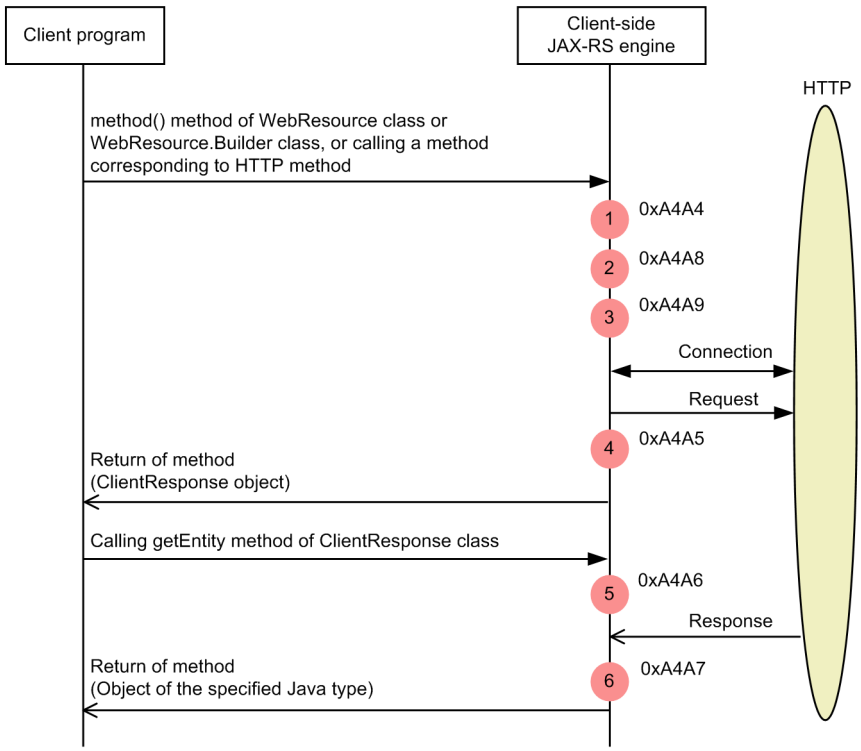
Legend:

n 0xnnnn : Indicates trace collection point and event ID. Performance analysis trace collection level is "Standard".

: Request

: Response

● When sending an HTTP request by specifying the Java type and receiving an HTTP response with the general-purpose type



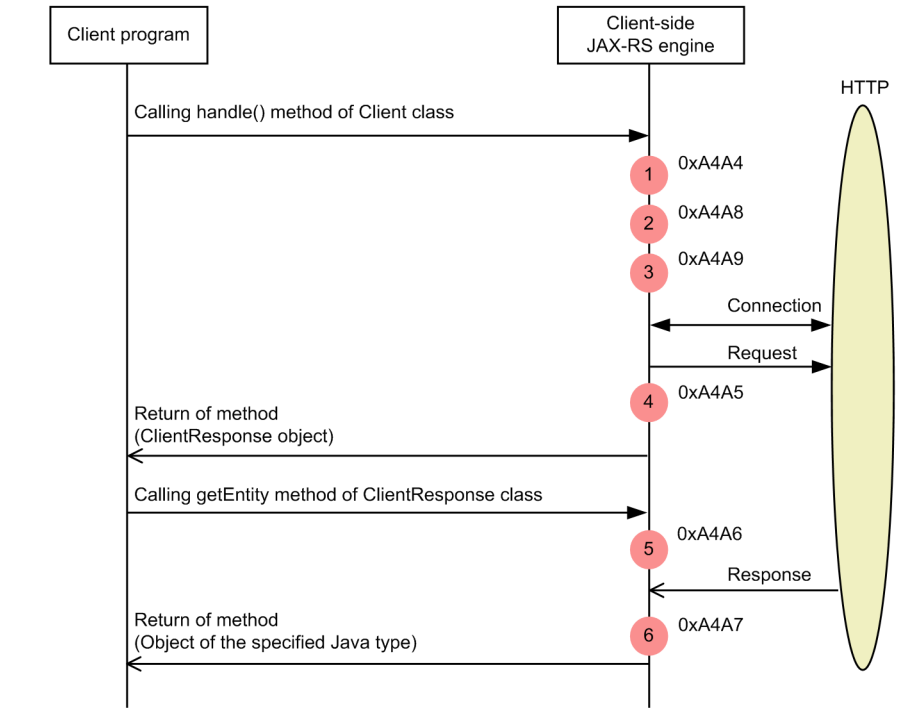
Legend:

n 0xnnnn : Indicates trace collection point and event ID. Performance analysis trace collection level is "Standard".

→ : Request

← : Response

● When sending and receiving an HTTP request and an HTTP response by the general-purpose type



Legend:

n 0xnnnn : Indicates trace collection point and event ID. Performance analysis trace collection level is "Standard".

: Request

: Response

The following table lists and describes the relationship between each trace collection point and event ID.

Table 39-27: Trace collection points in the standard level and information that is output (client APIs for the JAX-RS functionality and RESTful Web Services)

No. in figure	Collection point	Output information	
		Event ID	Optional information
1	When calling an HTTP method starts	0xA4A4	End point URL
2	Before marshalling	0xA4A8	--
3	After marshalling	0xA4A9	--
4	When calling an HTTP method is complete	0xA4A5	The exception name when an exception is thrown
5	Before unmarshalling	0xA4A6	--
6	After unmarshalling	0xA4A7	The exception name when an exception is thrown

Legend

--: No optional information is available.

39.4.3 Method of performance analysis based on Trace based performance analysis

The Trace based performance analysis file is a text file that outputs the Trace based performance analysis in a CSV format.

(1) For Web Services

This subsection describes how to use the performance analysis trace file for analyzing the response time of all the Web Services including the JAX-WS engine and the response time of a Web Service (UP). Note that this subsection describes an example of Web Services developed using Web Service Implementation Classes and stub-based Web Service clients. The event IDs might differ for other Web Services and Web Service clients, so substitute and read as and when required.

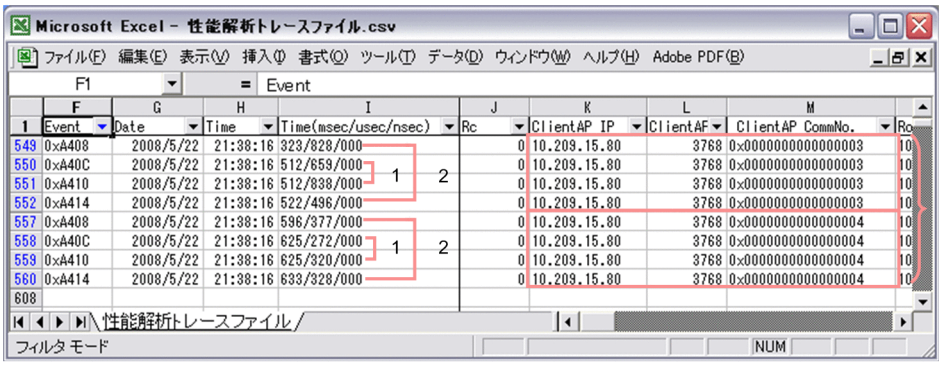
Using the event IDs '0xA408', '0xA40C', '0xA410', and '0xA414' as the keys, the Trace based performance analysis file will be filtered. The following table lists and describes the trace collection points for each event ID.

Table 39-28: Trace collection points corresponding to the event IDs to be filtered

No.	Event ID	Trace collection points
1	0xA408	When the servlet of the JAX-WS engine starts
2	0xA40C	Before the Web Service is invoked
3	0xA410	After the Web Service is invoked
4	0xA414	Before a SOAP Message of the JAX-WS engine is sent

The following figure shows an example of filtering the trace based performance analysis files by setting the event IDs '0xA408', '0xA40C', '0xA410', and '0xA414' as keys, when a request is sent twice.

Figure 39-16: Example of filtering the Trace based performance analysis



1. Time difference between 0xA40C and 0xA41O
   Response time of the created program
2. Time difference between 0xA408 and 0xA414
   Response time of all the Web Services
3. Client AP information is identical (same request is being processed)

You can check the request processing status from the client application information. If the same request is being processed, the same client application information is output.

You can analyze the response time of all the Web Services including the JAX-WS engine from the trace collection time of the event IDs '0xA408' and '0xA414' in the identical client application information. You can also analyze the response time of a Web Service (UP) from the trace collection point of '0xA40C' and '0xA410'.

(2) For Web resources

This subsection describes how to analyze the response time of Web resources including the JAX-RS engine and the response time of a Web resource (UP) by using the performance analysis trace file.

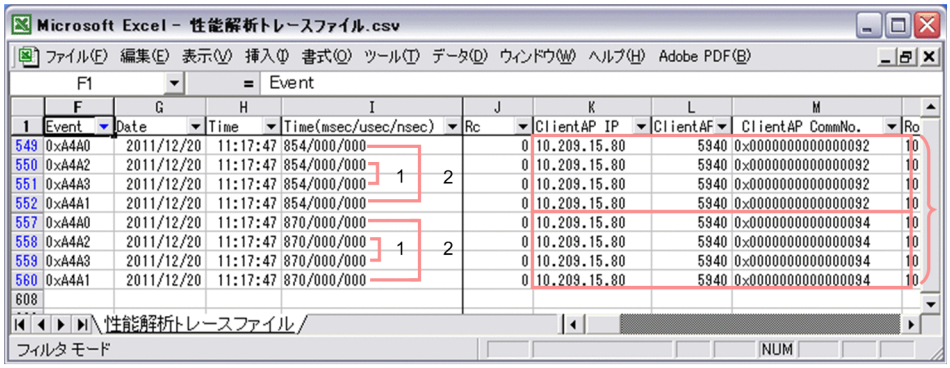
You filter the performance analysis trace file by setting the event IDs [0xA4A0], [0xA4A2], [0xA4A3], and [0xA4A1] as keys. The following table lists the event IDs and their trace collection points.

Table 39-29: Trace collection points for the event IDs to be filtered

No.	Event ID	Trace collection point
1	0xA4A0	When starting a servlet of a JAX-RS engine
2	0xA4A2	Before calling a Web resource
3	0xA4A3	After calling a Web resource
4	0xA4A1	Before sending an HTTP message of a JAX-RS engine

The following figure shows an example of filtering the performance analysis trace files by setting the event IDs [0xA4A0], [0xA4A2], [0xA4A3], and [0xA4A1] as keys, when a request is sent twice.

Figure 39-17: Example of filtering performance analysis traces



1. Time difference between 0xA4A2 and 0xA4A3
   Response time of the created program
2. Time difference between 0xA4A0 and 0xA4A1
   Response time of the entire Web resource
3. Client AP information is identical (same request is being processed)

You can check the request processing status from the client application information. When processing the same request, the same client application information is output.

You can analyze the response time of all the Web resources including the JAX-RS engine, starting from the time of the trace collection for the event IDs [0xA4A0] and [0xA4A1] of the same client application information. Also, you can analyze the response time of the Web resource (UP) starting from the time of the trace collection for [0xA4A2] and [0xA4A3].

Appendixes

A. Migrating from an Earlier Version

This appendix describes the migration of SOAP applications developed by using the SOAP application development support function and SOAP Communication Infrastructure that are functionality of earlier versions.

The SOAP applications and SOAP clients developed using the SOAP application development support function and the SOAP Communication Infrastructure can be used in the same environment by installing an upgraded version. In this case, specific settings are not required.

When you use the SOAP applications and SOAP client developed using the SOAP application development support function and the SOAP Communication Infrastructure on the JAX-WS engine, the environment switches to that of the JAX-WS engine after installing the upgraded version.

Also, the Web Service and Web Service client are created again. At this time, create the Web Service and Web Service client in compliance with the support range of JAX-WS 2.1 specifications and Cosminexus JAX-WS functionality.

A.1 Installing an upgraded version

The SOAP application development support function, SOAP Communication Infrastructure, JAX-WS functionality, and JAX-RS functionality are installed when installing Cosminexus.

This subsection describes the procedure and precautions for installing an upgraded version on a machine that has an earlier version installed.

(1) Migrating a J2EE server

You install an upgraded version of Cosminexus in accordance with the migration procedure described in *10.3 Migration procedure of Application Server* in the *uCosminexus Application Server Maintenance and Migration Guide* (in the case of update installation) or *3.9.2 Migration procedure of Application Server* in the *uCosminexus Application Server Maintenance and Migration Guide*.

If you install an upgraded version, the settings are such that SOAP application development support function and the SOAP Communication Infrastructure are used.

(2) Un-deploying a Web Service

When you migrate a Web Service (or a SOAP application), first un-deploy the Web Service (or the SOAP application) deployed on the J2EE server.

(3) Switching the operating environment

The switching of the SOAP application development support function, SOAP Communication Infrastructure, JAX-WS engine, and the JAX-RS engine is defined in the option definition file for J2EE servers. This subsection describes the specified contents and the specification method of the option definition file for J2EE servers.

For details on the option definition file for J2EE servers, see *2.3 usrconf.cfg (Option definition file for J2EE servers)* in the manual *uCosminexus Application Server Definition Reference Guide*.

- **When the JAX-WS and JAX-RS engines are used**

  When you use the JAX-WS and **JAX-RS** engines, enable the `cjjaxws.jar` and `cjjaxrs.jar` lines in `add.class.path` of the option definition file for J2EE servers and disable the `hitsaaj.jar` line. Specify the definition according to the following example:

```
...
#add.class.path=<cosminexus.home>\c4web\lib\hitsaaj.jar
add.class.path=<cosminexus.home>\jaxws\lib\cjjaxws.jar
add.class.path=<cosminexus.home>\jaxrs\lib\cjjaxrs.jar
...
```

- **When the SOAP application development support function/ SOAP Communication Infrastructure is used**

  When you use the SOAP application development support function/ SOAP Communication Infrastructure, enable the `hitsaaj.jar` line in `add.class.path` of the option definition file for J2EE servers and disable the `cjjaxws.jar` line. Specify the definition according to the following example:

```
...
add.class.path=<cosminexus.home>\c4web\lib\hitsaaj.jar
#add.class.path=<cosminexus.home>\jaxws\lib\cjjaxws.jar
...
```

Notes

You specify the settings so as to avoid the inconsistency in the lines to be enabled and disabled. For example, the operations might not function properly if all the lines `hitsaaj.jar`, `cjjaxws.jar`, and `cjjaxrs.jar` are removed or added.

The methods of editing the option definition file for J2EE servers are as follows:

In both the described methods, restart the J2EE server after editing (storing) the option definition file for the J2EE server. If the J2EE server is not restarted, the edited contents are not applied.

(a) Directly editing the option definition file for J2EE servers

To directly edit the file, open the option definition file for a J2EE server stored in the following location, in a text editor and change the contents:

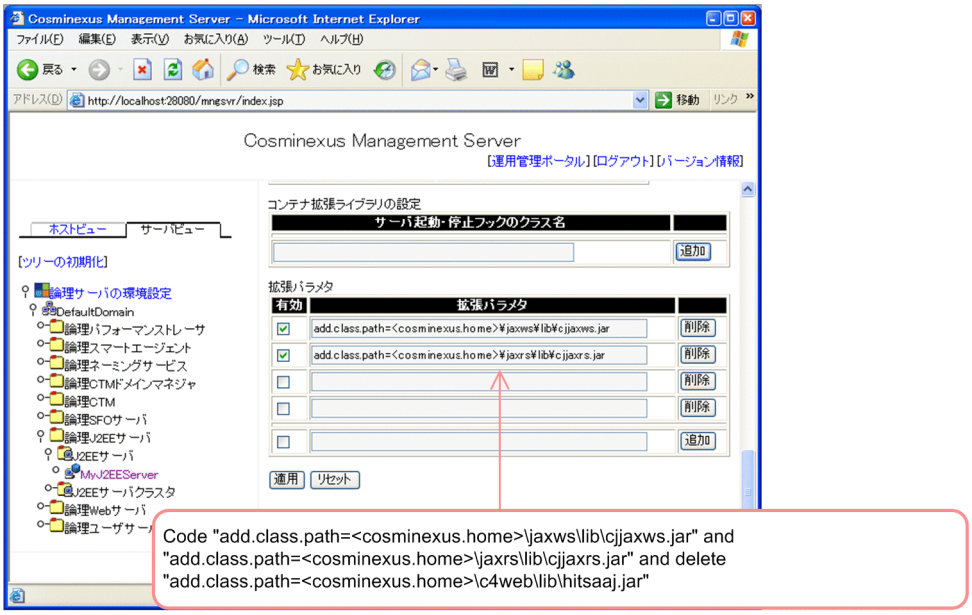
Cosminexus-installation-directory`/ CC/ server/ usrconf/ ejb/` *J2EE-server-name*`/ usrconf.cfg`

(b) When using the management portal of Management Server

When using the management portal of Management Server, you specify the settings using the extension parameters of the J2EE container settings window.

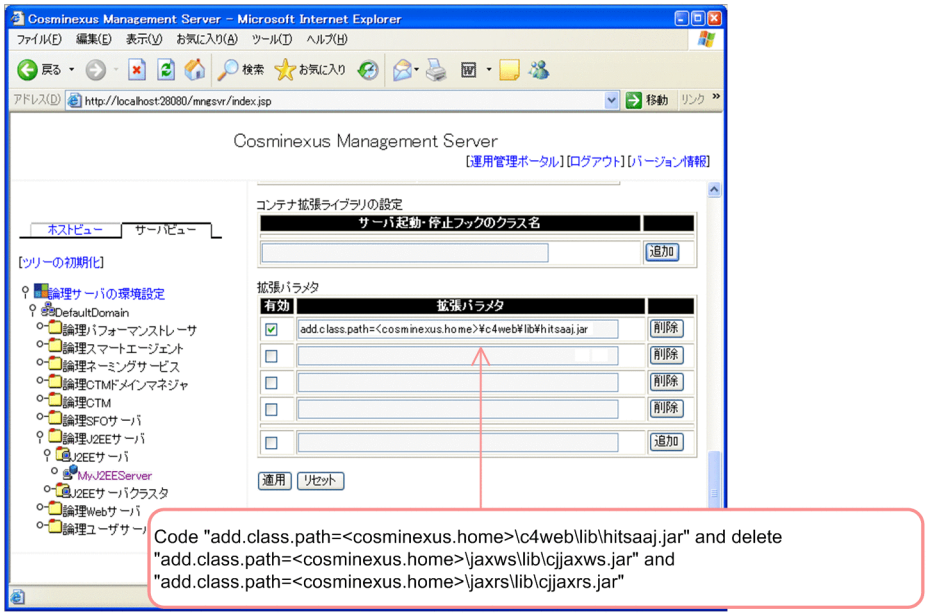
The following figure describes an example for specifying the settings with the JAX-WS and JAX-RS engines.

Figure A-1: Example of settings with the JAX-WS and JAX-RS engines when using the management portal



The following figure shows an example for specifying the settings when using the SOAP application development support function and SOAP Communication Infrastructure.

Figure A-2: Example of settings for the SOAP application development support function and SOAP Communication Infrastructure when using the management portal



For details on setting the J2EE container window of the management portal, see the section *10.9.2 J2EE container settings* in the *uCosminexus Application Server Management Portal User Guide*.

(c) When the Smart Composer functionality is used

When you use the Smart Composer functionality, the option definition file for J2EE servers is added as an extension parameter of J2EE in the Easy Setup definition file. For details on the Smart Composer functionality, see the *uCosminexus Application Server System Setup and Operation Guide*. For details on the Easy Setup definition file, see *Chapter 4 Files Used in the Smart Composer functionality* in the *uCosminexus Application Server Definition Reference Guide*.

The following example describes the settings when using the JAX-WS and JAX-RS engines:

```
<param>
 <param-name>add.class.path</ param-name>
 <param-value>&lt;cosminexus.home&gt;\jaxws\lib\cjjaxws.jar</ param-value>
</ param>
<param>
 <param-name>add.class.path</param-name>
 <param-value>&lt;cosminexus.home&gt;\jaxrs\lib\cjjaxrs.jar</param-value>
</param>
```

The following is an example of setup when the SOAP application development support function and SOAP Communication Infrastructure are used:

```
<param>
 <param-name>add.class.path</ param-name>
 <param-value>&lt;cosminexus.home&gt;\c4web\lib\hitsaaj.jar</ param-value>
</ param>
```

(4) 08-70 are upgraded to 09-00 or later

This subsection describes the notes for upgrading from 08-00 through 08-70 and installing Cosminexus 09-00 or later.

(a) Verifying host name in an SSL connection

`Hostname Verification` in an SSL connection differs in 08-00 through 08-70 and in 09-00 or later.

08-00 through 08-70

When connecting from a Web Service client to an SSL protocol-compliant Web Service, whether the host name included in the endpoint address matches with the authorized host name is not verified.

09-00 or later

When connecting from a Web Service client to an SSL protocol-compliant Web Service, whether the host name included in the endpoint address matches with the authorized host name is verified. `HostnameVerifier` to be used is a part of the default implementation of JDK. For details on the operations of default `HostnameVerifier` of JDK, see the *JDK documentation*.

Specify `true` in the `com.cosminexus.xml.ws.client.http.HostnameVerificationProperty` property to disable the verification of the host names in 09-00 or later in the same way as in 08-00 through 08-70.

(b) Mapping from a fault through an exception class

When you execute the `cjwsimport` command, a WSDL fault is mapped to a Java type according to the JAX-WS 2.2 specifications. In such cases, the conditions for creating the wrapper exception class differ in 08-00 through 08-70 and in 09-00 or later.

08-00 through 08-70

When the corresponding `wsdl:fault` element is defined for the child element of the `wsdl:binding` and `wsdl:portType` elements, the wrapper class is generated. If the `wsdl:fault` element is defined for any one of the `wsdl:binding` and `wsdl:portType` elements, the wrapper class is not generated.

09-00 or later

When the `wsdl:fault` element is defined for the child element of the `wsdl:portType` element, the wrapper class is generated. You need not define the corresponding `wsdl:fault` element as the child element of the `wsdl:binding` element.

(c) Operation of the Javax.activation.DataHandler object

The operation of the `javax.activation.DataHandler` object differs in 08-00 to 08-70 and 09-00 or later.

08-00 to 08-70

When the data received in the attachment of the `wsi: swaRef` format or MTOM/XOP specifications format is the `javax.activation.DataHandler` object, import all the data from the input stream.

09-00 or later

When the data received in the attachment of the `wsi: swaRef` format or the MTOM/XOP specification format is the the `javax.activation.DataHandler` object, sequentially import the data from the input stream.

Because the receiving process does not complete at the receiver side of the attachment in the `wsi: swaRef` format or the MTOM/XOP specification format unless all the data is imported from the input stream contained in the `javax.activation.DataHandler` object, the sending process at sender side waits until the receiving process completes at the receiving side.

To eliminate this state, you must either import all the data from the `java.io.InputStream` object contained in the `javax.activation.DataHandler` object, or export the imported stream data to the output stream by using `writeTo(java.io.OutputStream)` method of the `javax.activation.DataHandler` class.

(d) Operation of the javax.xml.transform.Source object

The operation of the `javax.xml.transform.Source` object differs in 08-00 to 08-70 and 09-00 or later.

08-00 to 08-70

When the data received in the attachment of MTOM/XOP specification format is the `javax.xml.transform.Source` object, import all the data from the input stream.

09-00 or later

When the data received in the attachment of MTOM/XOP format is the `javax.xml.transform.Source` object, sequentially import the data from the input stream. The JAXB converts the `javax.xml.transform.Source` object into `javax.xml.transform.stream.StreamSource` object.

Because the receiving process does not complete at the receiving side of attachment in the MTOM/XOP specification format unless all the data is imported from the input stream contained in the

javax.xml.transform.stream.StreamSource object, the sending process at the sender side waits until the receiving process completes at the receiving side.

To eliminate this state, you must import all the data from the java.io.Reader object contained in the javax.xml.transform.stream.StreamSource object.

(e) Differences in operation when apt command is used

When the apt command is used while combining all the following conditions, the operations differ in 08 to 08-70 and 09-00 or later.

- The document/literal style is specified.
- The Web Services Implementation Class has an implicit SEI.
- The Web Services Implementation Class inherits another Web Services Implementation Class and overrides the methods of the parent class.
- The parent Web Services Implementation Class has two or more methods and the javax.jws.WebMethod annotation is annotated in the initial public method.
- The javax.jws.WebMethod annotation is not annotated in the second and subsequent public methods in the parent Web Services Implementation Class.
- The javax.jws.WebMethod (exclude=true) annotation is annotated by overriding the initial public methods in the child Web Services Implementation Class.
- The second and subsequent public methods are not overridden in the parent Web Services Implementation Class of the child Web Services Implementation Class.

08-00 to 08-70

The error message is output in the standard error output and log, and the process ends (KDJW61093-E).

09-00 or later

The process successfully ends.

The examples of implementation are as follows.

Parent Web Services Implementation Class

```
@WebService
@SOAPBinding(style=SOAPBinding.Style.DOCUMENT, use=SOAPBinding.Use.LITERAL,
 parameterStyle=SOAPBinding.ParameterStyle.WRAPPED)
public class AddNumbersParentImpl {
    ...
    @WebMethod
    public int addNumbers1(int number1, int number2) {
 ...
    }
    public int addNumbers2(int number1, int number2) {
 ...
    }
}
```

Child Web Services Implementation Class

```
@WebService
@SOAPBinding(style=SOAPBinding.Style.DOCUMENT, use=SOAPBinding.Use.LITERAL,
 parameterStyle=SOAPBinding.ParameterStyle.WRAPPED)
public class AddNumbersChildImpl extends AddNumbersParentImpl {
 ...
    @WebMethod(exclude=true)
    public int addNumbers1(int number1, int number2) {
 ...
    }
}
```

(f) Differences in run time operation at service side when the addressing functionality is used

When the addressing functionality is used and when all the following conditions are combined, the run time operations at the service side differ in 08-50 to 08-70 and 09-00 or later.

- The messages are in the SOAP 1.1 format.
- The `wsa: ReplyTo` element does not exist in the addressing header of the request message.
- The `wsa: FaultTo` element is present in the addressing header of the request message, and the URI of the Web Service is set.
- If an error occurs on the service side, and the service side returns a SOAP fault.

08-50 to 08-70

Returns HTTP Status Code: 202

09-00 or later

Returns HTTP Status Code: 500.

A.2 Compatibility of WSDL created in an earlier version

WSDL created in an earlier version can be used on the JAX-WS engine. However, there is a limit to the XML Schema coded in WSDL. This subsection describes the limitations on the data typeconstraint facet, and occurrence count.

The following table describes the data type limitations:

Table A－1: XML Schema data type limitations

No.	Data type			Maximum value	Maximum digits
1	`duration` type (format: P*nY*n*M*n*D*T*n*H*n*M*n*S*)	*nY* (year)		2147483647	--
2		*nM* (month)			
3		*nD* (day)			
4		*nH* (hour)			
5		*nM* (minute)			
6		*nS* (second)	1 second or more		
7			Less than 1 second		
8	`date` type (format: *CCYY-MM-DD*)	*CCYY*[#1]			
9	`gYearMonth` type (format: *CCYY-MM*)	*CCYY*[#1]			
10	`gYear` type (format: *CCYY*)	*CCYY*[#1]			
11	`dateTime` type (format: *CCYY-MM-DDThh:mm:ss*)	*CCYY*[#1]			
		Less than 1 second[#2]			
12	`dateTime` type (format: *CCYY-MM-DDThh:mm:ss*)	Less than 1 second[#2]			
13	`time` type (format: *hh:mm:ss*)			Not defined.	
14	`base64Binary` type				Depends on the memory.
15	`hexBinary` type				
16	`decimal` type				
17	`integer` type				
18	`nonPositiveInteger` type				
19	`nonNegativeInteger` type				

No.	Data type	Maximum value	Maximum digits
20	`negativeInteger` type	Not defined.	Depends on the memory.
21	`positiveInteger` type		

Legend:
--: Not applicable

#1
If exceeding "9999", you can add the digits.

#2
Subsequent to "ss", after the decimal point, you can specify a value that is less than 1 second.

The following table describes the limitations related to the constraint facet:

Table A-2: Limitations related to the XML Schema constraint facet

No.	Constraint facet	Maximum value	Maximum digits
1	`length`	2147483647	--
2	`minLength`		
3	`maxLength`		
4	`totalDigits`		
5	`fractionDigits`		
6	`maxInclusive`	Not defined.	Depends on the memory.
7	`maxExclusive`		
8	`minInclusive`		
9	`minExclusive`		

Legend:
--: Not applicable

The following table describes the limitations on the specification of the occurrence count:

Table A-3: Limitations related to the specification of the XML Schema occurrence count

No.	Specification of the occurrence count	Maximum value	Maximum digits
1	`minOccurs`	2147483647	--
2	`maxOccurs`		

Legend:
--: Not applicable

B. Migrating from the POJO Web Service to the EJB Web Service

You can use the EJB functionality by operating the Web Service developed in POJO as an EJB Web Service. This appendix describes how to migrate the POJO Web Service to the EJB Web Service. Note that when the POJO Web Service is adequate, migration to EJB is not required.

The following table describes how to migrate the POJO Web Service to the EJB Web Service:

Table B-1: How to migrate to the EJB Web Service

No.	Components of the POJO Web Service	How to migrate to the EJB Web Service
1	Web Service Implementation Class	Add `javax.ejb.Stateless` annotation in the source code and compile.
2	Java class other than No.1 (SEI, JavaBeans class (Stub))	You can use the class file as is even after migrating to the EJB Web Service.
3	WSDL	The URL that is published as a Web Service differs in the POJO Web Service and the EJB Web Service. Therefore, change the URL related values for the location attributes of `soap:address` element, and then use. For the URL of EJB, see *10.2.2(1) Discovery*.
4	DD such as cosminexus-jaxws.xml, web.xml, and application.xml	You cannot use the DD created for the POJO Web Service as is in the EJB Web Service. Create a new DD for the EJB Web Service.

While migrating the POJO Web Service to the EJB Web Service, you must modify the source code of the Web Service Implementation Class. The description in this section assumes that the class files required for the execution of the POJO Web Service and the source code required for the Web Service Implementation Class are already available.

The following is the procedure for migration:

(1) Creating the EJB Web Service Implementation Class

Annotate in the source code of a POJO Web Service Implementation Class by using the `javax.ejb.Stateless` annotation. Compile the annotated source code and create the class file.

(2) Diverting the class files required for executing the Web Service and the WSDL

You can also use the class files of SEI and JavaBean class (stub) required for executing the POJO Web Service when executing as an EJB Web Service.

If WSDL used for executing the POJO Web Service exists, these files can be used for executing the EJB Web Service as well. However, because the URL published as a Web Service differs when executing the POJO Web Service as an EJB Web Service, change and then use the URL related values for the location attributes of soap:address element of WSDL. For the URL of the EJB Web Service, see *10.2.2(1) Discovery*.

(3) Creating a EJB JAR file

Store the class files and WSDL of (1) (2) in the EJB JAR file. For the configuration of the EJB JAR file, see *3.5.2 Configuring EJB JAR files*.

(4) Creating a Deployment descriptor

Create `application.xml` that specifies the EJB JAR file created in (3). The following is an example of `application.xml`. Specify the name of the EJB JAR file created in (3) in `statelessjava.jar`.

```
<?xml version="1.0" encoding="UTF-8"?>
<application version="6" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/
javaee/application_6.xsd">

  <description>Sample application &amp;quot;statelessjava_dynamic_generate&amp;quot;</
description>
  <display-name>Sample_application_statelessjava_dynamic_generate </display-name>
  <module>
    <ejb>statelessjava_dynamic_generate.jar</ejb>
  </module>
</application>
```

For creating `application.xml` of version 5, set up the `version` attribute of the `application` element to `5` and the second location information of the `xsd:schemaLocation` attribute to `http://java.sun.com/xml/ns/javaee/application_5.xsd`.

In EJB Web Service, create `web.xml` while adding settings of the Servlet filter to `web.xml`. For details, see *3.5.4 Creating WAR file for settings of EJB Web Service*. Also, `cosminexus-jaxws.xml` does not apply to the EJB Web Service.

(5) Creating a EAR file

Create an EAR file containing the EJB JAR files created in (3) (4) and `application.xml`. For EAR file configuration , see *3.5.3 Creating EAR files*.

C. Calculating the Memory Usage for JAX-WS Engine

This appendix describes the memory usage for the JAX-WS engine in the Java heap area (machine on which Web Service is executed) using the following cases:

- Memory usage when the application starts
- Memory usage per request
- Memory usage per request when attachment is used
- Memory usage per unit time

Note that the default value is assumed for all the settings related to the JAX-WS engine. The memory usage explained hereafter includes the memory usage in the internal classes of the JAX-WS engine, the memory usage for Java API invoked by the internal classes and the amount of memory used in the internal implementation classes of JavaVM.

Furthermore, the JAX-WS engine-specific memory usage explained hereafter is a rough guideline. The memory usage in the actual system might differ slightly.

C.1 Memory usage when the application starts

This subsection describes the memory usage when the application including the Web Service starts. Application startup indicates when the `cjstartapp` command is executed or directly after the J2EE application is started from the Management Server.

When the application starts, the operations such as initialization of classes required for executing the JAX-WS engine are performed. The amount of memory required for initialization is about 21.2 MB regardless of the processing of the Web Service Implementation Class.

C.2 Memory usage per request

This subsection describes the memory required for processing each request. One request indicates the operations from the receipt of the request from the Web Service client, processing of the request, until the returnof a response.

In the first request and from the second request onwards, the JAX-WS engine-specific memory used is different.

The following indicates memory usage during the first request:

```
memory-usage-per-request(first) =
memory-used-by-the-Web-Service-Implementation-Class + JAX-WS-engine-specific-memory-
usage(2.43 MB)
```

The following indicates memory usage from the second request onwards:

```
memory-usage-per-request(second-time-onwards) =
memory-used-by-the-Web-Service-Implementation-Class + JAX-WS-engine-specific-memory-
usage(119.9 KB)
```

Also, the formula for calculating the memory usage when multiple requests arrive per unit time is as follows:

```
memory-usage-per-unit-time =
memory-usage-per-request(first) + {memory-usage-per-request(second-time-onwards)
(number-of-requests-processed-per-unit-time - 1)}
```

C.3 Memory usage per request when attachments are used

This subsection describes the amount of memory required for processing each request in a Web Service application using attachments. One request indicates the operations from the receipt of the request from the Web Service client, processing of the request, until the return of a response.

The calculation method described in this subsection, assumes a Web Service that receives an attachment from the Web Service client and returns the attachment as is to the Web Service client.

(1) When the attachment size is 10 KB

The following indicates memory usage during the first request:

```
memory-usage-per-request(first) =
memory-used-by-the-Web-Service-Implementation-Class + JAX-WS-engine-specific-memory-
usage(2.68 MB)
```

The following indicates memory usage from the second request onwards:

```
memory-usage-per-request(second-time-onwards) =
memory-used-by-the-Web-Service-Implementation-Class + JAX-WS-engine-specific-memory-
usage(193 KB)
```

(2) When the attachment size is 100 KB

The following indicates memory usage during the first request:

```
memory-usage-per-request(first) =
memory-used-by-the-Web-Service-Implementation-Class + JAX-WS-engine-specific-memory-
usage (2.73 MB)
```

The following indicates memory usage from the second request onwards:

```
memory-usage-per-request(second-time-onwards) =
memory-used-by-the-Web-Service-Implementation-Class + JAX-WS-engine-specific-memory-
usage (411 KB)
```

(3) When the attachment size is 1 MB

The following indicates memory usage during the first request:

```
memory-usage-per-request(first) =
memory-used-by-the-Web-Service-Implementation-Class + JAX-WS-engine-specific-memory-
usage (4.54 MB)
```

The following indicates memory usage from the second request onwards:

```
memory-usage-per-request(second-time-onwards) =
memory-used-by-the-Web-Service-Implementation-Class + JAX-WS-engine-specific-memory-
usage (2.24 MB)
```

C.4 Calculating the memory usage per unit time

The formula for calculating the memory usage when multiple requests arrive per unit time is as follows:

```
memory-usage-per-unit-time =
memory-usage-per-request(first) + {memory-usage-per-request(second-time-onwards)
(number-of-requests-processed-per-unit-time - 1)}
```

In this example, the memory usage per hour is calculated for a system in which 60 requests arrive in one minute and attachments of about 10 KB are handled on an average. In this system, the memory used by the Web Service Implementation Class is assumed to be 100 KB per request.

In this case, the memory usage per request for the first time is as follows:

```
100KB + 2.68MB = 2.78MB
```

Also, the memory usage per request from the second time onwards is as follows:

```
100KB + 193KB = 293KB
```

From these results, you can calculate the memory usage per hour as follows:

```
2.78MB + {293KB x (60 x 60 - 1)} = 1,033MB
```

If you add the amount of memory for application startup to the memory usage per hour, you understand the amount of memory of Java heap used in one hour after this application starts.

```
21.2MB + 1, 033MB = 1,054.2MB
```

D. Glossary

Terminology used in this manual

See the manual *uCosminexus Application Server and BPM/ESB Platform Terminology Guide*.

Index

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W

X