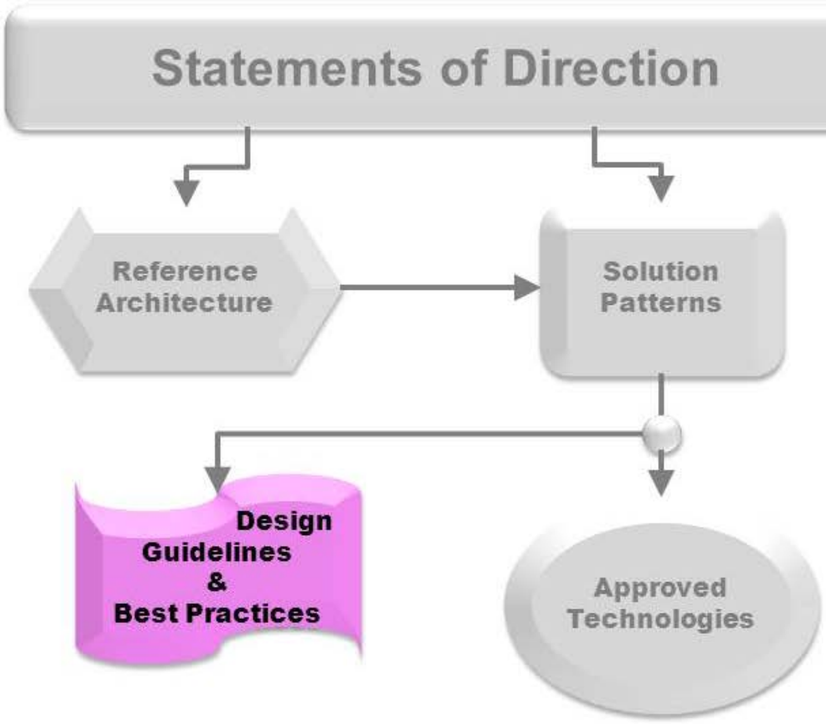
## Design Guidelines and Best Practices

# DFS SOAP Web Services Standard & Development Guide (Using EJB3.1 & JAX-WS)

**Revision 3.0**

**01/30/2013**



## Table of Contents

## List of Figures

## List of Tables

# 1.0 Introduction

At the time this document was created, the reusable services in DFS have either been implemented as RMI or SOAP Web Services. RMI is a tightly coupled protocol and is not a suitable implementation for reusable services. Discover intends to discontinue RMI as a reusable service. SOAP Web Service is a better implementation for services. It is loosely coupled, language independent, interoperable, and supports clients' timeouts. However, there are various implementations of SOAP Web Service. Using WAS 6.x, Discover implemented services using JAX-RPC. With WAS 8, Discover is phasing out JAX-RPC implementations, using the new standard of JAX-WS.

SOAP Web Service has many varieties of implementation styles. This document defines a standard SOAP Web Service implementation for DFS distributed services. It also provides best practices for Web Service development so that the services are implemented in a consistent method across the enterprise to achieve the best interoperability with external partners. This document also provides instructions for service developers to create new Web Services and for migrating existing services to the new DFS standard targeting the IBM WebSphere8.0 application server.

For directions on the usage of Web Services within DFS, please refer to the **SOAP Web Services Statement of Direction**

# 2.0 Background

Web services are not new. First there was SOAP. However, SOAP only described what the messages looked like. Then there was WSDL which describes the Web Service. But WSDL did not instruct the user/developer how to write web services in Java. Then came JAX-RPC 1.0 and its revision, JAX-RPC1.1. The Web Services deployed on WAS6.1 in DFS are all JAX-RPC1.1 services.

Later revisions to JAX-RPC1.1 were named JAX-WS to better describe the nature of web services. Thus the successor to JAX-RPC 1.1 is JAX-WS 2.0 - the Java API for XML-based web services.

For services deployed on WAS8.0, the JAX-WS2.2 version will be used for DFS. The existing services on WAS6.1 will be migrated to JAX-WS2.2. To help developers understand the major differences between JAX-RPC1.1 and JAX-WS2.2, this chapter provides a high level comparison between the two specs. The following table lists the key differences:

**Table 4-1: Comparisons Between Web Services and Java Applications**

| | **JAX-RPC1.1(WAS6.1)** | **JAX-WS2.2(WAS8.0)** |
|---|---|---|
| SOAP Version | SOAP 1.1 | SOAP 1.1, SOAP 1.2 |
| WSDL Version | WSDL1.1 | WSDL1.1, WSDL1.2 |
| WS-I Basic Profile | WS-I Basic Profile 1.0 | WS-I Basic Profile 1.0, 1.1, 1.2, 2.0 |
| Data Mapping | Java and XML data mapping | JAXB2.2 |
| Interface mapping model | Synchronous | Synchronous and Asynchronous |
| Binary Data Transmission | Attachment or Base64 encoding | MTOM1.0 |
| Programming Model | Web Service Interface | Web Service Annotations |

Referring to the table above, JAX-WS still supports SOAP1.1 and WSDL1.1 so interoperability with JAX-RPC will not be affected. JAX-WS also provides a number of benefits over JAX-RPC such as simplified programming and use of annotation. Note that JAX-WS uses JAXB to marshal and unmarshal between Java objects and XML. As a result, some Java types supported by JAX-RPC may not be supported by JAX-WS, such as ***array*** and ***HashMap*** (supported in JAX-RPC). Section 4.1 of this document provides directions on how to deal with these data types.

# 3.0 DFS SOAP Web Service Standards

Web Services can be implemented in many ways. There are various styles and specifications. It is important for an organization to standardize these choices so these services can be implemented in a consistent way for best inter-operability and easy management. This section defines a list of standard items for Web Service implementation.

## 3.1. Enterprise Service Router (ESR)

An ESR will be used to mediate between consumers and the actual service implementation to add following benefits:

- Decouples consumers from service providers by virtualizing service interfaces
- Allow service versioning by transforming old message structures to new format with no impact to consumers
- Provides message based routing and failover control for increased availability
- Enforces service level management by monitoring throughput and response times
- Enables common approach for error handling and alerting
- Provides security enforcement point and removes most security management from application teams
- Provides common spot to capture service metrics
- Provides protocol transformation e.g. transform HTTP to JMS
- Triggers events from the message when necessary
- Provides debug capability for service developers

## 3.2. DFS SOAP Web Service Standards

### 3.2.1. Protocol: SOAP1.1 over HTTP/HTTPS

While SOAP over HTTP/HTTPS is the DFS Web Service Standard, SOAP over JMS/MQ can be an option if services have special requirements. However, the AD teams need to work with their solution architect and get ARB approval if they plan to use SOAP over JMS/MQ.

### 3.2.2. Java Web Services Implementation Spec

JAX-WS2.2

### 3.2.3. Web Service implementation: EJB3.1

There are two implementations when creating a Web Service: stateless session EJB and regular Java Bean. The stateless session EJB3.1 implementation is preferred for the following reasons:

1. The EJB container provides services such as container managed transactions and instance pooling.

2. The majority of current DFS middle tier applications and services are implemented in EJBs. The EJB implementation approach is consistent with our current application implementation.
3. Using an EJB provides the ability to have a dual interface for RMI and Web Service consumers.

### 3.2.4. Style: Document/Literal

A WSDL document describes a Web service. A WSDL binding describes how the service is bound to a messaging protocol, particularly the SOAP messaging protocol. A WSDL SOAP binding can be either a Remote Procedure Call (RPC) style binding or a document style binding. A SOAP binding can also have an encoded use or a literal use. This gives us four style/use models:

1. RPC/encoded
2. RPC/literal
3. Document/encoded
4. Document/literal

Document/literal is chosen as the DFS standard for the following reasons:

1. document/literal is WS-I compliant and interoperable
2. the SOAP message can be validated by any XML validation tools
3. it is the most widely used style for Web Services

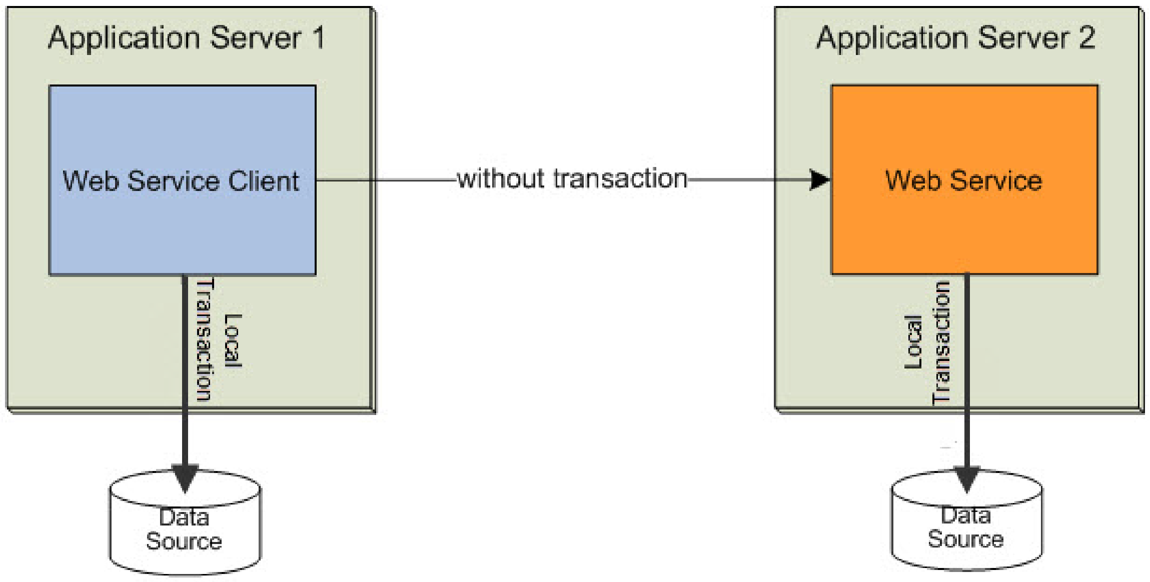
### 3.2.5. Transaction: EJB container managed local transaction

DFS has limited support for two-phase commit. The majority of DFS RMI applications/services use EJB container managed transactions. EJB container managed transaction is also inline with WS-AT (Web Service Atomic Transaction) specification.

The following are the standard for setting EJB container managed transactions:

1. A Web Service should not expect the consumers to initiate a transaction when invoking the service. On the other side, the service consumer should not invoke a Web Service within a transaction. If requirements require a service be invoked as part of consumer transaction, contact Enterprise Architecture.
2. Use EJB container managed transactions. Do not use Bean managed transactions. In EJB3.1 these transaction attributes are annotated within EJB code instead of deployment descript configurations. Refer to “WAS8 Migration Guide” for more details.
3. Normally the transaction attribute for the EJB should be set to either “Not Supported” or “Requires New”. If you need to set the transaction to other values, contact Enterprise Architecture.

**Figure 5-1: DFS Web Service Transaction Standard**



### 3.2.6. Web Service Timeouts

The developer must set the WS proxy timeout for all WS calls. The actual timeout value should be determined based on the service SLA and the project requirements. Refer to the "**Timeout Guidelines for Reusable Web Services**" for better understanding of different layers of timeouts for Web Services.

The SIF provides the capability that allows Web Service consumers to set timeouts at method level via SIF configuration file. Refer to the SIF ICD for the details.

### 3.2.7. ESR Enforcement

All calls to reusable Web Services must go through an ESR. Please refer to "ECCs DataPower Reusable Template" in Repository Manager for details on how to create DataPower artifacts using templates.

### 3.2.8. Message Compression

Message compression does not always improve performance. Message compression will save network traffic time, but it takes additional CPU time for compressing and decompressing the message. Tests within the DFS LAN environment show there are no noticeable performance differences between these two implementations for different payload sizes (1K to 100K). If the service needs to send large payloads over the WAN, contact Enterprise Architecture for available options using message compression.

### 3.2.9. Binary Data transmission

MTOM should be used for binary data (e.g. images, PDF) transmission. Refer to "**Rational Application Developer for WebSphere Software V8 Programming Guide**" section 14.15 for procedures to implement MTOM.

### 3.2.10. WS-I Compliant

Web Services created in DFS must be compliant with the WS-I specification. WS-I compliant ensures the DFS services follow the industry's conformance standards with regards to interoperability. WS-I BP 1.2 is the DFS standard compliance level. If a specific service requires a different compliance level in order to be integrated with external applications, contact Enterprise Architecture.

RAD can be configured to enforce the development of WS-I compliant Web Services. Refer to Appendix A for setting WS-I compliance level in RAD.

### 3.2.11. Web Service Exception

Each Web Service operation must throw an application Exception. As services board the Enterprise Service Router (ESR), the expectation is that error messages are standardized in a manner that allows the ESR to manage errors in a common way. As part of the standard error format, one attribute is specified as the "Exception Type". This error code is examined by the ESR and used to make intelligent decisions for retry and failover handling. It is the responsibility of the physical service to return these errors as part of the SOAP Exception.

The standard DFSExceptionInterface has been created and packaged into DFSCommon.jar. Developers should include this library into their service projects. For details on how to create web service exceptions that conform to the DFS web service standard refer to Appendix D and the DFSCommon ICD.

Security requirements prohibit revealing detailed exception information, such as exception stack trace, to the external consumers. External consumers should only get generic exception messages. The ESR in zone 1 should be configured to filter out any detailed Exception information (including application exception, sever exception and ESR exception) and replace them with generic exception messages. All exceptions should be logged with a priority level that indicates the potential seriousness of the condition ESR detects.

### 3.2.12. Consumer Identification (ConsumerID)

The consumer ID is a DFS standard soap header element for web service consumers. It is used to identify the consumer application. This ID is required to be logged on DataPower along with the errors for troubleshooting, impact analysis and tracking service usage by consumers. This ID also has other intended use in future such as assisting enforcement of consumer specific SLAs, staging specific consumers to a new service version, etc. The ConsumerID element consists of the following attributes

> **xmlns:** xml namespace where the ConsumerID element is defined
> **appGroup:** Application group in Troux Repository
> **appShortName:** Application short name in Troux Repository
> **appVersion:** Application version number

The SIF has been enhanced to set this ID in the SOAP header. DFS internal WS consumers should set this ID in the SIF configuration file. For detailed information on how to use SIF, please refer to the SIF ICD document in the Reuse Repository. The following is an example of the SOAP header that contains the consumer ID:

```
<soapenv:Header>
        <ConsumerID appGroup="applicationGroup"
        appShortName="applicationShortName"
        appVersion="1.0.0" userID="jdoe"/>
</soapenv:Header>
```

For naming consistency, please only use the names found in this **list** for "applicationGroup" and "applicationShortName". If your application cannot be found from the above list, contact Enterprise Architecture.

# 4.0 Web Services Implementations for Service Providers

This section provides developers with guide on JAX-WS SOAP Web Services best practices and coding requirements as well as how to create JAX-WS Web Services; how to migrate existing RMI and JAX-RPC into JAX-WS Web Services. Detailed step by step guide with screen shoots are provided in the Appendices.

Before initiating JAX-WS services development, set the specific RAD preferences so the services created meet DFS standards. Appendix A provides screen shoots on how to set these preferences.

## 4.1. Best Practices and Coding Requirements

### 4.1.1. Best Practices:

- Service operations should be designed as coarse-grained and business oriented whenever possible
- Simplicity is gold. Keep exposed operations, including the parameters and exceptions, as simple as possible. Minimize object inheritance and object nesting (objects containing other objects)

### 4.1.2. Items to Avoid:

- Do not use array (e.g. Customer[ ]). Use ***`java.util.List`*** if a collection of data needs to be transferred.
- Do not use HashMap. Use ***`java.util.List`*** if a collection of data needs to be transferred. If HashMap must be used, contained it within another object. Even using this approach, the client side proxy generated for java consumers will use a custom HashMap class rather than the standard ***`java.util.HashMap`***. It is recommended consumers create a ***`java.util.HashMap`*** and copy the contents from the custom HashMap into the standard HashMap. This will ensure consistent and understood behavior.
- Do not use method overloading (same method name with different parameters).
- Do not create business or utility methods within VOs (Value Objects). VOs should have getters and setters of their attributes as the only methods.
- Do not use interfaces or abstract classes as input or output parameters.
- Input and output parameters, including objects that are referenced by these parameters, must follow the Java Bean coding standard. (i.e. must have a public default constructor, must implement Serializable interface, and attributes must have public getters and setters.

## 4.2. Create a Bottom-up JAX-WS Web Service

Web Service can be created in one of the two approaches: Bottom up or Top down. The Bottom up approach is to create Java artifacts first then expose them as Web Service interfaces. It is typically an easier way to build a Web Service as developers just need to create the EJB and then annotate the EJB with Web Service annotations.

Detailed guide of EJB3.1 coding along with projects creation is out of the scope of this document. Please refer to "WAS8 Migration Guide" if you want to know more about EJB3.1. This document assumes that developers understand how to develop an EBJ3.1 stateless session bean with annotations. For detailed steps on how to create a new bottom-up JAX-WS Web Service using EJB3.1, refer to Appendix B.

### 4.3. Create a Top-down JAX-WS Web Service

The top-down approach is to create a WSDL first, then generate Java artifacts. For this approach, you use RAD WSDL editor to create a WSDL. Then use the Web Service wizard to create the web service and the skeleton Java classes.

Since creating a WSDL is typically more difficult than to create an EJB, normally you want to choose this approach when you have a standard service definition and want to implement this definition to provide the requested service. Refer to Appendix C for detailed steps on how to create a top-down JAX-WS Web Service.

### 4.4. Convert an Existing EJB2.1 into JAX-WS Web Service

Because the Java artifacts differ so much between EJB2.1 and EJB3.1, migration of the project workspace is not recommended. The recommend approach is to create a new workspace and new EJB3.1 projects within the new workspace. There are two steps involved to convert EJB2.1 to JAX-WS:

- Convert EJB2.1 into EJB3.1 on WAS8. Refer to "WAS8 Migration Guide" to complete this task.
- Create JAX-WS from the EJB3.1. Follow Appendix B; start from step 5 to create a JAX-WS web service.

### 4.5. Convert an Existing JAX-RPC into JAX-WS Web Service

If the JAX-RPC service was created using bottom-up approach using EJB2.1, then the conversion steps are the same as those specified in section 4.4. Convert EJB2.1 into EJB3.1 and then create JAX-WS from EJB3.1.

If the JAX-RPC service was created using top-down approach, then the conversion process will be similar to that of creating a new top-down JAX-WS service described in Appendix C. Note that in step 4, the old JAX-RPC WSDL must be imported instead of creating a new WSDL. After the JAX-RPC WSDL is imported, update it so that it conforms to JAX-WS.

### 4.6. Create a Client Proxy

Service provider will also need to create a Client Proxy if the Web Service is going to be consumed by DFS internal WebSphere applications. In this case the ECC SIF should be incorporated to create the client proxy. Refer to Appendix E for details of creating the Client Proxy using SIF. In order to keep naming consistence for client proxies, it is recommended to follow the "DFS Service Versioning Standard" for the naming of the client proxy

After creation, the client proxy component should be checked into ClearCase for other teams to use.

If the Web Service is going to be consumed by external vender applications, the WSDL should be provided to the external consumers. In this case the client proxy is not needed. Appendix F shows how to get WSDL if you are using bottom-up approach.

# 5.0 Web Service Consumer Implementations

This section is a guide for developers who want to consume a JAX-WS SOAP Web Service.

## 5.1. Consuming Internal Web Services

If consuming an internal WebSphere based service, then a client proxy jar will be provided by the Web Service provider. In this case the consumers can import this jar into their project workspaces and use it to invoke the Web Service. The client proxy jar contains an SIF configuration file. Developers need to configure the Web Service “url”, operation “timeout” and “ConsumerID”. Refer to the SIF ICD document for how to configure these attributes in the SIF configuration file.

## 5.2. Consuming External Web Services

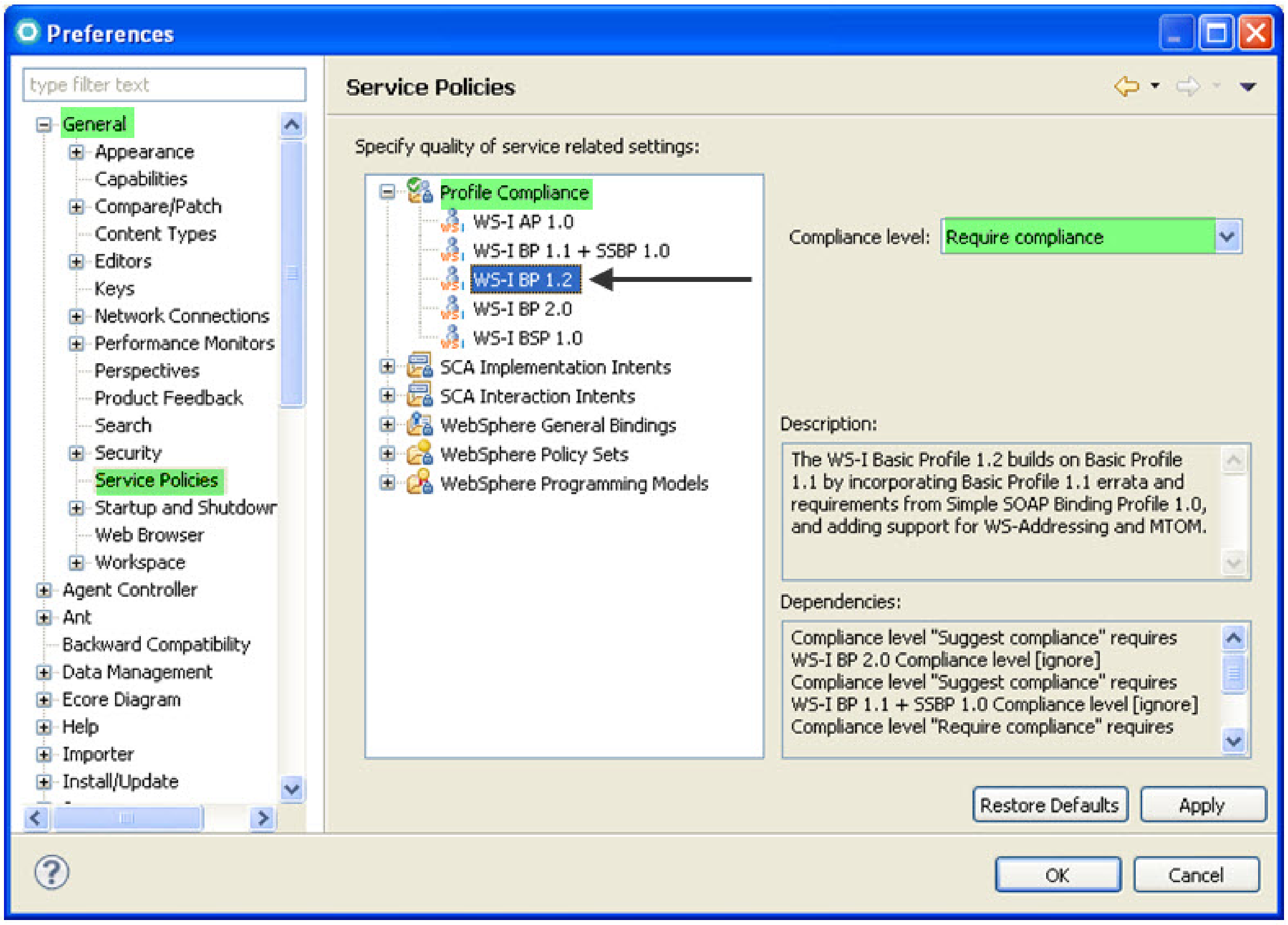
DFS applications consume external Web Services, a WSDL will normally be provided by the Web Services provider. The consumers need to generate Web Services client proxy from the WSDL.

## 6.0 Glossary

**Note: Also refer to the DFS Glossary**

# Appendix A: Set RAD Preferences for JAX-WS Development

- Set WS-I compliance level in RAD.
    1. Select Window→Preferences→ General → Service Policies → Profile Compliance
    2. Select WS-I BP 1.2
    3. For Compliance level choose Require compliance



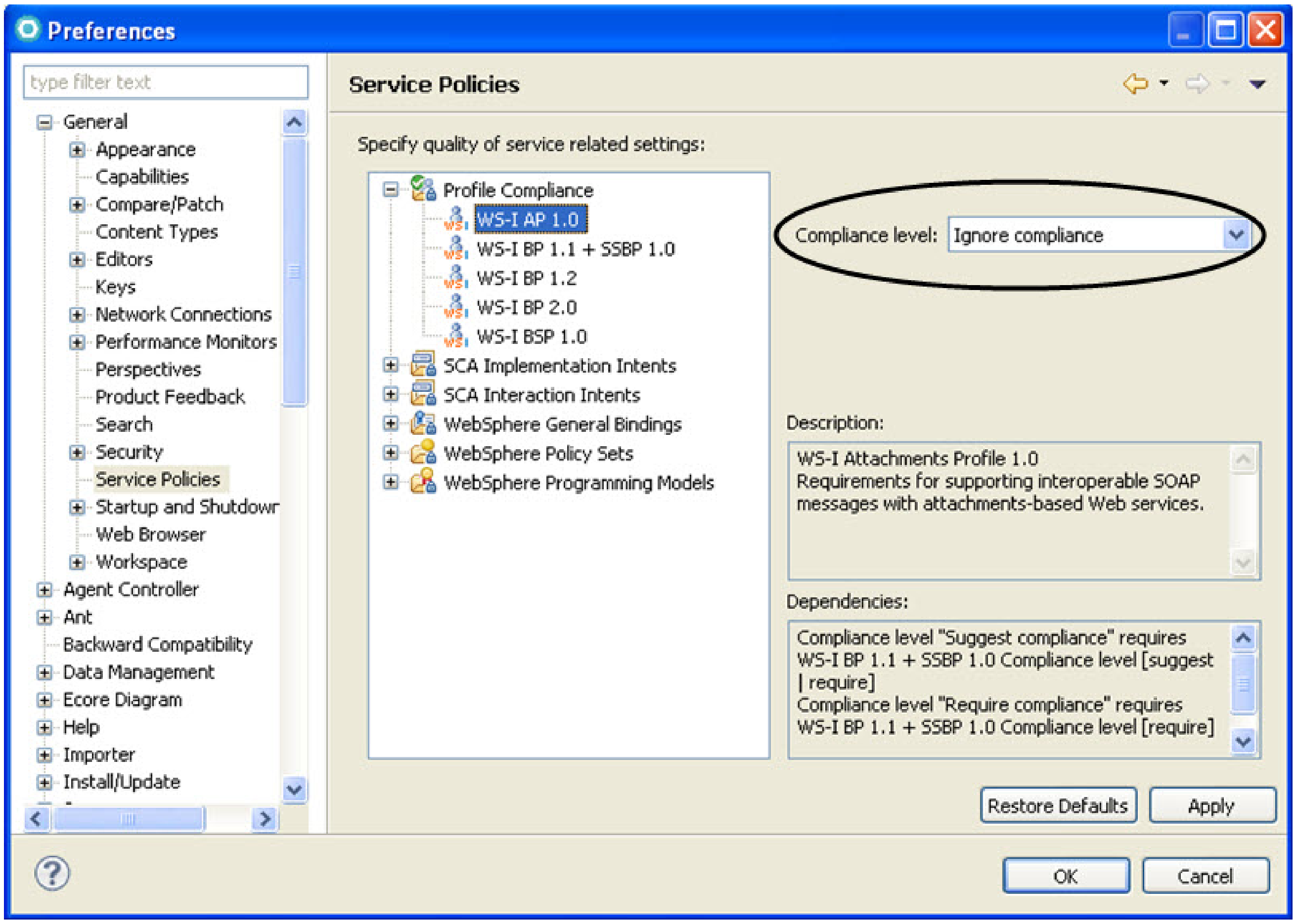
Specify quality of service related settings:

Compliance level: Require compliance

Description:
The WS-I Basic Profile 1.2 builds on Basic Profile 1.1 by incorporating Basic Profile 1.1 errata and requirements from Simple SOAP Binding Profile 1.0, and adding support for WS-Addressing and MTOM.

Dependencies:
Compliance level "Suggest compliance" requires WS-I BP 2.0 Compliance level [ignore]
Compliance level "Suggest compliance" requires WS-I BP 1.1 + SSBP 1.0 Compliance level [ignore]
Compliance level "Require compliance" requires

4. Set all other Profile Compliance to **Ignore compliance**. The screen shot below is for **WS-I AP 1.0**.
5. Repeat step 4, above, for **WS-I BP1.1 + SSBP1.0**, **WS-I BP 2.0** and **WS-I BSP 1.0**.
6. Click **Apply**



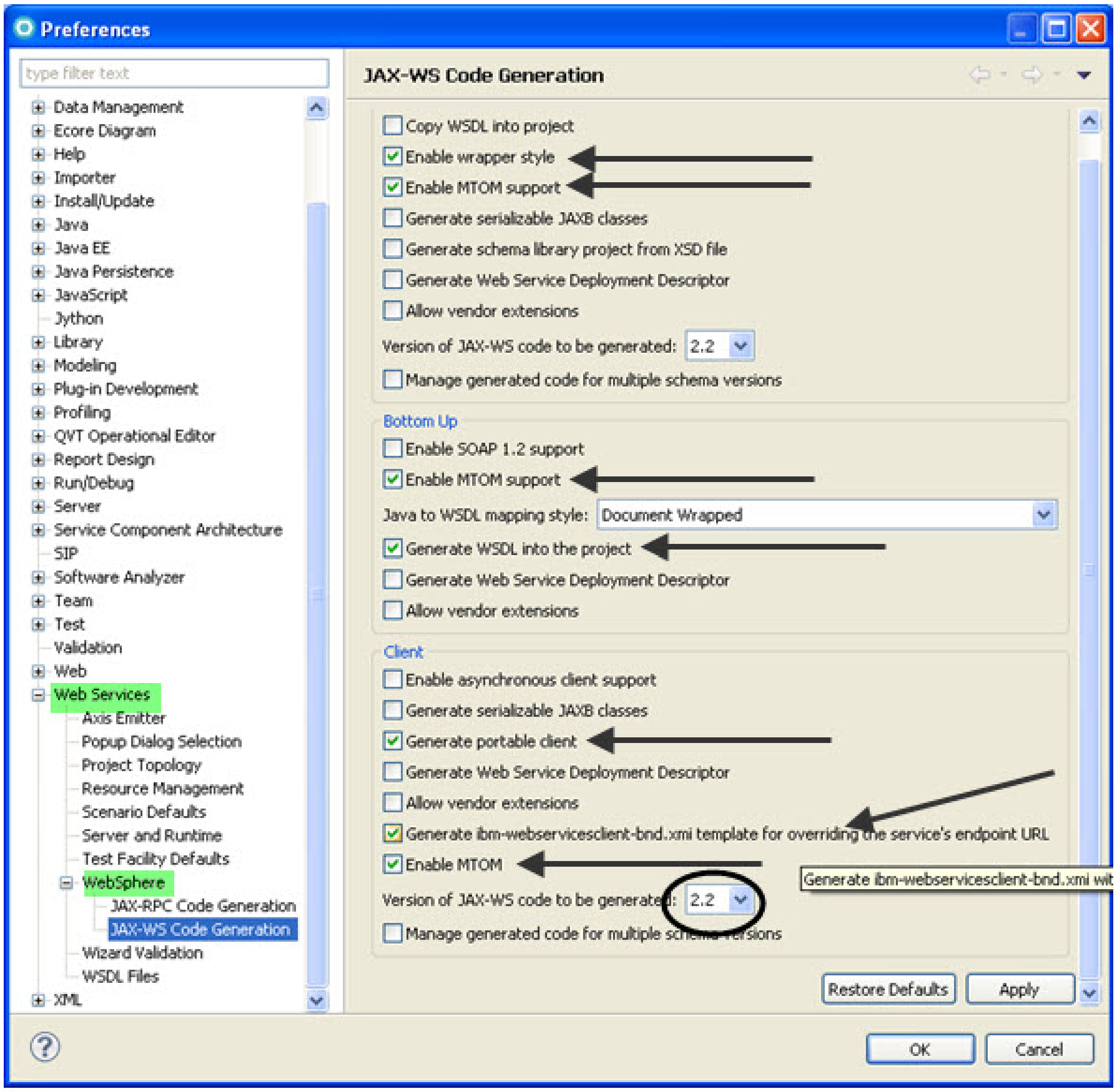
Specify quality of service related settings:

Compliance level: Ignore compliance

Description:
WS-I Attachments Profile 1.0
Requirements for supporting interoperable SOAP messages with attachments-based Web services.

Dependencies:
Compliance level "Suggest compliance" requires WS-I BP 1.1 + SSBP 1.0 Compliance level [suggest | require]
Compliance level "Require compliance" requires WS-I BP 1.1 + SSBP 1.0 Compliance level [require]

- Set Code Generation preference.
    1. Select Window→Preferences→ Web Services → WebSphere → JAX-WS Code Generation
    2. Select the items that are highlighted.



JAX-WS Code Generation

- [ ] Copy WSDL into project
- [✓] Enable wrapper style
- [✓] Enable MTOM support
- [ ] Generate serializable JAXB classes
- [ ] Generate schema library project from XSD file
- [ ] Generate Web Service Deployment Descriptor
- [ ] Allow vendor extensions

Version of JAX-WS code to be generated: 2.2

- [ ] Manage generated code for multiple schema versions

Bottom Up

- [ ] Enable SOAP 1.2 support
- [✓] Enable MTOM support

Java to WSDL mapping style: Document Wrapped

- [✓] Generate WSDL into the project
- [ ] Generate Web Service Deployment Descriptor
- [ ] Allow vendor extensions

Client

- [ ] Enable asynchronous client support
- [ ] Generate serializable JAXB classes
- [✓] Generate portable client
- [ ] Generate Web Service Deployment Descriptor
- [ ] Allow vendor extensions
- [✓] Generate ibm-webservicesclient-bnd.xmi template for overriding the service's endpoint URL
- [✓] Enable MTOM

Version of JAX-WS code to be generated: 2.2

- [ ] Manage generated code for multiple schema versions

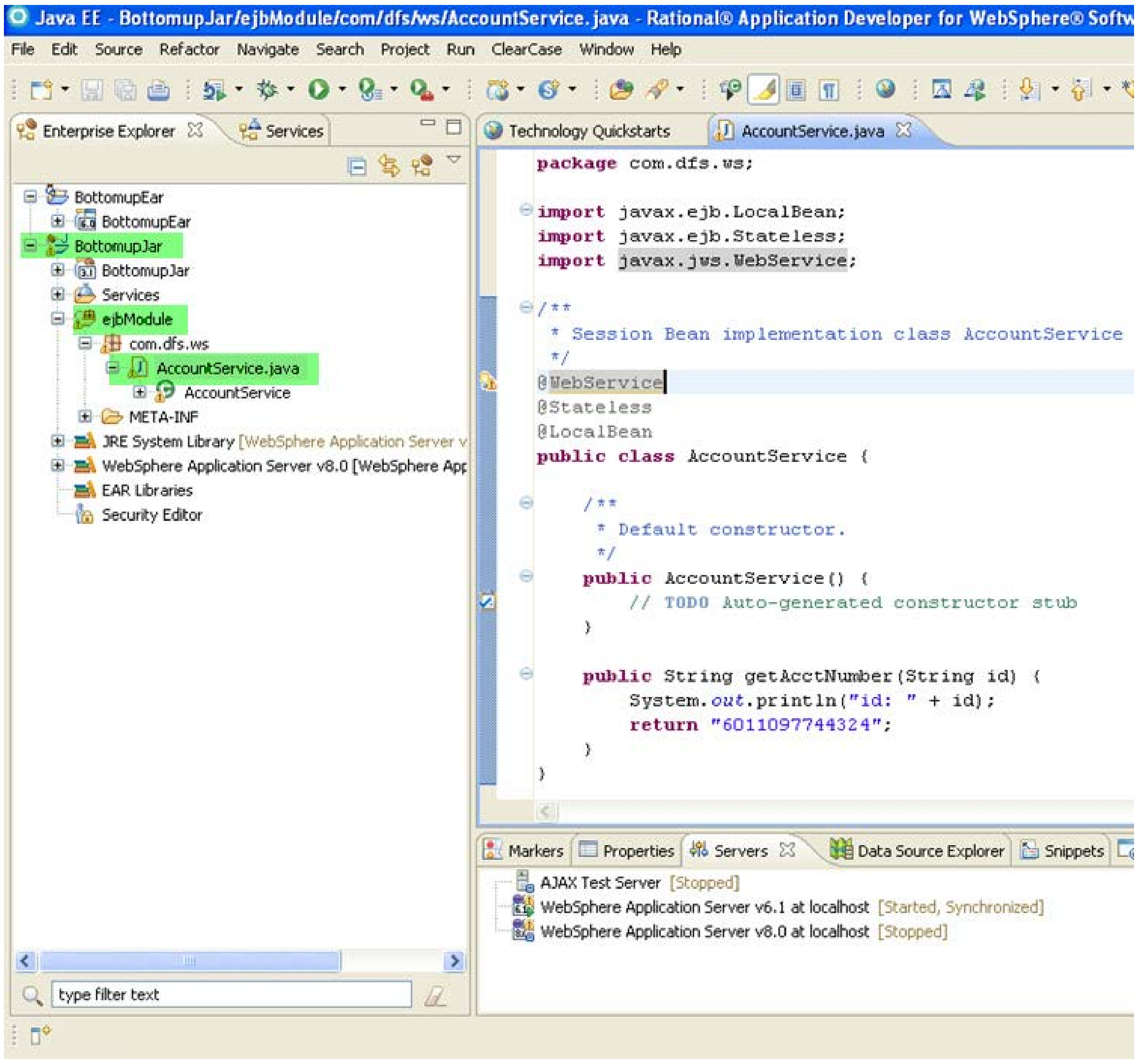
Generate ibm-webservicesclient-bnd.xmi wit

    3. The DFS standard is SOAP 1.1, so <u>do NOT select</u> **Enable SOAP 1.2**
    4. To send binary files such as jpg or PDF, select **Enable MTOM**
    5. Click **Apply** and **OK** to close the window.

# Appendix B: Create a Web Service Using Bottom-up Approach

1. Create a new Websphere server profile and a new WAS8 server that uses the new profile
2. Create an Enterprise Application Project
3. Create an EJB project
4. Create a stateless session EJB3.1
5. Annotate the EJB with JAX-WS annotations. For a complete list of JAX-WS2.2 annotations refer to: **Java API for XML Web Services (JAX-WS)**

*Note: The above link is subject to change since it is a non-DFS website.*


Java EE - BottomupJar/ejbModule/com/dfs/ws/AccountService.java - Rational® Application Developer for WebSphere® Softw
File Edit Source Refactor Navigate Search Project Run ClearCase Window Help
Enterprise Explorer
Services
Technology Quickstarts
AccountService.java
Markers
Properties
Servers
Data Source Explorer
Snippets
type filter text

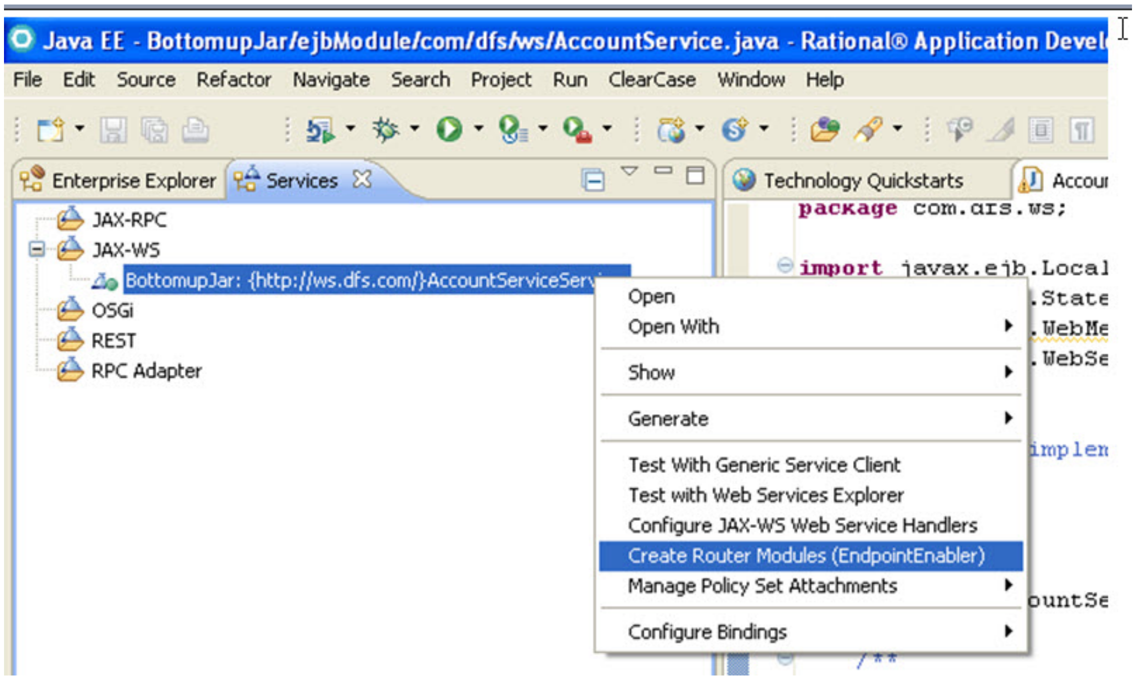
```
BottomupEar
    BottomupEar
BottomupJar
    BottomupJar
    Services
    ejbModule
        com.dfs.ws
            AccountService.java
                AccountService
        META-INF
    JRE System Library [WebSphere Application Server v
    WebSphere Application Server v8.0 [WebSphere App
    EAR Libraries
    Security Editor
```

```java
package com.dfs.ws;

import javax.ejb.LocalBean;
import javax.ejb.Stateless;
import javax.jws.WebService;

/**
 * Session Bean implementation class AccountService
 */
@WebService
@Stateless
@LocalBean
public class AccountService {

    /**
     * Default constructor.
     */
    public AccountService() {
        // TODO Auto-generated constructor stub
    }

    public String getAcctNumber(String id) {
        System.out.println("id: " + id);
        return "6011097744324";
    }
}
```

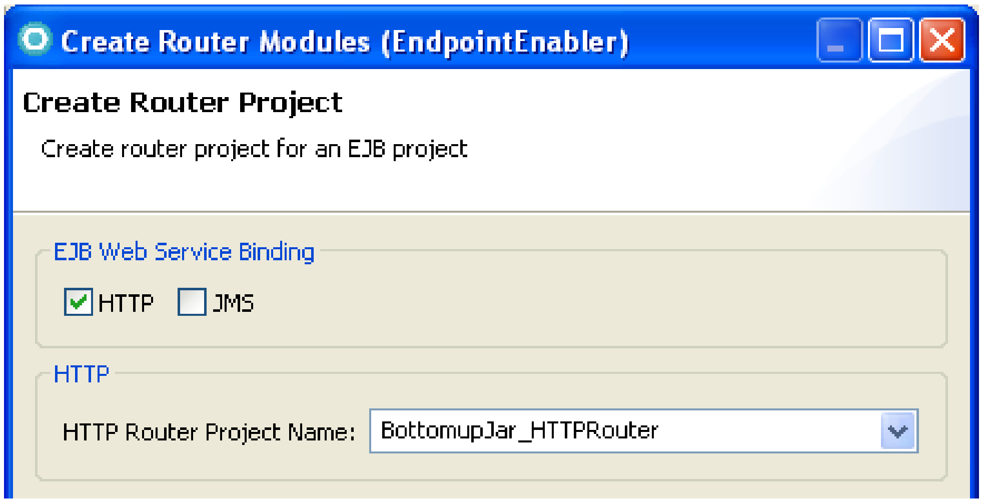
```
AJAX Test Server [Stopped]
WebSphere Application Server v6.1 at localhost [Started, Synchronized]
WebSphere Application Server v8.0 at localhost [Stopped]
```

6. Add the Enterprise Application Project to the server
7. Start the server
8. Generate a router module:

In the **Services** view, select **JAX-WS** and right-click the new service
Select **Create Router Modules (EndpointEnabler).**


Java EE - BottomupJar/ejbModule/com/dfs/ws/AccountService.java - Rational® Application Devel
File Edit Source Refactor Navigate Search Project Run ClearCase Window Help
Enterprise Explorer
Services
Technology Quickstarts
JAX-RPC
JAX-WS
BottomupJar: {http://ws.dfs.com/}AccountServiceServ
OSGi
REST
RPC Adapter
Open
Open With
Show
Generate
Test With Generic Service Client
Test with Web Services Explorer
Configure JAX-WS Web Service Handlers
Create Router Modules (EndpointEnabler)
Manage Policy Set Attachments
Configure Bindings

9. In the **Create Router Project** window, accept HTTP as the default EJB web service binding.

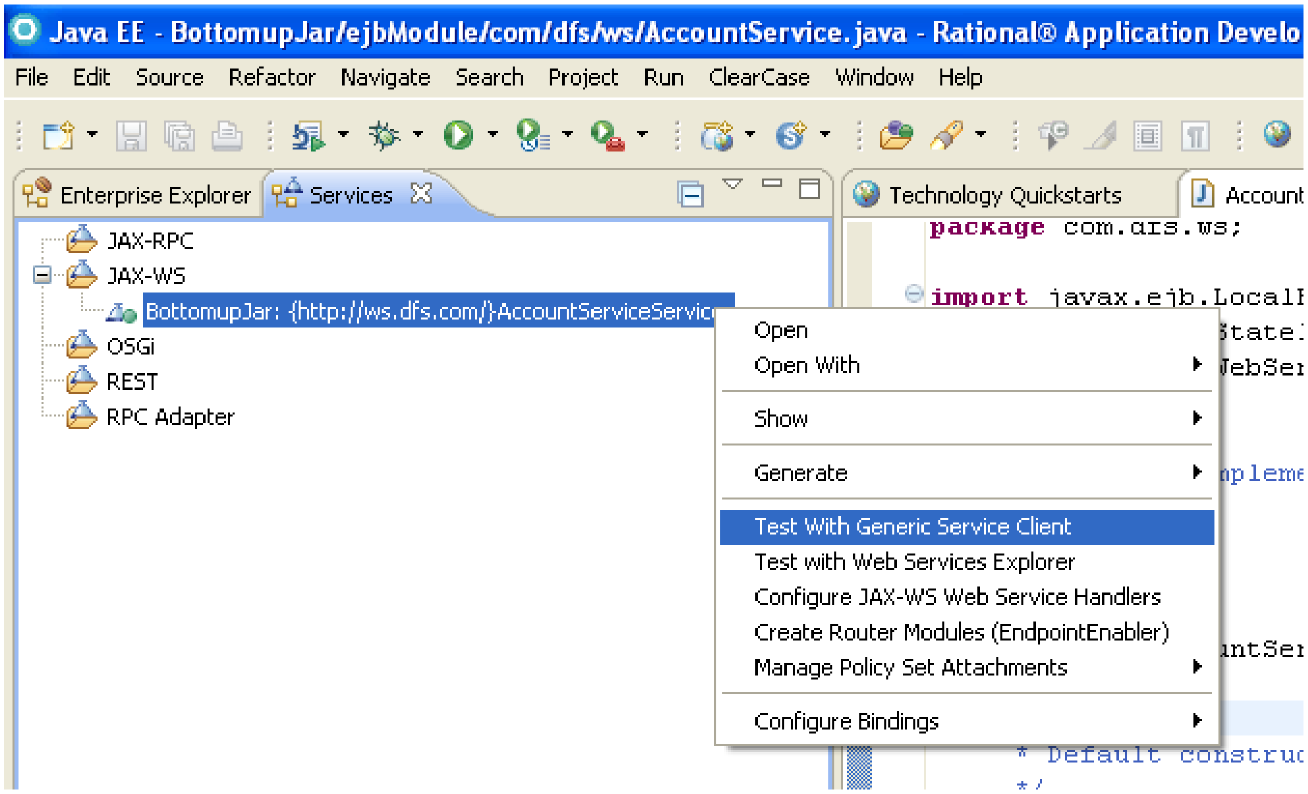

Create Router Modules (EndpointEnabler)
Create Router Project
Create router project for an EJB project
EJB Web Service Binding
HTTP
JMS
HTTP

HTTP [✓] JMS [ ]

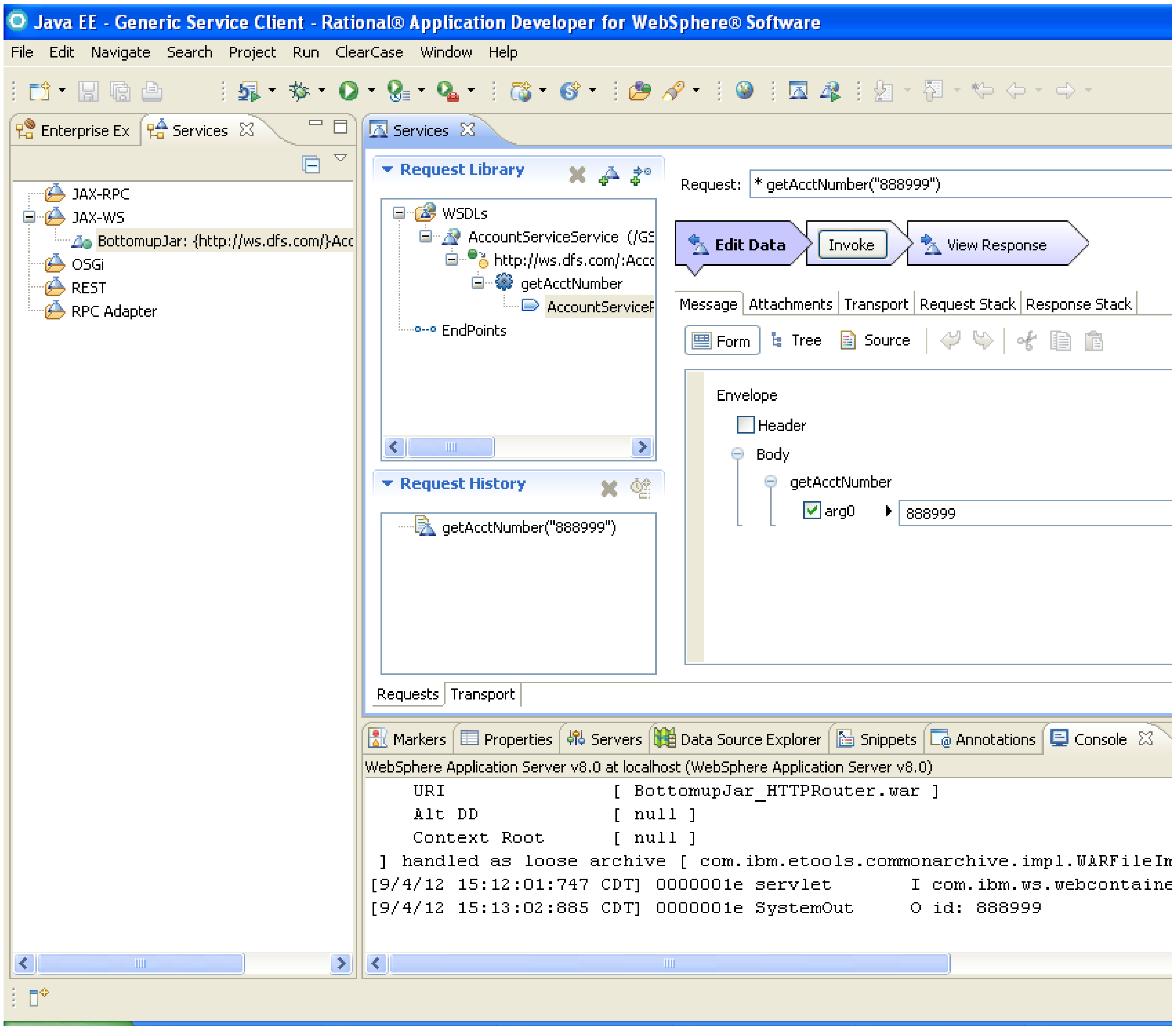
HTTP Router Project Name: BottomupJar_HTTPRouter

10. The JAX-WS web service is created. However, it is always a good practice to test newly created web services.

11. Test the web service: In the **Services** view, right-click the service and select **Test With Generic Service Client**


Java EE - BottomupJar/ejbModule/com/dfs/ws/AccountService.java - Rational® Application Develo
File Edit Source Refactor Navigate Search Project Run ClearCase Window Help
Enterprise Explorer
Services
Technology Quickstarts
Account
JAX-RPC
JAX-WS
BottomupJar: {http://ws.dfs.com/}AccountServiceServic
OSGi
REST
RPC Adapter
Open
Open With
Show
Generate
Test With Generic Service Client
Test with Web Services Explorer
Configure JAX-WS Web Service Handlers
Create Router Modules (EndpointEnabler)
Manage Policy Set Attachments
Configure Bindings

```
package com.dfs.ws;

import javax.ejb.LocalB
```


Java EE - Generic Service Client - Rational® Application Developer for WebSphere® Software
File Edit Navigate Search Project Run ClearCase Window Help
Enterprise Ex
Services
JAX-RPC
JAX-WS
BottomupJar: {http://ws.dfs.com/}Acc
OSGi
REST
RPC Adapter
Request Library
WSDLs
AccountServiceService (/GS
http://ws.dfs.com/:Acc
getAcctNumber
AccountServiceP
EndPoints
Request History
getAcctNumber("888999")
Requests Transport
Request: * getAcctNumber("888999")
Edit Data
Invoke
View Response
Message Attachments Transport Request Stack Response Stack
Form Tree Source
Envelope
Header
Body
getAcctNumber
arg0
888999
Markers Properties Servers Data Source Explorer Snippets Annotations Console

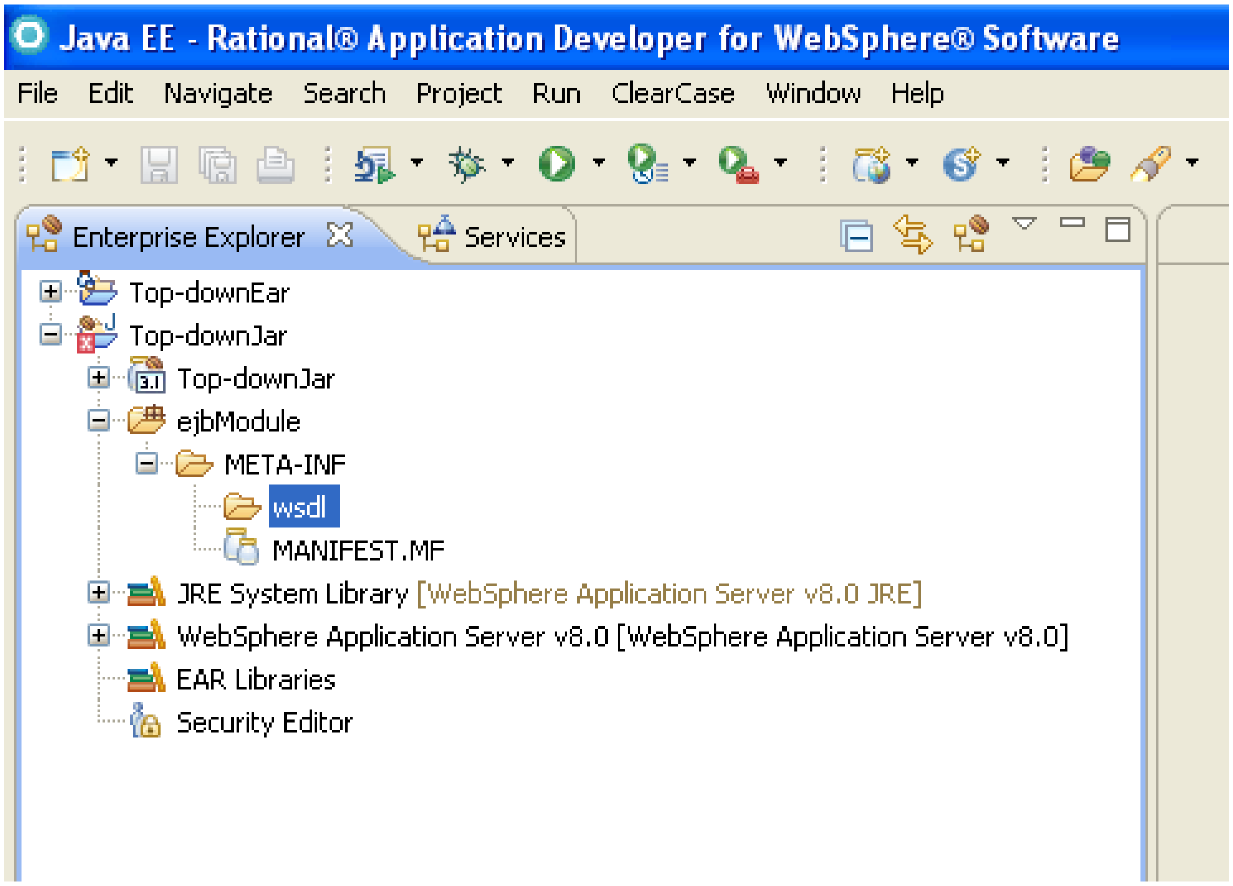
WebSphere Application Server v8.0 at localhost (WebSphere Application Server v8.0)

```
    URI                 [ BottomupJar_HTTPRouter.war ]
    Alt DD              [ null ]
    Context Root        [ null ]
 ] handled as loose archive [ com.ibm.etools.commonarchive.impl.WARFileIm
[9/4/12 15:12:01:747 CDT] 0000001e servlet       I com.ibm.ws.webcontaine
[9/4/12 15:13:02:885 CDT] 0000001e SystemOut     O id: 888999
```

12. Select the operation you want to test and enter the input data; then click the **Invoke** arrow to test the web service.

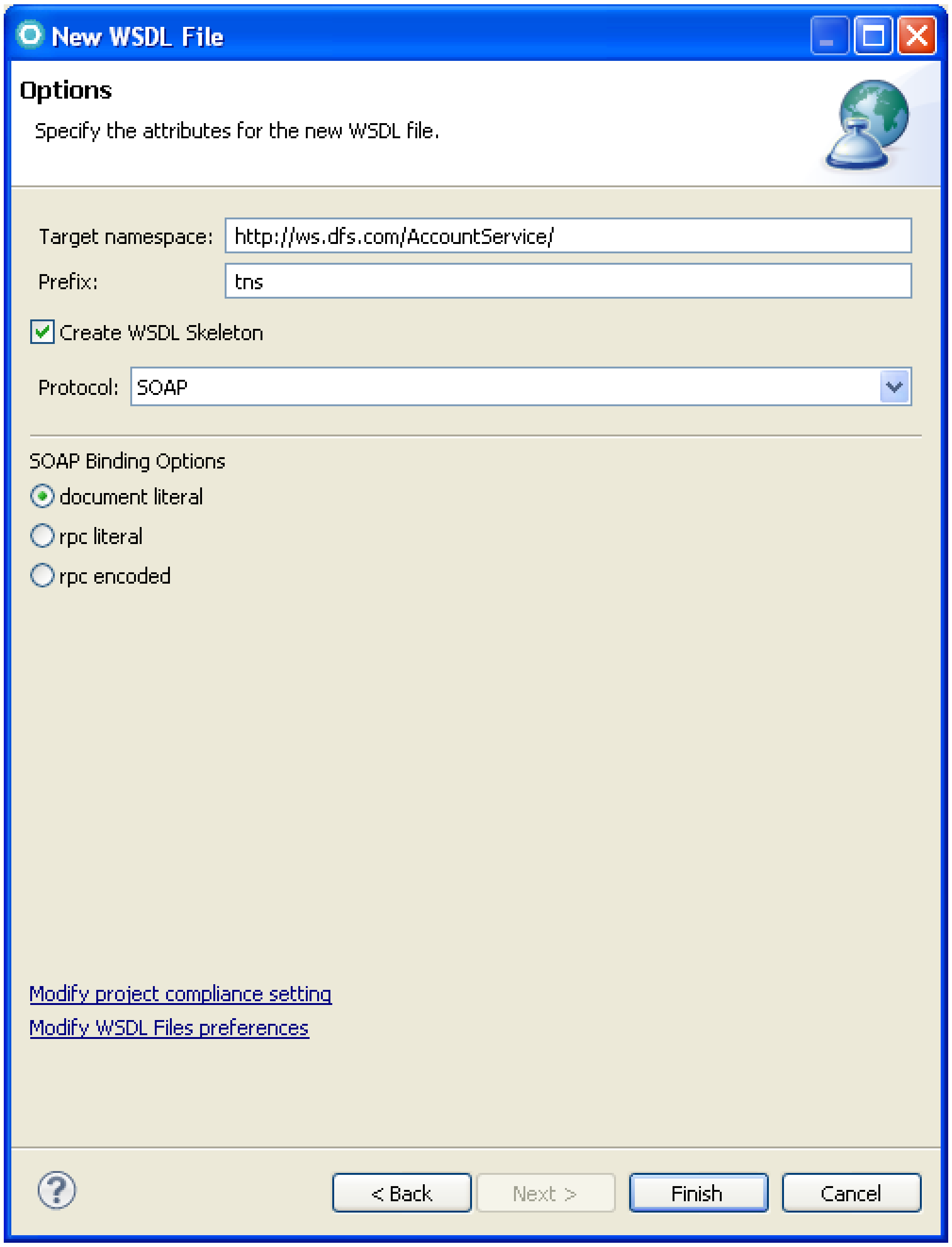
## Appendix C: Create a Web Service Using Top-down Approach

1. Create a new Websphere server profile and a new WAS8 server that uses the new profile
2. Create an Enterprise Application Project
3. Create an EJB project
4. Create a **wsdl** folder within **EJB project→ejbModule→META-INF**


Java EE - Rational® Application Developer for WebSphere® Software
File Edit Navigate Search Project Run ClearCase Window Help
Enterprise Explorer
Services
Top-downEar
Top-downJar
Top-downJar
ejbModule
META-INF
wsdl
MANIFEST.MF
JRE System Library [WebSphere Application Server v8.0 JRE]
WebSphere Application Server v8.0 [WebSphere Application Server v8.0]
EAR Libraries
Security Editor

5. Create a WSDL file within the wsdl folder:
   - right click the **wsdl** folder→**new**→**Other**
   - In the New Wizard, select **Web services**→ **WSDL**. Then click **Next**
   - Enter a name for the **WSDL** file. Click **Next**
   - In the Options window, ensure that the default options **Create WSDL Skeleton** and **document literal** are selected.

- Click **Finish.**


New WSDL File
Options
Specify the attributes for the new WSDL file.

Target namespace: http://ws.dfs.com/AccountService/

Prefix: tns

[✓] Create WSDL Skeleton

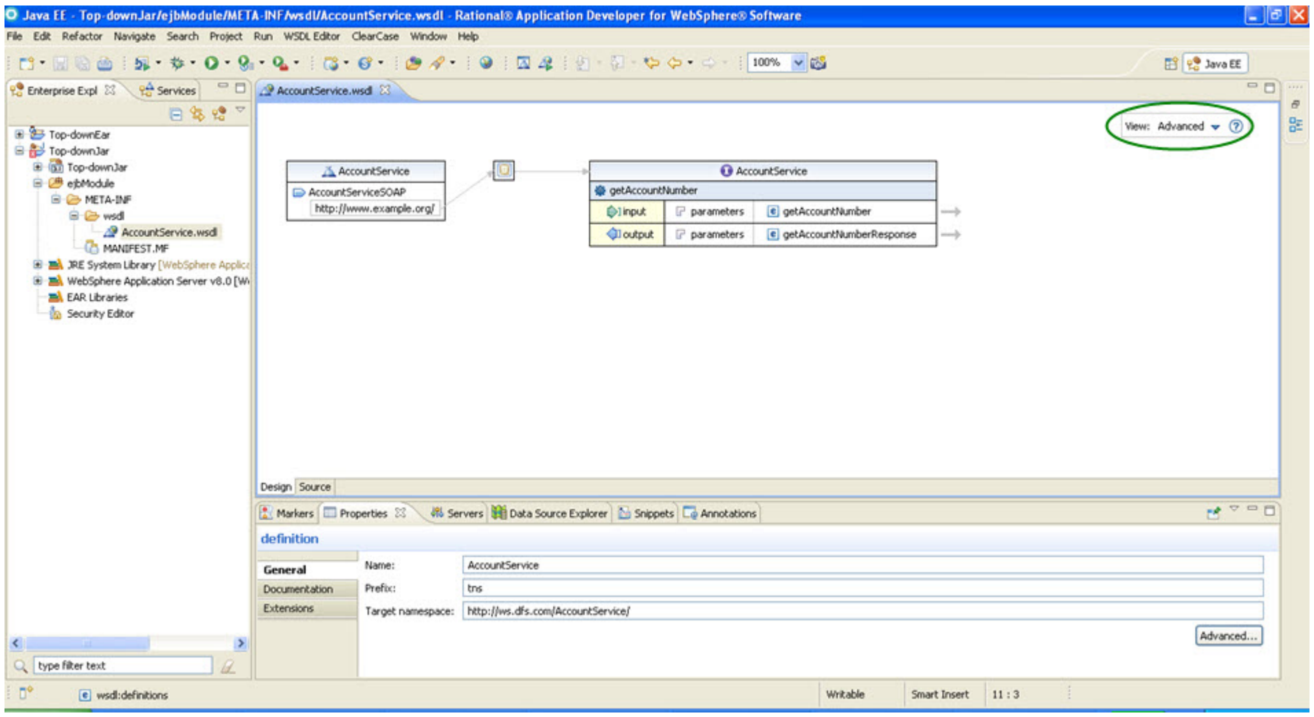
Protocol: SOAP

SOAP Binding Options

● document literal
○ rpc literal
○ rpc encoded

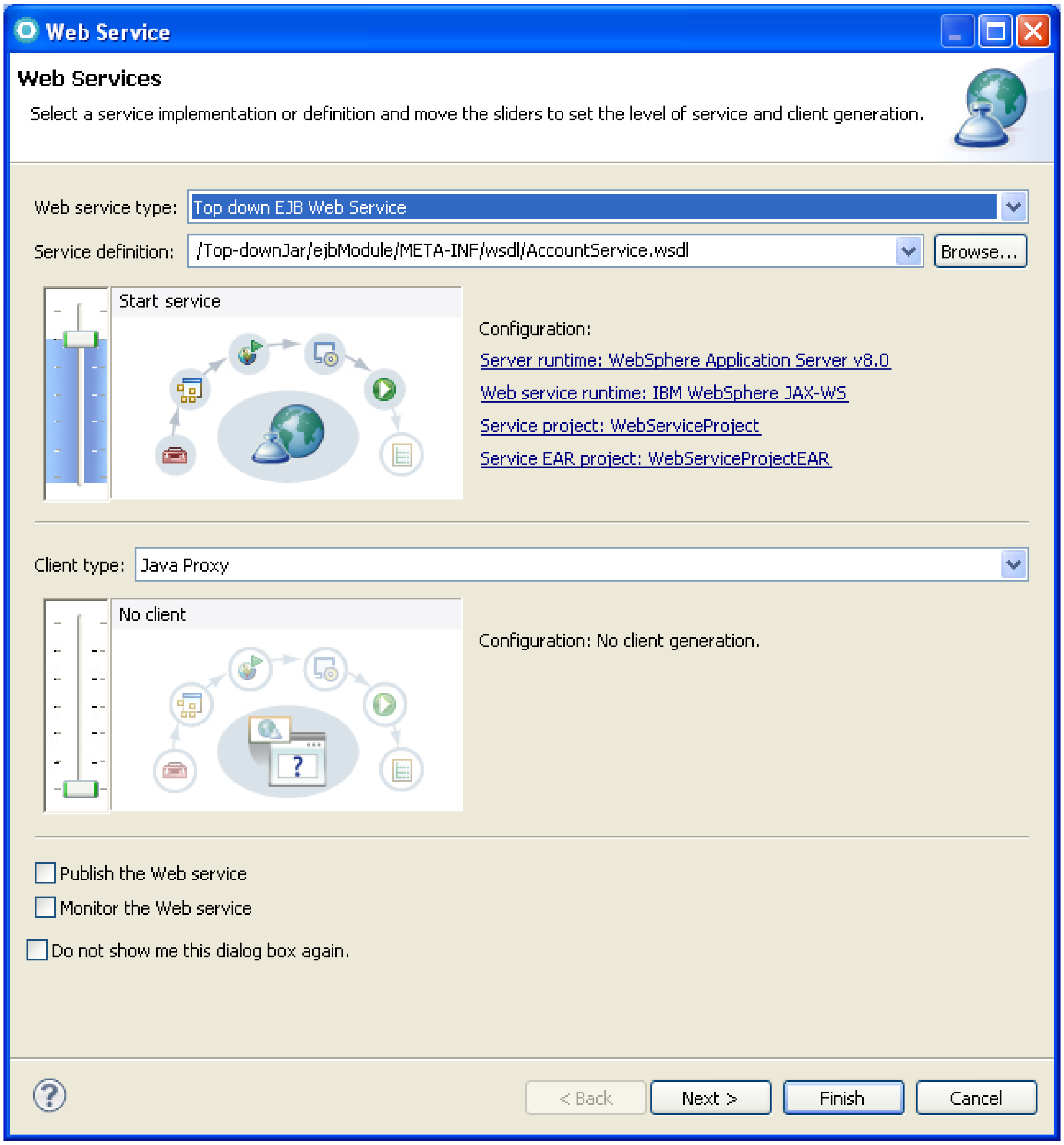
Modify project compliance setting
Modify WSDL Files preferences
< Back
Next >
Finish
Cancel

6. When the WSDL editor opens with the new WSDL file, select the **Design** tab.
7. In the WSDL editor, for View (in the upper-right corner), select **Advanced**
8. Select the **Properties** view. The WSDL file is ready to edit.


Java EE - Top-downJar/ejbModule/META-INF/wsdl/AccountService.wsdl - Rational® Application Developer for WebSphere® Software
File Edit Refactor Navigate Search Project Run WSDL Editor ClearCase Window Help
Enterprise Expl
Services
Top-downEar
Top-downJar
ejbModule
META-INF
wsdl
AccountService.wsdl
MANIFEST.MF
JRE System Library [WebSphere Applic
WebSphere Application Server v8.0 [W
EAR Libraries
Security Editor
type filter text
AccountService.wsdl
View: Advanced
AccountService
AccountServiceSOAP
http://www.example.org/
getAccountNumber
input
output
parameters
getAccountNumberResponse
Design Source
Markers Properties Servers Data Source Explorer Snippets Annotations
definition
General
Documentation
Extensions
Advanced...
wsdl:definitions
Writable Smart Insert 11 : 3

Name: AccountService
Prefix: tns
Target namespace: http://ws.dfs.com/AccountService/

9. After the WSDL is created, generate a skeleton EJB code.
   - Right-click the WSDL and select **Web Services→Generate Java bean skeleton**.
   - Change **Web service type** to **Top down EJB Web Service**.



Web Services

Select a service implementation or definition and move the sliders to set the level of service and client generation.

Web service type: Top down EJB Web Service

Service definition: /Top-downJar/ejbModule/META-INF/wsdl/AccountService.wsdl

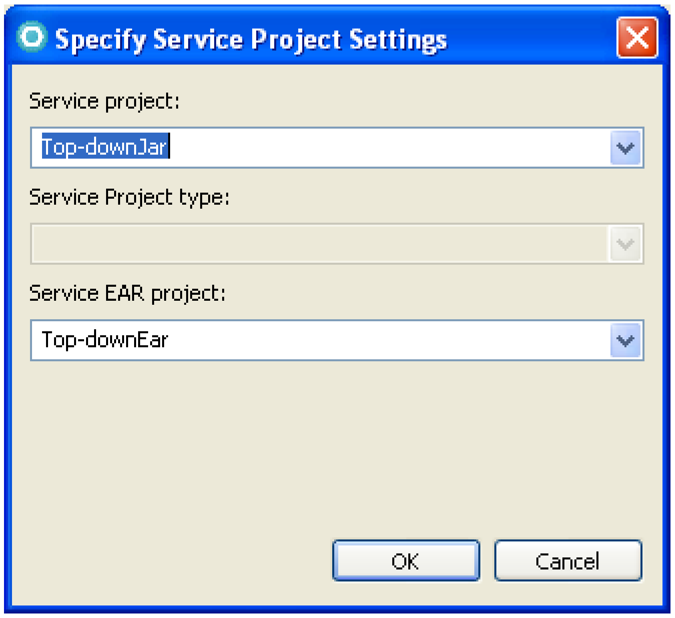
Configuration:
Server runtime: WebSphere Application Server v8.0
Web service runtime: IBM WebSphere JAX-WS
Service project: WebServiceProject
Service EAR project: WebServiceProjectEAR

Client type: Java Proxy

Configuration: No client generation.

☐ Publish the Web service
☐ Monitor the Web service
☐ Do not show me this dialog box again.

10. Click the **Service project** link
11. On the **Specify Service Project Setting** pop-up screen, change the service project to the EJB project you have created.
12. Click **OK**.

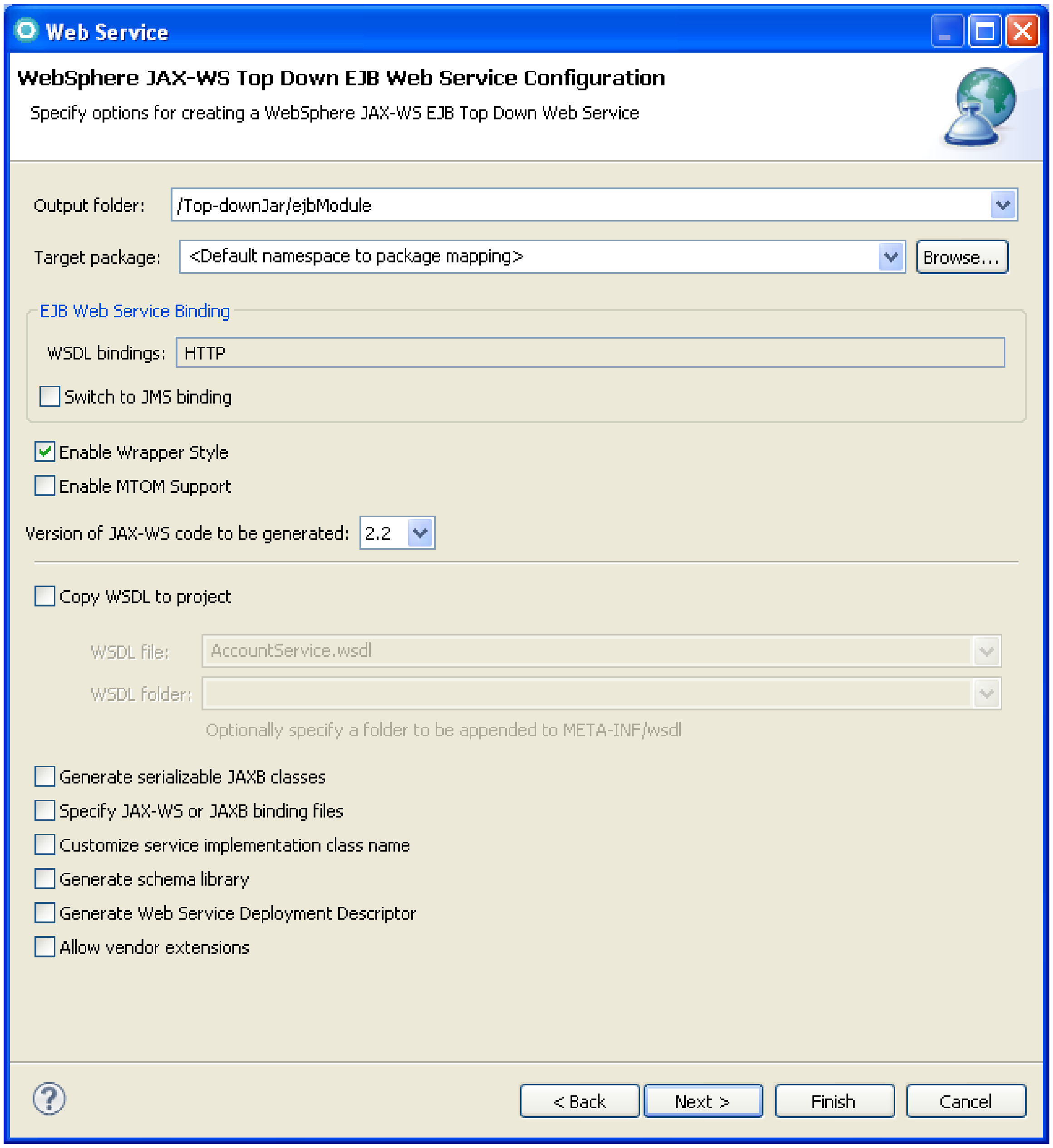


Service project: Top-downJar

Service Project type:

Service EAR project: Top-downEar

13. This returns to the **Web Service** screen.  Select any desired feature and click **Next.**



WebSphere JAX-WS Top Down EJB Web Service Configuration

Specify options for creating a WebSphere JAX-WS EJB Top Down Web Service

Output folder: /Top-downJar/ejbModule

Target package: <Default namespace to package mapping>

EJB Web Service Binding

WSDL bindings: HTTP

☐ Switch to JMS binding

☑ Enable Wrapper Style

☐ Enable MTOM Support

Version of JAX-WS code to be generated: 2.2

☐ Copy WSDL to project

WSDL file: AccountService.wsdl

WSDL folder:

Optionally specify a folder to be appended to META-INF/wsdl

☐ Generate serializable JAXB classes

☐ Specify JAX-WS or JAXB binding files

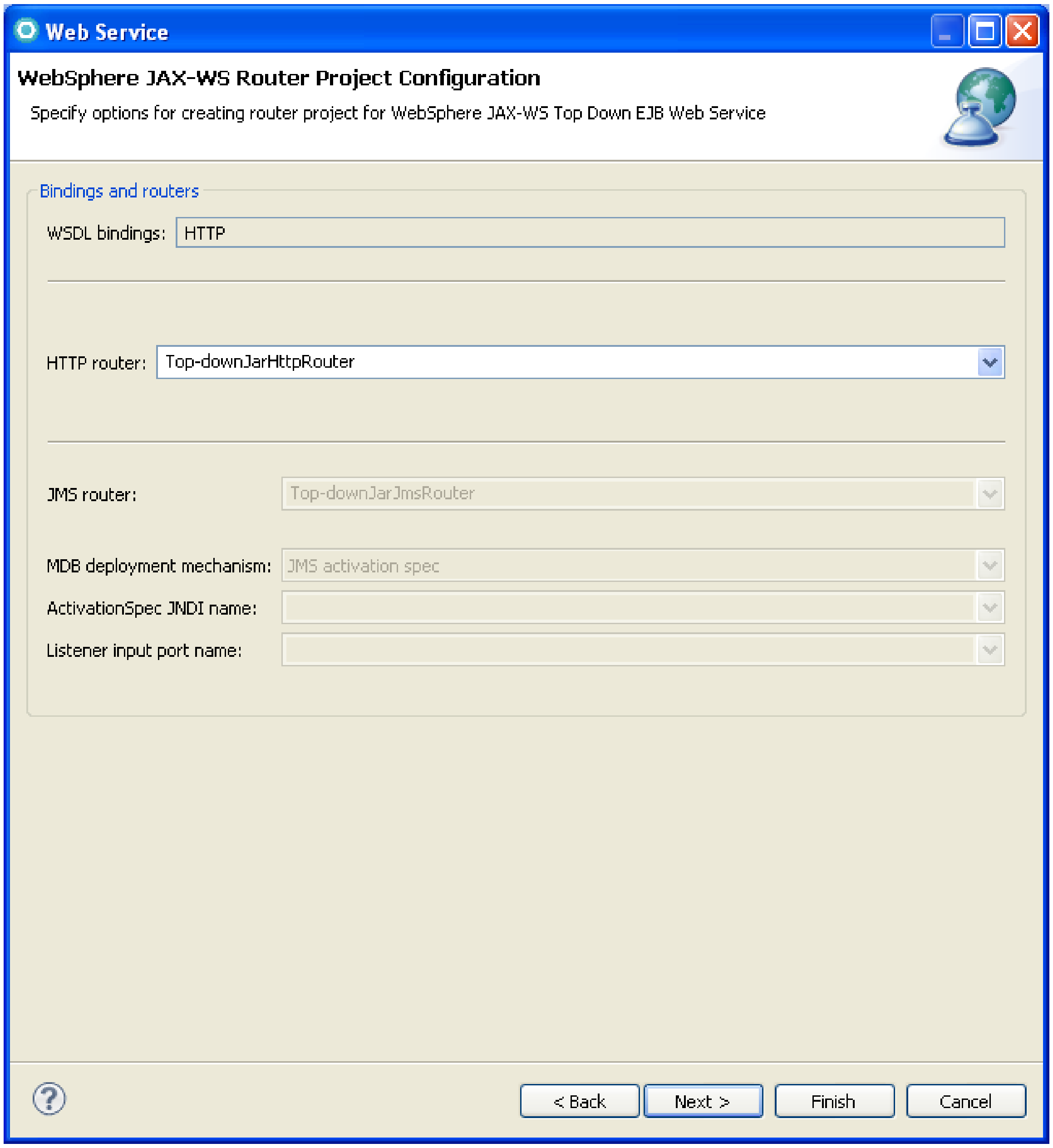
☐ Customize service implementation class name

☐ Generate schema library

☐ Generate Web Service Deployment Descriptor

☐ Allow vendor extensions

14. Select **Next** on the **WebSphere JAX-WS Router Project Configuration** page



**WebSphere JAX-WS Router Project Configuration**

Specify options for creating router project for WebSphere JAX-WS Top Down EJB Web Service

Bindings and routers

WSDL bindings: HTTP

HTTP router: Top-downJarHttpRouter

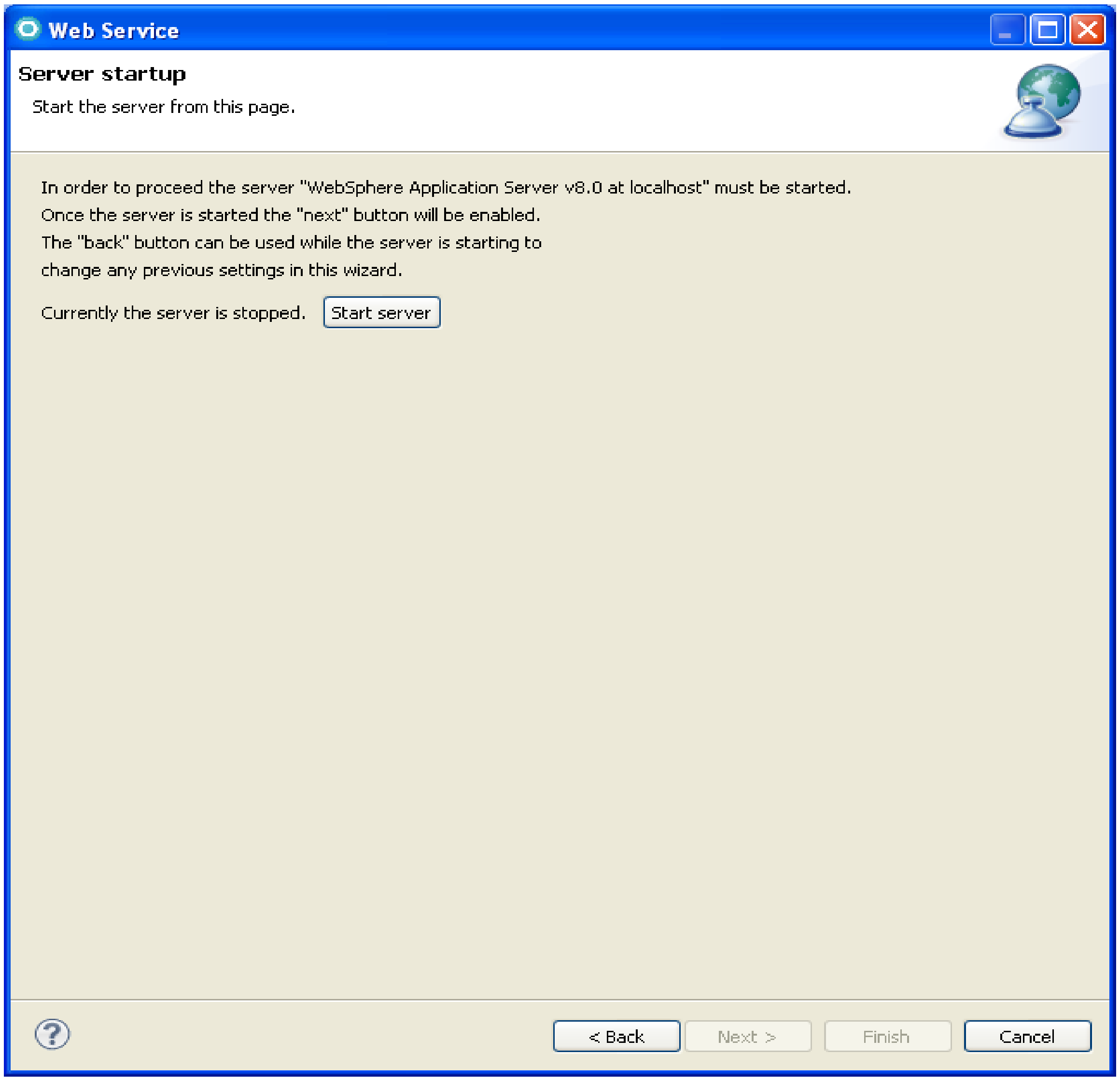
JMS router: Top-downJarJmsRouter

MDB deployment mechanism: JMS activation spec

ActivationSpec JNDI name:

Listener input port name:

15. If the server has not started, click **Start server**



Server startup
Start the server from this page.

In order to proceed the server "WebSphere Application Server v8.0 at localhost" must be started.
Once the server is started the "next" button will be enabled.
The "back" button can be used while the server is starting to
change any previous settings in this wizard.

Currently the server is stopped.

16. After the server starts, click **Finish** to complete the Web Service generation process.
17. Review the generated classes and add necessary coding.
18. Test the web service. Refer to Appendix A, test the Web Service with RAD Generic Service Client.

# Appendix D: DFS Standard WS Exception Definition

This appendix defines the **`DFSExceptionInterface`**. When creating Web Services, application developers have to create their own WS Exception that implements this interface and extends directly from **`java.lang.Exception`**. The Web Service interface methods can only throw and must throw this Exception. However, developers can define and use other application exceptions in the sub-layers of their services.

This appendix also provides a WS Exception implementation example. Developers can follow this example to create the WS Exception of their own. The implementing Exceptions must declare three **`string`** attributes: **`exceptionType`**, **`exceptionCode`** and **`alertMessage`**. The **`exceptionType`** attribute must be set using one of the constants defined in this interface.

The error codes are four digits long. Each error codes defined in this interface represent the generic error within that error category. Developers are free to define sub-categories using the last two digits. For example, **`3100`** is defined for **invalid input** which is a generic error. If developers want to sub-categorize it, they can use any value between 3101 – 3199, e.g. **`3110`** for an error such as “card number out of range.”

The Following is the **`DFSExceptionInterface`** definition that is included in the DFSCommon library.

```
package com.ecc.dfscommon;

public interface DFSExceptionInterface {

      /**
       * Set the “exceptionType” to this constant when
       * the input data is incorrect. Developers are free to define
       * any sub-type errors within 3101 to 3199 range.
       */
      public static final String SERVICE_ERROR_INVALID_INPUT = "3100";

      /**
       * Set the “exceptionType” to this constant when
       * an Exception is received when processing the request.
       * e.g. NullPointerExcetion
       * Developers are free to define any sub-type errors within
       * 3201 to 3299 range.
       */
      public static final String SERVICE_ERROR_PROCESSING = "3200";

      /**
       * Set the “exceptionType” to this constant when
       * an Exception is received when calling other services
       * Developers are free to define any sub-type errors within
       * 3301 to 3399 range.
       */
      public static final String SERVICE_ERROR_INVOCATION = "3300";

      /**
```

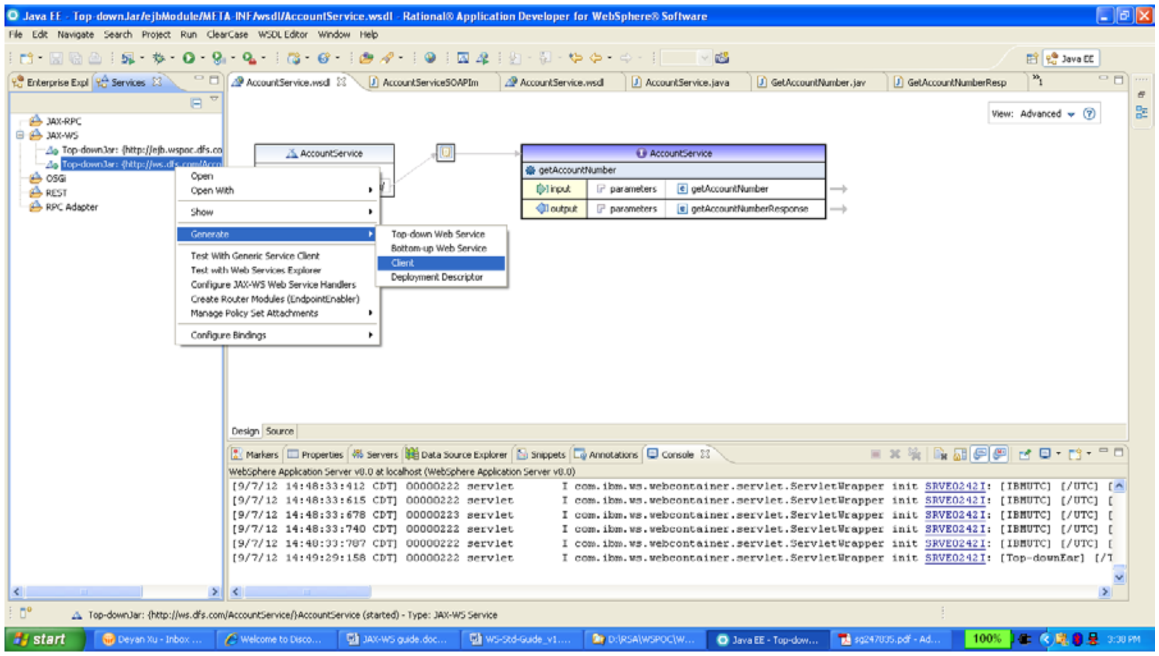
```
         * Set the “exceptionType” to this constant when
         * the Exception is caused by the application data etc.
         * e.g. duplicate primary key exception when inserting a row
         * to a db table.
         * Developers are free to define any sub-type errors within
         * 3701 to 3799 range.
         */
        public static final String SERVICE_ERROR_RESOURCE_APP_LEVEL = "3700";

        /**
         * Set the “exceptionType” to this constant when
         * the resource does not perform as expected.
         * e.g. timeout exception, connection closed, etc.
         * Developers are free to define any sub-type errors within
         * 3801 to 3899 range.
         */
        public static final String SERVICE_ERROR_RESOURCE_DEGRADATION = "3800";

        /**
         * Set the “exceptionType” to this constant when
         * a critical resource Exception is received.
         * e.g. the error code in the exception indicates that the server
         * is down, or in a hung state and is not able to accept requests.
         * Developers are free to define any sub-type errors within
         * 3901 to 3999 range.
         */
        public static final String SERVICE_ERROR_RESOURCE_CRITICAL_EXCEPTION =
        3900";

        /**
         * The implementing Exception must declare the following attributes:
         * String exceptionType
         * String exceptionCode
         * String alertMessage
         */

        public String getExceptionType();
        public String getExceptionCode();
        public String getAlertMessage();

        public void setExceptionType(String exceptionType);
        public void setExceptionCode(String exceptionCode);
        public void setAlertMessage(String alertMessage);
}
```

The following is an example of service Exception that implements the **`DFSExceptionInterface`**:

```
package com.xx.xxx.xxxx; //use the specific services own package name

public final class XxxxException extends Exception implements
DFSExceptionInterface {
    /**
     * The “exceptionType” will be used to help ESR to determine
     * how to route the SOAP message incase of service Exception.
     * It must be set using one of the constants defined in the
     * DFSExceptionInterface. The developers should choose a
     * proper constant based on the type of the original
     * Exceptions received from the sub-layers.
     */
    private String exceptionType = null;

    /**
     * The “exceptionCode” will be used by the service developers
     * to help debugging the issue. It is totally up to the
     * developers to decide how to use the “exceptionCode” so
     * that valuable information can be retrieved when they
     * debugging the issue. A good example would be to set
     * this attribute to the Oracle database error code when
     * the original Exception is a SQLException and using an
     * Oracle database.
     */
    private String exceptionCode = null;

    /**
     * The “alertMessage” will be used by the ESR to send alerts
     * to the Commend Center. It should contain good useful
     * information so that the support team can quickly
     * identify and address the issue.
     * Developers may set the “alertMessage” to include any useful
     * information such as: the name of the original class and method
     * that throws the Exception etc.
     * However, developers should not include the following information
     * since the ESR will pre-append this information to the "alertMessage":
     * 1. the service name and method that throws the Exception
     * 2. the exceptionType
     * 3. the original exceptionCode
     */
    private String alertMessage = null;

    // default constructor
    public XxxxException(){}

   // create getters and setters of the class attributes here
}
```

# Appendix E: Steps to Create Client Delegate

1. Create a utility project or a java project to hold the client proxy
2. Generate Web Service client proxy from the service.


Java EE - Top-downJar/ejbModule/META-INF/wsdl/AccountService.wsdl - Rational® Application Developer for WebSphere® Software
JAX-RPC
JAX-WS
Top-downJar: {http://ejb.wspoc.dfs.co
Top-downJar: {http://ws.dfs.com/Acco
OSGi
REST
RPC Adapter
Open
Open With
Show
Generate
Test With Generic Service Client
Test with Web Services Explorer
Configure JAX-WS Web Service Handlers
Create Router Modules (EndpointEnabler)
Manage Policy Set Attachments
Configure Bindings
Top-down Web Service
Bottom-up Web Service
Client
Deployment Descriptor
AccountService
getAccountNumber
input
output
parameters
getAccountNumberResponse
Design
Source
Console

```
WebSphere Application Server v8.0 at localhost (WebSphere Application Server v8.0)
[9/7/12 14:48:33:412 CDT] 00000222 servlet       I com.ibm.ws.webcontainer.servlet.ServletWrapper init SRVE0242I: [IBMUTC] [/UTC] [
[9/7/12 14:48:33:615 CDT] 00000222 servlet       I com.ibm.ws.webcontainer.servlet.ServletWrapper init SRVE0242I: [IBMUTC] [/UTC] [
[9/7/12 14:48:33:678 CDT] 00000223 servlet       I com.ibm.ws.webcontainer.servlet.ServletWrapper init SRVE0242I: [IBMUTC] [/UTC] [
[9/7/12 14:48:33:740 CDT] 00000222 servlet       I com.ibm.ws.webcontainer.servlet.ServletWrapper init SRVE0242I: [IBMUTC] [/UTC] [
[9/7/12 14:48:33:787 CDT] 00000222 servlet       I com.ibm.ws.webcontainer.servlet.ServletWrapper init SRVE0242I: [IBMUTC] [/UTC] [
[9/7/12 14:49:29:158 CDT] 00000222 servlet       I com.ibm.ws.webcontainer.servlet.ServletWrapper init SRVE0242I: [Top-downEar] [/T
```

Top-downJar: {http://ws.dfs.com/AccountService/}AccountService (started) - Type: JAX-WS Service

3. Several classes are generated. The class ***xxxPortProxy.java*** contains all exposed service methods and will be used by the SIF to invoke the Web Service.
4. Follow ECC Service Invocation Utility ICD document to incorporate SIF.

*Note: IBM recommends caching the proxy that is generated from the WSDL to improve performance. SIF has implemented proxy caching within its framework, so if SIF is used, there is no need to cache the proxy. However, if SIF is not being used, then cache the proxy within the consuming applications. This is mainly for 3rd party provided WSDL.*

## Revision History and Contributors

| Revision Date | Approved By | Author | Changes | Revision |
|---|---|---|---|---|
| 09/15/2012 | | Deyan Xu | Initial Draft | 2.2 |
| 11/14/2012 | | Deyan Xu | Updates | 2.4 |
| 01/30/2013 | | R Hauenstein | Updates and reformatted | 3.0 |
| | | | | |

**Reviewers/Contributors:**

| Name | Department/Contribution |
|---|---|
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |

## Meta Tags

**Note: Meta Tag in bold is the unique identifier**

**bt_soap_web_services_standard**

bt_soap_web_services_development_guide

bt_web services_ejb

bt_web_services_jax