Oracle® Communications Billing and Revenue Management

Web Services Manager

Release 15.1
F93204-01
April 2025

ORACLE

Oracle Communications Billing and Revenue Management Web Services Manager, Release 15.1

F93204-01

Contents

Preface

This guide provides guidelines for installing and setting up Oracle Communications Billing and Revenue Management (BRM) Web Services Manager.

Audience

This document is intended for systems integrators, system administrators, database administrators, and other individuals who are responsible for installing, configuring, and customizing Web services for BRM.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customer access to and use of Oracle support services will be pursuant to the terms and conditions specified in their Oracle order for the applicable services.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

1

Using Web Services

Learn how to use Oracle Communications Billing and Revenue Management (BRM) Web Services Manager, which enables BRM opcodes to be exposed through web services. Web Services Manager supports SOAP web services and is packaged as an integration pack.

Topics in this document:

About WSDL Files and BRM Opcodes

Web Services Manager exposes BRM opcodes as operations through different web services. The web services define the opcodes that can be called and the attributes to include.

The web service APIs are grouped by functional area. For example, the **BRMBillServices** web service defines the billing web service APIs, and the **BRMPymtServices** web service defines the payment web service APIs. Web Services Manager includes one WSDL file for each web service.

Web Services Manager contains two different types of WSDL files. One type is for web services that support the payload as an XML string data type. The second type is for web services that support the payload as an XML element data type. For example:

- The **BRMBalService** web service defines balances web service APIs that take the payload as an XML string data type.
- The **BRMBalService_v2** web service defines balances web service APIs that take the payload as an XML element data type.

  File names with a **_v2** suffix support the payload as an XML element data type.

> **Note:**
>
> For deployments into a web server, such as Oracle WebLogic Server or Tomcat, the WSDL and schema (XSD) files for web services that support the payload as an XML string data type are included in the **infranetwebsvc.war** file. If you customize any web services, copy the customized schema files and WSDL files to the **infranetwebsvc.war** file.

Web services that support the payload as an XML element data type describe the input in a well-defined structure. Any standards-compliant SOAP development application can generate a client stub.

Table 1-1 describes the web services included that take the payload as an XML string.

Table 1-1 Web Services Included in Web Services Manager that Take the Payload as an XML String

Web Service Name	Description
BRMARServices	Defines the accounts receivable web service, which includes the following opcodes: • PCM_OP_AR_ACCOUNT_ADJUSTMENT • PCM_OP_AR_BILL_ADJUSTMENT • PCM_OP_AR_GET_ACCT_ACTION_ITEMS • PCM_OP_AR_GET_ACCT_BAL_SUMMARY • PCM_OP_AR_GET_ACCT_BILLS • PCM_OP_AR_GET_BAL_SUMMARY • PCM_OP_AR_GET_BILL_ITEMS • PCM_OP_AR_ITEM_ADJUSTMENT • PCM_OP_AR_EVENT_ADJUSTMENT • PCM_OP_AR_GET_ACTION_ITEMS • PCM_OP_AR_GET_BILLS • PCM_OP_AR_RESOURCE_AGGREGATION See "Accounts Receivable FM Standard Opcodes" in *BRM Opcode Guide* for more information.
BRMBalServices	Defines the balances web service, which includes the following opcodes: • PCM_OP_BAL_GET_BALANCES • PCM_OP_BAL_GET_BAL_GRP_AND_SVC • PCM_OP_BAL_GET_ACCT_BAL_GRP_AND_SVC • PCM_OP_BAL_GET_ACCT_BILLINFO • PCM_OP_BAL_GET_ECE_BALANCES **Note:** You must perform configuration steps before calling this opcode. See "Configuring BRM to Use PCM_OP_BAL_GET_ECE_BALANCES". See "Balance FM Standard Opcodes" in *BRM Opcode Guide* for more information.
BRMBillServices	Defines the billing web service, which includes the following opcodes: • PCM_OP_BILL_GET_ITEM_EVENT_CHARGE_DISCOUNT • PCM_OP_BILL_GROUP_MOVE_MEMBER • PCM_OP_BILL_MAKE_BILL_NOW • PCM_OP_BILL_DEBIT • PCM_OP_BILL_GROUP_GET_PARENT See "Billing FM Standard Opcodes" in *BRM Opcode Guide* for more information.
BRMCollectionsServices	Defines the collections web service, which includes the following opcode: • PCM_OP_COLLECTIONS_SET_ACTION_STATUS See "Collections Manager FM Standard Opcodes" in *BRM Opcode Guide* for more information.
BRMCustcareServices	Defines the customer care web service, which includes the following opcode: • PCM_OP_CUSTCARE_MOVE_ACCT

Table 1-1 (Cont.) Web Services Included in Web Services Manager that Take the Payload as an XML String

Web Service Name	Description
BRMCustServices	Defines the customer web service, which includes the following opcodes: • PCM_OP_CUST_COMMIT_CUSTOMER • PCM_OP_CUST_MODIFY_CUSTOMER • PCM_OP_CUST_UPDATE_CUSTOMER • PCM_OP_CUST_UPDATE_SERVICES • PCM_OP_CUST_DELETE_ACCT • PCM_OP_CUST_DELETE_PAYINFO • PCM_OP_CUST_CREATE_PROFILE • PCM_OP_CUST_MODIFY_PROFILE • PCM_OP_CUST_DELETE_PROFILE See "Customer FM Standard Opcodes" in *BRM Opcode Guide* for more information.
BRMInvServices	Defines the invoicing web service, which includes the following opcode: • PCM_OP_INV_VIEW_INVOICE **Important:** You must configure your client application to convert the invoice data received from the PCM_OP_INV_VIEW_INVOICE opcode into the appropriate format. See "About Invoicing Output XML Data" in *BRM JCA Resource Adapter*. See "Invoicing FM Standard Opcodes" in *BRM Opcode Guide* for more information.
BRMPymtServices	Defines the payment web service, which includes the following opcode: • PCM_OP_PYMT_COLLECT See "Payment FM Standard Opcodes" in *BRM Opcode Guide* for more information.
BRMReadServices	Defines the read web service, which includes the following opcodes: • PCM_OP_READ_FLDS • PCM_OP_READ_OBJ • PCM_OP_SEARCH See "LDAP Base Opcodes" in *BRM Opcode Guide* for more information.
BRMSubscriptionServices	Defines the subscription web service, which includes the following opcodes: • PCM_OP_SUBSCRIPTION_CANCEL_PRODUCT • PCM_OP_SUBSCRIPTION_CANCEL_DISCOUNT • PCM_OP_SUBSCRIPTION_CANCEL_SUBSCRIPTION • PCM_OP_SUBSCRIPTION_CHANGE_DEAL • PCM_OP_SUBSCRIPTION_PURCHASE_DEAL • PCM_OP_SUBSCRIPTION_SET_BUNDLE • PCM_OP_SUBSCRIPTION_SET_DISCOUNT_STATUS • PCM_OP_SUBSCRIPTION_SET_DISCOUNTINFO • PCM_OP_SUBSCRIPTION_SET_PRODINFO • PCM_OP_SUBSCRIPTION_SET_PRODUCT_STATUS • PCM_OP_SUBSCRIPTION_TRANSFER_SUBSCRIPTION • PCM_OP_SUBSCRIPTION_GET_PURCHASED_OFFERINGS See "Subscription Management FM Standard Opcodes" in *BRM Opcode Guide* for more information.

Table 1-2 describes the web services that take the payload as an XML element.

Table 1-2 Web Services Included in Web Services Manager that Take the Payload as an XML Element

Web Service Name	Description
BRMACTServices_v2	Defines the activity web service, which includes the following opcodes: • PCM_OP_ACT_FIND • PCM_OP_ACT_LOAD_SESSION See "Activity FM Standard Opcodes" in *BRM Opcode Guide* for more information.
BRMARServices_v2	Defines the accounts receivable web service, which includes the following opcodes: • PCM_OP_AR_ACCOUNT_ADJUSTMENT • PCM_OP_AR_ACCOUNT_WRITEOFF • PCM_OP_AR_BILL_ADJUSTMENT • PCM_OP_AR_BILL_DISPUTE • PCM_OP_AR_BILL_SETTLEMENT • PCM_OP_AR_BILL_WRITEOFF • PCM_OP_AR_BILLINFO_WRITEOFF • PCM_OP_AR_EVENT_ADJUSTMENT • PCM_OP_AR_EVENT_DISPUTE • PCM_OP_AR_EVENT_SETTLEMENT • PCM_OP_AR_GET_ACCT_ACTION_ITEMS • PCM_OP_AR_GET_ACCT_BAL_SUMMARY • PCM_OP_AR_GET_ACCT_BILLS • PCM_OP_AR_GET_ACTION_ITEMS • PCM_OP_AR_GET_BAL_SUMMARY • PCM_OP_AR_GET_BILLS • PCM_OP_AR_GET_BILL_ITEMS • PCM_OP_AR_GET_DISPUTES • PCM_OP_AR_GET_DISPUTE_DETAILS • PCM_OP_AR_GET_ITEMS • PCM_OP_AR_GET_ITEM_DETAILS • PCM_OP_AR_ITEM_ADJUSTMENT • PCM_OP_AR_ITEM_DISPUTE • PCM_OP_AR_ITEM_SETTLEMENT • PCM_OP_AR_ITEM_WRITEOFF • PCM_OP_AR_RESOURCE_AGGREGATION See "Accounts Receivable FM Standard Opcodes" in *BRM Opcode Guide* for more information.

Table 1-2 (Cont.) Web Services Included in Web Services Manager that Take the Payload as an XML Element

Web Service Name	Description
BRMBALServices_v2	Defines the balances web service, which includes the following opcodes: • PCM_OP_BAL_CHANGE_VALIDITY • PCM_OP_BAL_GET_BALANCES • PCM_OP_BAL_GET_ECE_BALANCES **Note:** You must perform configuration steps before calling this opcode. See "Configuring BRM to Use PCM_OP_BAL_GET_ECE_BALANCES". • PCM_OP_BAL_GET_BAL_GRP_AND_SVC • PCM_OP_BAL_GET_ACCT_BAL_GRP_AND_SVC • PCM_OP_BAL_GET_ACCT_BILLINFO See "Balance FM Standard Opcodes" in *BRM Opcode Guide* for more information.
BRMBILLServices_v2	Defines the billing web service, which includes the following opcodes: • PCM_OP_BILL_DEBIT • PCM_OP_BILL_FIND • PCM_OP_BILL_GET_ITEM_EVENT_CHARGE_DISCOUNT • PCM_OP_BILL_GROUP_GET_PARENT • PCM_OP_BILL_GROUP_MOVE_MEMBER • PCM_OP_BILL_ITEM_EVENT_SEARCH • PCM_OP_BILL_ITEM_REFUND • PCM_OP_BILL_MAKE_BILL_NOW • PCM_OP_BILL_REVERSE • PCM_OP_BILL_SET_LIMIT_AND_CR • PCM_OP_BILL_VIEW_INVOICE See "Billing FM Standard Opcodes" in *BRM Opcode Guide* for more information.
BRMCOLLECTIONSServices_v2	Defines the collections web service, which includes the following opcode: • PCM_OP_COLLECTIONS_SET_ACTION_STATUS See "Collections Manager FM Standard Opcodes" in *BRM Opcode Guide* for more information.
BRMCUSTCAREServices_v2	Defines the customer care web service, which includes the following opcode: • PCM_OP_CUSTCARE_MOVE_ACCT

Table 1-2 (Cont.) Web Services Included in Web Services Manager that Take the Payload as an XML Element

Web Service Name	Description
BRMCUSTServices_v2	Defines the customer web service, which includes the following opcodes: • PCM_OP_CUST_COMMIT_CUSTOMER • PCM_OP_CUST_CREATE_PROFILE • PCM_OP_CUST_DELETE_ACCT • PCM_OP_CUST_DELETE_PAYINFO • PCM_OP_CUST_DELETE_PROFILE • PCM_OP_CUST_FIND • PCM_OP_CUST_FIND_PAYINFO • PCM_OP_CUST_FIND_PROFILE • PCM_OP_CUST_GET_NOTE • PCM_OP_CUST_MODIFY_CUSTOMER • PCM_OP_CUST_MODIFY_PROFILE • PCM_OP_CUST_SET_NOTE • PCM_OP_CUST_SET_STATUS • PCM_OP_CUST_SET_TAXINFO • PCM_OP_CUST_UPDATE_CUSTOMER • PCM_OP_CUST_UPDATE_SERVICES See "Customer FM Standard Opcodes" in *BRM Opcode Guide* for more information. • PCM_OP_CUST_POL_GET_PLANS • PCM_OP_CUST_POL_GET_DEALS • PCM_OP_CUST_POL_GET_PRODUCTS • PCM_OP_CUST_POL_READ_PLAN See "Customer FM Policy Opcodes" in *BRM Opcode Guide* for more information.
BRMINVServices_v2	Defines the invoicing web service, which includes the following opcode: • PCM_OP_INV_VIEW_INVOICE **Important:** You must configure your client application to convert the invoice data received from the PCM_OP_INV_VIEW_INVOICE opcode into the appropriate format. See "About Invoicing Output XML Data" in *BRM JCA Resource Adapter*. See "Invoicing FM Standard Opcodes" in *BRM Opcode Guide* for more information.
BRMPYMTServices_v2	Defines the payment web service, which includes the following opcode: • PCM_OP_PYMT_COLLECT See "Payment FM Standard Opcodes" in *BRM Opcode Guide* for more information.
BRMREADServices_v2	Defines the read web service, which includes the following opcodes: • PCM_OP_READ_FLDS • PCM_OP_READ_OBJ • PCM_OP_SEARCH • PCM_OP_TEST_LOOPBACK See "LDAP Base Opcodes" in *BRM Opcode Guide* for more information.

Table 1-2 (Cont.) Web Services Included in Web Services Manager that Take the Payload as an XML Element

Web Service Name	Description
BRMSUBSCRIPTIONServices_v2	Defines the subscription web service, which includes the following opcodes: • PCM_OP_SUBSCRIPTION_CANCEL_DEAL • PCM_OP_SUBSCRIPTION_CANCEL_PRODUCT • PCM_OP_SUBSCRIPTION_CANCEL_DISCOUNT • PCM_OP_SUBSCRIPTION_CANCEL_SUBSCRIPTION • PCM_OP_SUBSCRIPTION_CHANGE_DEAL • PCM_OP_SUBSCRIPTION_GET_HISTORY • PCM_OP_SUBSCRIPTION_PURCHASE_DEAL • PCM_OP_SUBSCRIPTION_PURCHASE_FEES • PCM_OP_SUBSCRIPTION_READ_ACCT_PRODUCTS • PCM_OP_SUBSCRIPTION_SERVICE_BALGRP_TRANSFER • PCM_OP_SUBSCRIPTION_SET_BUNDLE • PCM_OP_SUBSCRIPTION_SET_DISCOUNT_STATUS • PCM_OP_SUBSCRIPTION_SET_DISCOUNTINFO • PCM_OP_SUBSCRIPTION_SET_PRODINFO • PCM_OP_SUBSCRIPTION_SET_PRODUCT_STATUS • PCM_OP_SUBSCRIPTION_TRANSFER_SUBSCRIPTION • PCM_OP_SUBSCRIPTION_TRANSITION_DEAL • PCM_OP_SUBSCRIPTION_TRANSITION_PLAN • PCM_OP_SUBSCRIPTION_GET_PURCHASED_OFFERINGS See "Subscription Management FM Standard Opcodes" in *BRM Opcode Guide* for more information.

Configuring BRM to Use PCM_OP_BAL_GET_ECE_BALANCES

Before you can call the PCM_OP_BAL_GET_ECE_BALANCES opcode, you must configure BRM to support the opcode. To do so, add the following entry to your Connection Manager (CM) configuration file (*BRM_home***/sys/cm/pin.conf**):

```
- cm em_group ece PCM_OP_BAL_GET_ECE_BALANCES
```

Stop and restart the CM for the changes to take effect.

About Testing Web Services Manager

You can develop custom applications that interact with BRM through Web Services Manager. Use a SOAP development environment that supports importing WSDL files (for example, SoapUI) to develop and test your custom web service applications. SOAP development applications may have minor differences in product configuration. Consult your SOAP development application documentation for configuration information.

In general, do the following to develop and test your web services applications:

1. Download and install a SOAP development application.
2. Configure a new project in your SOAP development application.

3. Write a client application that communicates with web services using the SOAP protocol or use the tools integrated in the development application.

   See "" for an example of developing a client application.

4. Import the web service definitions using the WSDL files.

   See "Determining the WSDL URLs for Web Services Manager" for information about generating the WSDL URLs.

   See "About WSDL Files and BRM Opcodes" for more information about the available WSDLs.

5. Run the required commands to set up your application server environment.
6. Configure the properties of the web services operations in your SOAP development environment with valid credentials.
7. Send a web service request to BRM from the SOAP development environment client.

   See "" for an example of this process.

   See "Sending SOAP Requests to BRM Web Services" for information about sending requests to Web Services Manager if OAuth 2.0 is enabled..

8. View the web service response in the SOAP development environment.

Determining the WSDL URLs for Web Services Manager

The WSDL URLs that you use depends on how Web Services Manager is deployed:

- Web Services Manager Is Used in Standalone Mode
- Web Services Manager Is Deployed Into an External Web Server

Web Services Manager Is Used in Standalone Mode

To find the WSDL URLs in standalone mode, go to the following URL:

https://*hostname*:*port*/**metro**

where:

- *hostname* is the name or IP address for Web Services Manager.
- *port* is the port number for Web Services Manager.

Web Services Manager displays the WSDL URLs for each available service.

Web Services Manager Is Deployed Into an External Web Server

To find the WSDL URLs for Web Services Manager when it is deployed into a web server, look in the **infranetwebsvc.war** and **infranetwebsvc.war** files.

For example, in WebLogic Server:

1. Log in to the WebLogic Server Remote Console.
2. Click **Monitoring Tree**, then **Deployments**, then **Application Runtime Data**, then *deploymentName*, where *deploymentName* is the deployment name you chose when deploying the software, for example **WSM**.
3. In the tree on the left, click **Component Runtimes**, then click *adminServerName***I**_**BrmWebServices**, then in the tree click **Servlets**.

   The available web services are displayed in a table.

4. Click the name of the servlet for the web service in the table.

   The settings for the web service are displayed.

5. Scroll to the right until you see the **URL** column. It contains the URL for the web service.

Testing a Web Services Implementation Using a Client Application

To test your web services implementation, you can write a client application that communicates with the web service using the SOAP protocol.

This sample procedure demonstrates how to use the **InfranetBalanceTestClient.java** sample code with the PCM_OP_GET_BALANCES opcode to verify communication between BRM and the web service. The sample uses WebLogic Server, but you can apply the concepts to any type of implementation.

> **Note:**
>
> Ensure that your JAVA_HOME is pointing to Java 21. See *BRM Compatibility Matrix* for information about software compatibility.

To test your implementation:

1. Do one of the following, which sets up the WebLogic Server environment:
   - If WebLogic is installed on a Linux host, run *WebLogic_home***/wlserver/server/bin/setWLSEnv.sh**
   - If WebLogic is installed on a Windows host, run *WebLogic_home***/server/bin/setenv.exe**

   where *WebLogic_home* is the directory in which you installed the WebLogic Server.

2. Create an XML file (for example, **build-stubs.xml**) using the following text:

```
<project name="buildWebservice" default="all">
<property name="buildDir" value="./myapps" />
<property name="jarFiles" value="jars" />
<taskdef name="clientgen"
         classname="weblogic.wsee.tools.anttasks.ClientGenTask"
         classpath="<WEBLOGIC_HOME>/wlserver/server/lib/weblogic.jar"/>

<target name="all" depends="jar" description="builds everything"></target>

<target name="generate-client">
  <clientgen wsdl="http://hostname:port/infranetwebsvc/BRMBalService?WSDL"
             packageName="test_client"
             destDir= "./myapps"/>
</target>

<target name="compile" depends="generate-client" description="compile source files">
  <echo>Compiling adapter files</echo>
  <javac destdir="${buildDir}">
    <src path="${buildDir}"/>
  </javac>
</target>

<target name="jar" depends="compile" description="generate jar file(s)">
  <jar jarfile="clientStub.jar" basedir="${buildDir}">
    <exclude name="**/*.java"/>
```

```
  </jar>
</target>

<target name="clean" description="remove files created by target prepare">
  <delete dir="${buildDir}"/>
</target>
</project>
```

This XML file uses the WebLogic Server clientgen task to automatically generate a utility library that provides low-level SOAP communication (client stubs).

3. Run the following command, which creates the client stubs:

```
ant -file build-stubs.xml
```

This process generates the **clientstubs.jar** file, which contains stubs used by the client.

4. The test client code (*source_home*/**InfranetBalanceTestClient.java**, where *source_home* is the directory where your source code files are stored) then creates an flist, converts it to XML, and calls the PCM_OP_GET_BALANCES opcode.

The following is a sample **InfranetBalanceTestClient.java** file:

```
import java.io.IOException;
import test_client.*;  // corresponds to package name clientgen generated

public class InfranetBalanceTestClient

  {public static void main(String[] args) {
    try {
        String wsdlUrl = "http://hostname:port/infranetwebsvc/BRMBalService?
WSDL";
            BRMBalService_Service service = new
BRMBalService_Service_Impl( wsdlUrl );
            BRMBalService_PortType port = service.getBRMBalService_Ptt();

            String XMLInput="<?xml version=\"1.0\" encoding=\"UTF-80\"?>
<PCM_OP_BAL_GET_BALANCES_inputFlist> <PIN_FLD_POID>0.0.0.1 /account 1 0</
PIN_FLD_POID> </PCM_OP_BAL_GET_BALANCES_inputFlist>";
            System.out.println("Input: \n" + XMLInput);

            // Invoke the web method
            String result = port.pcmOpBalGetBalances(0, XMLInput);

            System.out.println("result: \n"+ result);
    } catch (Exception ex) {
            ex.printStackTrace();
    }
  }
}
```

5. Create another XML file (for example, **build_client.xml**) using the following text:

> **Note:**
>
> Replace the paths for the JAR files as required.

```
<project name="test_client" default="all">

  <target name="all" depends="run"/>
```

```
  <path id="classpath">
    <pathelement path="clientStub.jar"/>
    <pathelement path="./classes"/>
    <pathelement path="<WEBLOGIC_HOME>/wlserver/server/lib/weblogic.jar"/>
  </path>

  <target name="compile">
    <mkdir dir="classes"/>
    <javac srcdir="src"
           destdir="classes"
           classpathref="classpath"/>
  </target>
  <target name="run" depends="compile">
    <java classname="InfranetBalanceTestClient"
          fork="yes"
          classpathref="classpath">
    </java>
  </target>
</project>
```

6. Build and run the test with the **build_client.xml** file using regular Ant tasks:

```
ant -file build_client.xml
```

Example of Reading an Account Object in BRM Using Web Services

This section provides an example of reading an account object using web services when OAuth 2.0 is not configured.

See "Sending SOAP Requests to BRM Web Services" for information about sending requests to Web Services Manager if OAuth 2.0 is enabled..

To read an account object in BRM using web services, you call the pcmOpReadObj web service API that maps to the PCM_OP_READ_OBJ opcode. The pcmOpReadObj web service API is included in the **BRMReadServices_v2** web service, which contains web service APIs that are related to reading accounts. See "About WSDL Files and BRM Opcodes" for more information about the web services included in the Web Services Manager package.

You use URLs to create SOAP clients for web services. The URL is generated by JAX-WS. See "Determining the WSDL URLs for Web Services Manager" for information about generating the WSDL URLs.

In standalone mode, a sample URL for the **BRMReadServices_v2** web service is:

http://*hostname*:*port*/metro/jaxws/BRMReadServices_v2?wsdl**

When Web Services Manager is deployed into a web server, a sample URL for the **BRMReadServices_v2** web service, identified by the **com.portal.jax.read.BRMReadServicePttImpl_WEBSERVICE** servlet, is:

http://*hostIPAddress*:*port*/BrmWebServices/BRMReadServices_v2?wsdl**

> **Note:**
>
> To call a web service, users are required to authenticate using a valid user name and a password. Users can call only those web services that they are authorized to call.

Sample SOAP Request Input XML File

The following sample shows a SOAP request for the **pcmOpReadObj** web service API:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:bus="http://xmlns.oracle.com/BRM/schemas/BusinessOpcodes">
   <soapenv:Header/>
   <soapenv:Body>
      <bus:pcmOpReadObjRequest>
         <bus:PCM_OP_READ_OBJ_Request>
            <bus:flags>0</bus:flags>
            <bus:PCM_OP_READ_OBJ_inputFlist>
               <bus:POID>0.0.0.1 /account 1 0</bus:POID>
            </bus:PCM_OP_READ_OBJ_inputFlist>
         </bus:PCM_OP_READ_OBJ_Request>
      </bus:pcmOpReadObjRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

Sample SOAP Response Output XML File

The following sample shows a SOAP response message for the **pcmOpReadObj** web service API:

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <pcmOpReadObjResponse xmlns="http://xmlns.oracle.com/BRM/schemas/BusinessOpcodes">
         <brm:AAC_ACCESS xmlns:brm="http://xmlns.oracle.com/BRM/schemas/
BusinessOpcodes"/>
         <brm:AAC_PACKAGE xmlns:brm="http://xmlns.oracle.com/BRM/schemas/
BusinessOpcodes"/>
         <brm:AAC_PROMO_CODE xmlns:brm="http://xmlns.oracle.com/BRM/schemas/
BusinessOpcodes"/>
         <brm:AAC_SERIAL_NUM xmlns:brm="http://xmlns.oracle.com/BRM/schemas/
BusinessOpcodes"/>
         <brm:AAC_SOURCE xmlns:brm="http://xmlns.oracle.com/BRM/schemas/
BusinessOpcodes"/>
         <brm:AAC_VENDOR xmlns:brm="http://xmlns.oracle.com/BRM/schemas/
BusinessOpcodes"/>
         <brm:ACCESS_CODE1 xmlns:brm="http://xmlns.oracle.com/BRM/schemas/
BusinessOpcodes"/>
         <brm:ACCESS_CODE2 xmlns:brm="http://xmlns.oracle.com/BRM/schemas/
BusinessOpcodes"/>
         <brm:ACCOUNT_NO xmlns:brm="http://xmlns.oracle.com/BRM/schemas/
BusinessOpcodes">ROOT</brm:ACCOUNT_NO>
         <brm:ACCOUNT_TAG xmlns:brm="http://xmlns.oracle.com/BRM/schemas/
BusinessOpcodes"/>
         <brm:ACCOUNT_TYPE xmlns:brm="http://xmlns.oracle.com/BRM/schemas/
BusinessOpcodes">2</brm:ACCOUNT_TYPE>
         <brm:ATTRIBUTE xmlns:brm="http://xmlns.oracle.com/BRM/schemas/
BusinessOpcodes">0</brm:ATTRIBUTE>
         <brm:BAL_GRP_OBJ xmlns:brm="http://xmlns.oracle.com/BRM/schemas/
BusinessOpcodes">0.0.0.1 /balance_group 1 0</brm:BAL_GRP_OBJ>
         <brm:BRAND_OBJ xmlns:brm="http://xmlns.oracle.com/BRM/schemas/
BusinessOpcodes">0.0.0.1 /account 1 0</brm:BRAND_OBJ>
         <brm:BUSINESS_TYPE xmlns:brm="http://xmlns.oracle.com/BRM/schemas/
BusinessOpcodes">0</brm:BUSINESS_TYPE>
         <brm:CLOSE_WHEN_T xmlns:brm="http://xmlns.oracle.com/BRM/schemas/
BusinessOpcodes">1970-01-01T00:00:00Z</brm:CLOSE_WHEN_T>
         <brm:CONTEXT_INFO xmlns:brm="http://xmlns.oracle.com/BRM/schemas/
```

```
BusinessOpcodes">
            <brm:CORRELATION_ID>1724742721778T49</brm:CORRELATION_ID>
         </brm:CONTEXT_INFO>
         <brm:CREATED_T xmlns:brm="http://xmlns.oracle.com/BRM/schemas/
BusinessOpcodes">2024-05-05T15:26:19Z</brm:CREATED_T>
         <brm:CURRENCY xmlns:brm="http://xmlns.oracle.com/BRM/schemas/
BusinessOpcodes">840</brm:CURRENCY>
         <brm:CURRENCY_SECONDARY xmlns:brm="http://xmlns.oracle.com/BRM/schemas/
BusinessOpcodes">0</brm:CURRENCY_SECONDARY>
         <brm:CUSTOMER_SEGMENT_LIST xmlns:brm="http://xmlns.oracle.com/BRM/schemas/
BusinessOpcodes"/>
         <brm:EFFECTIVE_T xmlns:brm="http://xmlns.oracle.com/BRM/schemas/
BusinessOpcodes">2024-05-05T15:26:19Z</brm:EFFECTIVE_T>
         <brm:GL_SEGMENT xmlns:brm="http://xmlns.oracle.com/BRM/schemas/
BusinessOpcodes"/>
         <brm:GROUP_OBJ xmlns:brm="http://xmlns.oracle.com/BRM/schemas/
BusinessOpcodes">0.0.0.0  0 0</brm:GROUP_OBJ>
         <brm:INCORPORATED_FLAG xmlns:brm="http://xmlns.oracle.com/BRM/schemas/
BusinessOpcodes">0</brm:INCORPORATED_FLAG>
         <brm:INTERNAL_NOTES flags="0x00" offset="0" size="0" xmlns:brm="http://
xmlns.oracle.com/BRM/schemas/BusinessOpcodes"/>
         <brm:ITEM_POID_LIST xmlns:brm="http://xmlns.oracle.com/BRM/schemas/
BusinessOpcodes">0.0.0.1|/item/misc 1 0</brm:ITEM_POID_LIST>
         <brm:LASTSTAT_CMNT xmlns:brm="http://xmlns.oracle.com/BRM/schemas/
BusinessOpcodes"/>
         <brm:LAST_STATUS_T xmlns:brm="http://xmlns.oracle.com/BRM/schemas/
BusinessOpcodes">2024-05-05T15:26:19Z</brm:LAST_STATUS_T>
         <brm:LINEAGE xmlns:brm="http://xmlns.oracle.com/BRM/schemas/BusinessOpcodes">/</
brm:LINEAGE>
         <brm:LOCALE xmlns:brm="http://xmlns.oracle.com/BRM/schemas/BusinessOpcodes"/>
         <brm:MOD_T xmlns:brm="http://xmlns.oracle.com/BRM/schemas/
BusinessOpcodes">2024-05-05T15:26:19Z</brm:MOD_T>
         <brm:NAME xmlns:brm="http://xmlns.oracle.com/BRM/schemas/BusinessOpcodes">Brand
Host</brm:NAME>
         <brm:NAMEINFO elem="1" xmlns:brm="http://xmlns.oracle.com/BRM/schemas/
BusinessOpcodes">
            <brm:ADDRESS/>
            <brm:CANON_COMPANY/>
            <brm:CANON_COUNTRY/>
            <brm:CITY/>
            <brm:COMPANY/>
            <brm:CONTACT_TYPE/>
            <brm:COUNTRY/>
            <brm:EMAIL_ADDR/>
            <brm:FIRST_CANON>system</brm:FIRST_CANON>
            <brm:FIRST_NAME>System</brm:FIRST_NAME>
            <brm:GEOCODE/>
            <brm:LAST_CANON>administrator</brm:LAST_CANON>
            <brm:LAST_NAME>Administrator</brm:LAST_NAME>
            <brm:MIDDLE_CANON/>
            <brm:MIDDLE_NAME/>
            <brm:SALUTATION/>
            <brm:SERVICE_OBJ>0.0.0.0  0 0</brm:SERVICE_OBJ>
            <brm:STATE/>
            <brm:TAXPKG_TYPE>0</brm:TAXPKG_TYPE>
            <brm:TITLE/>
            <brm:ZIP/>
         </brm:NAMEINFO>
         <brm:NEXT_ITEM_POID_LIST xmlns:brm="http://xmlns.oracle.com/BRM/schemas/
BusinessOpcodes"/>
         <brm:OBJECT_CACHE_TYPE xmlns:brm="http://xmlns.oracle.com/BRM/schemas/
BusinessOpcodes">0</brm:OBJECT_CACHE_TYPE>
```

```
        <brm:POID xmlns:brm="http://xmlns.oracle.com/BRM/schemas/
BusinessOpcodes">0.0.0.1 /account 1 1</brm:POID>
        <brm:READ_ACCESS xmlns:brm="http://xmlns.oracle.com/BRM/schemas/
BusinessOpcodes">L</brm:READ_ACCESS>
        <brm:RESIDENCE_FLAG xmlns:brm="http://xmlns.oracle.com/BRM/schemas/
BusinessOpcodes">0</brm:RESIDENCE_FLAG>
        <brm:STATUS xmlns:brm="http://xmlns.oracle.com/BRM/schemas/
BusinessOpcodes">10100</brm:STATUS>
        <brm:STATUS_FLAGS xmlns:brm="http://xmlns.oracle.com/BRM/schemas/
BusinessOpcodes">0</brm:STATUS_FLAGS>
        <brm:TIMEZONE_ID xmlns:brm="http://xmlns.oracle.com/BRM/schemas/
BusinessOpcodes"/>
        <brm:VAT_CERT xmlns:brm="http://xmlns.oracle.com/BRM/schemas/BusinessOpcodes"/>
        <brm:WRITE_ACCESS xmlns:brm="http://xmlns.oracle.com/BRM/schemas/
BusinessOpcodes">L</brm:WRITE_ACCESS>
      </pcmOpReadObjResponse>
   </S:Body>
</S:Envelope>
```

Sample Java Client

If you want to read an account object using a client application, see the instructions in "Testing a Web Services Implementation Using a Client Application" and use the **InfranetReadTestClient.java** file below in place of the **InfranetBalanceTestClient.java** file in the procedure, adjusting the other steps as appropriate.

```
import java.io.IOException;
import test_client.*;  // corresponds to package name clientgen generated

public class InfranetReadTestClient {
public static void main(String[] args) {
        try {
            String wsdlUrl =
"http://hostname:port/infranetwebsvc/BRMReadService?WSDL";
            BRMReadService_Service service = new BRMReadService_Service_Impl( wsdlUrl );
            BRMReadService_PortType port = service.getBRMReadService_Ptt();

            String XMLInput="<?xml version=\"1.0\" encoding=\"UTF-80\"?>
<PCM_OP_READ_OBJ_inputFlist> <PIN_FLD_POID>0.0.0.1 /account 1 0</PIN_FLD_POID> </
PCM_OP_READ_OBJ_inputFlist>";
            System.out.println("Input: \n" + XMLInput);

            // Invoke the web method
            String result = port.pcmOpReadObj(0, XMLInput);
            System.out.println("result: \n"+ result);
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

Using Metrics and Tracing (Standalone only)

The standalone version of Web Services Manager includes additional tracing and metric reporting capabilities.

More information is contained in the following topics:

- Working with Metrics
- Enabling Tracing

Working with Metrics

The standalone version of Web Services Manager includes additional tracing and metric reporting capabilities.

On cloud native implementations of BRM, the metric collection can be integrated with a Prometheus dashboard.

The following metric types are available:

- **base**: Operating system or Java runtime measurements
- **vendor**: REST.request and other key performance indicator measurements
- **application**: Metrics registered in the application code. The following metrics are available for the Web Services Manager, for all APIs, with successful and failed metrics recorded separately.
  - **http_request_duration_seconds_count**: Contains the number of requests received.
  - **http_request_duration_seconds**: Contains the percentile values (0.5,0.75,0.95,0.98,0.99,0.999) for response times.
  - **http_request_duration_seconds_max**: Contains the maximum response time (in seconds) recorded.
  - **http_request_duration_seconds_sum**: Contains the sum of all the response times recorded, in seconds.

Metrics for individual APIs are included when the API is called, and the API call is listed in the metric. For example:

```
http_request_duration_seconds{api="PCM_OP_READ_OBJ",status="success",quantile="0.75",scope="application"}.
```

This metric indicates that the **PCM_OP_READ_OBJ** API is in the **application** scope, and when it is **successful**, its duration is in the 75th percentile (**0.75**).

To access the metrics, use the following URL:

http://*hostname*:*port*/observe/metrics**

To filter the metrics based on their type, use the following URL:

http://*hostname*:*port*/observe/metrics?scope=***scope***

where *scope* is one of the following: **base**, **vendor**, or **application**.

Enabling Tracing

Tracing provides end-to-end visibility into the Web Services Manager's operations, which helps identify and locate performance issues. You can view tracing information using Zipkin, which is an open-source tracing system. See the Zipkin website: https://zipkin.io/ for more information about Zipkin and its user interface. See "Tracing Guide" in the Helidon documentation: https://helidon.io/docs/v4/se/tracing for information about more tracing configuration options.

When tracing is enabled, the logs generated by the Web Services Manager for each request contain the **trace_id** associated with the request. You can use this to assist in tracing any issues.

To enable and configure tracing:

1. Open the *BRM_home*/**apps/brm_wsm/config/Infranet.properties** file.
2. Set the following entry in the file to **true**:

   ```
   tracing.enabled=true
   ```

3. Set the following entries to the correct values for your environment:

   ```
   tracing.host=hostname
   tracing.port=port
   ```

4. Save and exit the file.
5. Stop and restart the Web Services Manager. See "Running and Stopping Standalone Web Services Manager" for more information.

About Data Masking in Web Services Responses

SOAP output response XML files may contain masked fields as configured by your BRM implementation. Subscriber fields, including payment information and user credentials, may be hidden in responses for securing sensitive subscriber data.

See "Masking Sensitive Customer Data" in *BRM Managing Customers* for more information on configuring data masking.

2

Installing Web Services Manager

Learn how to install and set up Oracle Communications Billing and Revenue Management (BRM) Web Services Manager.

Topics in this document:

Installing Web Services Manager

> **Note:**
>
> If you already installed the product, you must uninstall its features before reinstalling them.

Before you install and configure Web Services Manager:

- Unless you are using the Web Services Manager in standalone mode, you must install a supported, standards-compliant server. See "Supported Servers" for a list of the servers supported by Web Services Manager. See server documentation for more information.
- You must install BRM. See "Installing BRM" in *BRM Installation Guide* for more information.
- You must also increase the heap size used by the Java Virtual Machine (JVM) before running the installation program to avoid "Out of Memory" error messages in the log file. For information, see "Problem: Java Out of Memory Error" in *BRM System Administrator's Guide*.

> **Note:**
>
> Oracle recommends that you install Web Services Manager on the system on which BRM is installed.

To install Web Services Manager, see "Installing Individual BRM Components" in *BRM Installation Guide*. The optional component to select is **Webservice Manager** *version*, where *version* is the same as the version of BRM that is installed.

Supported Servers

If you are deploying Web Services Manager on an external server (instead of using it in standalone mode), Web Services Manager is supported on the following servers:

- Oracle WebLogic Server

- Apache Tomcat server

Uninstalling Web Services Manager

To uninstall Web Services Manager, see "Uninstalling Optional Components" in *BRM Installation Guide*.

3

Deploying and Running Web Services Manager

Learn how to deploy and start the Oracle Communications Billing and Revenue Management (BRM) Web Services Manager.

Topics in this document:

Running and Stopping Standalone Web Services Manager

When you are using the Web Services Manager in standalone mode, you do not need to deploy the software. You can start and stop it using scripts that are provided with the product.

To start standalone Web Services Manager, run the following command from the *BRM_home***/bin** directory:

```
start_brm_wsm
```

When started this way, the application takes into account the values in the *BRM_home***/apps/brm_wsm/config/infranet.properties** file.

To stop standalone Web Services Manager, run the following command from the *BRM_home***/bin** directory:

```
stop_brm_wsm
```

Deploying and Running Web Services Manager on WebLogic Server

You can deploy Web Services Manager on WebLogic Server through the WebLogic Server Remote Console. Depending on the type of payload supported by web services, deploy one of the following files:

- **infranetwebsvc.war**: Includes web services that support the payload as an XML string data type.
- **BrmWebServices.war**: Includes web services that support the payload as an XML element data type.

If you customize web services, regenerate **infranetwebsvc.war** or **BrmWebServices.war** and use the generated version. Otherwise, you should use the default **infranetwebsvc.war** or **BrmWebServices.war** file. For more information about customizing web services, see "Customizing Web Services for WebLogic Server or Tomcat Deployments".

To deploy Web Services Manager on WebLogic Server:

1. Create the WebLogic Server domain. See the discussion about creating a WebLogic domain in *Fusion Middleware Creating Domains Using the Configuration Wizard* for detailed instructions.
2. If you deploy the **BrmWebServices.war** file, set the heap size required to start WebLogic Server:
   a. Open the *WebLogic_home***/user_projects/domains/***domain_Name***/ setDomainEnv.sh** file in a text editor.

      where *WebLogic_home* is the directory in which WebLogic Server is installed, and *domain_name* is the name of the domain you created in step 1.

   b. Add the following entry:

      ```
      USER_MEM_ARGS ="-Xms2048m -Xmx2048m"
      ```

   c. Save and close the file.

   d. Restart WebLogic Server.
3. Do one of the following:

   - If you customized web services:

     a. Extract the *BRM_home***/deploy/web_services/infranetwebsvc.war** or the *BRM_home***/deploy/web_services/BrmWebServices.war** file to *local_dir*.

        where *BRM_home* is the directory in which BRM is installed, and *local_dir* is a directory on the machine on which you installed WebLogic Server.

     b. Copy the **CustomFields.jar** files to the *local_dir***/WEB-INF/lib** directory. See "Setting Up Web Services Manager to Support Custom Opcodes" for more information.

        > **Note:**
        >
        > The JRE version that was used to generate **CustomFields.jar** must be the same or lower than the version of the WebLogic Server JRE.

     c. Open the *BRM_home***/deploy/web_services/Infranet.properties** file in a text editor.

     d. Modify the following entry:

        ```
        infranet.custom.field.package = package
        ```

        where *package* is the name of the package that contains the **CustomOp.java** file. For example: **com.portal.classFiles**.

     e. Add all the custom fields to the **Infranet.properties** file.

     f. Save and close the file.

     g. Copy the *BRM_home***/deploy/web_services/Infranet.properties** file to *local_dir***/ WEB-INF/classes** or to the home directory on the machine on which WebLogic Server is installed.

     h. Regenerate the WAR file by running one of the following commands:

        To regenerate the **infranetwebsvc.war** file:

        ```
        jar -cvf infranetwebsvc.war *
        ```

To regenerate the **BrmWebServices.war** file:

```
jar -cvf BrmWebServices.war *
```

- If you did not customize web services:

  a. Extract the *BRM_home*/**deploy/web_services/infranetwebsvc.war** or the *BRM_home*/**deploy/web_services/BrmWebServices.war** file to *local_dir*.

  b. Copy the *BRM_home*/**deploy/web_services/Infranet.properties** file to *local_dir*/**WEB-INF/classes** directory or to the home directory on the machine on which WebLogic Server is installed.

  c. Regenerate the WAR file by running one of the following commands:

  To regenerate the **infranetwebsvc.war** file:

  ```
  jar -cvf infranetwebsvc.war *
  ```

  To regenerate the **BrmWebServices.war** file:

  ```
  jar -cvf BrmWebServices.war *
  ```

4. Log in to WebLogic Server Remote Console.
5. Click **Edit Tree**, then **Deployments**, then **App Deployments**.

   The existing deployments are displayed in a table.
6. Click **New**.
7. Enter the name for the deployment in the **Name** field, move your administration server name in the **Targets** list to the **Chosen** area, and turn off **Upload**.
8. Click Choose File next to the **Source** field and browse to the **infranetwebsvc.war** or **BrmWebServices.war** file.
9. Click **Create**.

   The deployed application is added to the list.
10. Click the name of your new deployment to view and configure any other deployment options.

To start Web Services Manager for web services:

1. Click **Monitoring Tree**, then **Deployments**, then **Application Management**.

   A table containing the deployments is displayed.
2. Select the box next to the deployments you created, such as **infranetwebsvc** or **WSM**, and click **Start**.

Deploying and Running Web Services Manager on Tomcat Server

You can deploy Web Services Manager on Apache Tomcat Server through the Tomcat Web Application Manager. See *BRM Compatibility Matrix* to ensure you are using the supported version of Apache Tomcat.

Depending on the type of payload supported by web services, deploy one of the following files:

- **infranetwebsvc.war**: Includes the web services that support the payload as an XML string data type. See "Deploying and Running infranetwebsvc.war".

- **BrmWebServices.war**: Includes the web services that support the payload as an XML element data type. See "Deploying and Running BrmWebServices.war".

If you customize web services, regenerate **infranetwebsvc.war** or **BrmWebServices.war** and use the generated version. Otherwise, you should use the default **infranetwebsvc.war** or **BrmWebServices.war** file. For more information about customizing web services, see "Customizing Web Services for WebLogic Server or Tomcat Deployments".

Deploying and Running infranetwebsvc.war

To deploy Web Services Manager for web services that support the payload as an XML string data type on Tomcat server:

1. BRM Web Services Manager uses earlier versions of Java libraries that Tomcat no longer supports. To use BRM Web Services Manager with Tomcat, you must convert the libraries to work with Tomcat:
   a. Download version 1.0.0 of the Eclipse Transformer to the *BRM_home***/deploy/web_services** directory.
   b. Extract the contents of the distribution JAR file using the following command:

      ```
      jar xf org.eclipse.transformer.cli-1.0.0-distribution.jar
      ```

   c. Convert the .war file to use Tomcat's version of the libraries using the following command:

      ```
      java -jar org.eclipse.transformer.cli-1.0.0.jar infranetwebsvc.war
      infranetwebsvc-jakarta.war
      ```

2. Create the Tomcat server domain.

   See the Tomcat documentation for detailed instructions.

3. Download the JAX-WS Reference Implementation (RI) library, version 4.0.3, from.
4. Extract the **jaxws-ri-4.0.3.zip** file and copy the following files to *Tomcat_home***/lib**, where *Tomcat_home* is the directory in which the Tomcat server is installed:
   - **gmbal-api-only.jar**
   - **ha-api.jar**
   - **jakarta.activation-api.jar**
   - **jakarta.xml.bind-api.jar**
   - **jakarta.xml.soap-api.jar**
   - **jakarta.xml.ws-api.jar**
   - **jaxb-core.jar**
   - **jaxb-impl.jar**
   - **jaxb-jxc.jar**
   - **jaxb-xjc.jar**
   - **jaxws-rt.jar**
   - **jaxws-tools.jar**
   - **management-api.jar**
   - **saaj-impl.jar**
   - **stax-ex.jar**

- **streambuffer.jar**

5. Download **xalan-2.7.0.jar** file and copy it file to *Tomcat_home***/lib**.
6. In the **infranetwebsvc-jakarta.war/WEB-INF/web.xml** file, uncomment the servlet-to-URL mapping and save and close the file.
7. Log in to the Tomcat Web Application Manager.
8. In the **War file to deploy** section, click **Browse**.
9. Select the **infranetwebsvc-jakarta.war** file and click **Deploy**.

   Tomcat Web Application Manager displays the deployed application in the **Applications** list.

For more information about the BRM web services included in Web Services Manager that take the payload as an XML string data type, see "Using Web Services".

Deploying and Running BrmWebServices.war

To deploy Web Services Manager for web services that support the payload as an XML element data type on Tomcat server:

1. BRM Web Services Manager uses earlier versions of Java libraries that Tomcat no longer supports. To use BRM Web Services Manager with Tomcat, you must convert the libraries to work with Tomcat:
   a. Download version 1.0.0 of the Eclipse Transformer to the *BRM_home***/deploy/web_services** directory.
   b. Extract the contents of the distribution JAR file using the following command:

      ```
      jar xf org.eclipse.transformer.cli-1.0.0-distribution.jar
      ```

   c. Convert the **.war** file to use Tomcat's version of the libraries using the following command:

      ```
      java -jar org.eclipse.transformer.cli-1.0.0.jar BrmWebServices.war
      BrmWebServices-jakarta.war
      ```

2. Create the Tomcat server domain.

   See the Tomcat documentation for detailed instructions.
3. Download the JAX-WS Reference Implementation (RI) library, version 4.0.3.
4. Extract the **jaxws-ri-4.0.3.zip** file and copy the following files to *Tomcat_home***/lib**, where *Tomcat_home* is the directory in which the Tomcat server is installed:
   - **gmbal-api-only.jar**
   - **ha-api.jar**
   - **jakarta.activation-api.jar**
   - **jakarta.xml.bind-api.jar**
   - **jakarta.xml.soap-api.jar**
   - **jakarta.xml.ws-api.jar**
   - **jaxb-core.jar**
   - **jaxb-impl.jar**
   - **jaxb-jxc.jar**
   - **jaxb-xjc.jar**

- **jaxws-rt.jar**
- **jaxws-tools.jar**
- **management-api.jar**
- **saaj-impl.jar**
- **stax-ex.jar**
- **streambuffer.jar**

5. Download **xalan-2.7.0.jar** file and copy it file to *Tomcat_home***/lib**.
6. In the **BrmWebServices-jakarta.war/WEB-INF/web.xml** file, uncomment the servlet-to-URL mapping and save and close the file.
7. Log in to the Tomcat Web Application Manager.
8. In the **War file to deploy** section, click **Browse**.
9. Select the **BrmWebServices-jakarta.war** file and click **Deploy**.

   Tomcat Web Application Manager displays the deployed application in the **Applications** list.
10. In the Applications list, click the **/BrmWebServices** link.
11. The Tomcat Web Application Manager displays an HTTP and an HTTPS URL for the BRM web services.

    A sample URL for the **BRMCUSTServices_v2** web service is as follows:

    http://*hostname*:*port*/BrmWebServices/BRMCUSTServices_v2?wsdl**

    where:

    - *hostname* is the domain IP address of the application server on which Web Services Manager is deployed.
    - *port* is the domain port number of the application server on which Web Services Manager is deployed.

      Web Services Manager displays the WSDL URLs for each available service.

For more information about the BRM web services included in Web Services Manager that take the payload as an XML element data type, see "Using Web Services".

4

Configuring Web Services Manager

Learn how to configure Oracle Communications Billing and Revenue Management (BRM) Web Services Manager by connecting the deployed application to the BRM system and configuring security, authorization, and Java logging for the deployed application.

Topics in this document:

Validating Input and Output XML Data

Web Services Manager validates the input and output XML by comparing the XML fields and values against the opcode XML schema.

The opcode specifications, schemas, and WSDL files are packaged along with Web Services Manager. The package includes the **opspec.xsd** file and the **pin_opspec_to_schema** utility. Use the **opspec.xsd** file to write opcode specifications for custom opcodes that need to be exposed as a web service. Use the **pin_opspec_to_schema** utility to generate the schema files from the opcode specification files.

For more information, see the following topics:

Validating Input and Output XML Data for a Standalone Server

To configure Web Services Manager to validate the input and output XML against the target opcode XML schema on a standalone server:

1. Open the *BRM_home***/apps/brm_wsm/config/Infranet.properties** file.
2. Set the following entries in the file to **true**:

   ```
   webservices.input.validation.enabled=true
   webservices.output.validation.enabled=true
   ```

3. (Optional) Set the following entries in the file to **true** if you want the system to log the error instead of failing the request:

   ```
   webservices.soap.input.validation.reportonly=false
   webservices.soap.output.validation.reportonly=false
   ```

4. Set the following entry to the correct location of your **.xsd** files:

   ```
   webservices.schema.location=fileLocation
   ```

where *fileLocation* is a directory with appropriate permissions for Web Services Manager. The default is **${PIN_HOME}/apps/brm_wsm/schemas**.

5. Save and exit the file.
6. Stop and restart the Web Services Manager. See "Running and Stopping Standalone Web Services Manager" for more information.

Validating Input and Output XML Data for WebLogic Server or Tomcat

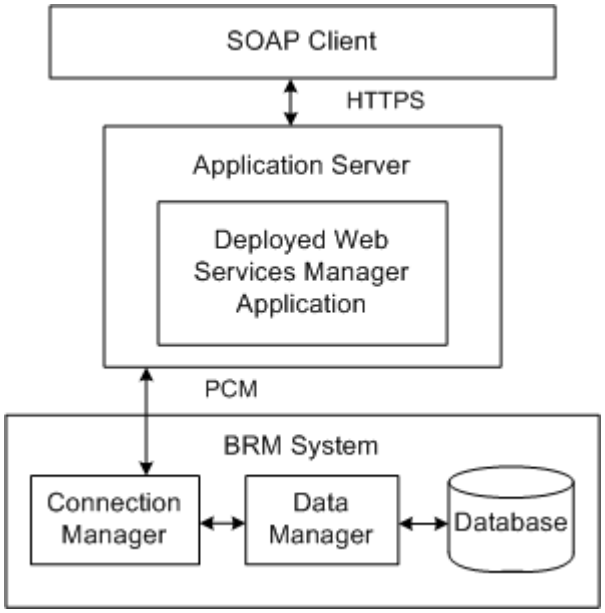
To configure Web Services Manager to validate the input and output XML against the target opcode XML schema when using WebLogic Server or Tomcat:

1. Open the *local_dir***/WEB-INF/classes/Infranet.properties** file.
2. Add the following entries to the file:
   - **webservices.input.validation.enabled=true**
   - **webservices.output.validation.enabled=true**
3. Do one of the following:
   - If you are using WebLogic Server, copy the schema files packaged as a part of Web Services Manager installation from the *BRM_home***/deploy/web_services/schemas** directory to the *local_dir***/common/lib** directory.
   - If you are using any supported server, copy the schema files from the *BRM_home***/deploy/web_services/schemas** directory to the *local_dir***/WEB-INF/classes** directory.

About Connecting Web Services Manager to the BRM System

Web Services Manager connects to the BRM system through a BRM Connection Manager (CM). Figure 4-1 shows how BRM and the SOAP client communicate with the deployed application. Web Services Manager translates Portal Communication Module (PCM) communications sent from a CM in the BRM system into SOAP requests sent to the SOAP client over HTTP. Web Services Manager translates SOAP responses sent from the SOAP client over HTTP into PCM communications that are returned to the CM.

Figure 4-1 Architecture of Web Services Manager in the BRM System



Connecting Web Services Manager to the BRM System

If you customized web services, use the custom **infranetwebsvc.war** or **BrmWebServices.war** file. Otherwise, you should use the default **infranetwebsvc.war** or **BrmWebServices.war** file. For more information about customizing web services, see "Customizing Web Services for a Standalone Server" or "Customizing Web Services for WebLogic Server or Tomcat Deployments".

To connect Web Services Manager to the BRM system:

1. On your application server, copy the *BRM_home***/deploy/web_services/Infranet.properties** file to one of the following:
   - *local_dir***/WEB-INF/classes** directory, where *local_dir* is a directory on the machine on which you installed your application server.

     > **Note:**
     >
     > If you copy the **Infranet.properties** file to the *local_dir***/WEB-INF/classes** directory, extract the **infranetwebsvc.war** file or **BrmWebServices.war** file to a local directory (*local_dir*) on the system on which your application server is installed.

   - The home directory on the machine on which you installed your application server.
2. Open the **Infranet.properties** file in a text editor.
3. If your BRM server and Web Services Manager instances are running on the same server, update these parameters:

```
infranet.connection=pcp://root.0.0.0.1:password@ipAddress:port/0.0.0.1/service/
admin_client 1
infranet.login.type=1
```

where:

- *password* is the password for the BRM server.
- *ipAddress* is the IP address of the system on which BRM is installed.
- *port* is the port number used by the application server on which BRM is installed.

4. If your BRM server is running on a different server than Web Services Manager, comment out the **infranet.connection** parameter and *add* the **infranet.wallet.location** parameter:

```
#infranet.connection=
infranet.wallet.location=wallet_location
```

where *wallet_location* is the PCP connection to your BRM server with the path to your client Oracle wallet. For example: **pcp://root.0.0.0.1:***password@ipAddress***:***port***/0.0.0.1/service/admin_client/scratch/pin00/WALLET**.

5. If SSL is enabled in the CM, update these parameters:

```
infranet.pcp.ssl.enabled=true
infranet.pcp.ssl.wallet.location=wallet_directory
```

where *wallet_directory* is the path to your client Oracle wallet. The client Oracle wallet contains the optional client SSL certificate and the private key, and it contains the Trusted CA certificate.

6. If you added custom opcodes or custom fields for web services, add the enum values of the custom fields.

For example, if you created the **custom_fld_usage_id** field and the enum value for the **custom_fld_usage_id** field is 10001, add this entry:

```
infranet.custom.field.10001=custom_fld_usage_id
```

For information about mapping enum values, see "Creating Custom Fields" in *BRM Developer's Guide*.

7. (Optional) To configure the connection pool parameters, modify the following entries:

```
infranet.connectionpool.minsize=min_connections
infranet.connectionpool.maxsize=max_connections
infranet.connectionpool.timeout=connection_timeout
infranet.connectionpool.maxrequestlistsize=connection_maxrequest
infranet.connectionpool.maxidletime=connection_maxidle
```

where:

- *min_connections* is the minimum number of connections allowed in the pool. The default number is **1**.
- *max_connections* is the maximum number of connections allowed in the pool. The default number is **8**.
- *connection_timeout* is the connection pool timeout in milliseconds. The default value is **30000 milliseconds**.
- *connection_maxrequest* is the maximum number of connection requests the connection pool can queue before returning an error. The default number is **50**.
- *connection_maxidle* is the time in milliseconds that an idle (unused) connection remains in the connection pool before it is removed. The default value is **10000 milliseconds**.

8. (Optional) To configure logging for Web Services Manager if it is deployed into a web server, modify the following entry:

```
webservices.log.enabled=log_value
```

where *log_value* is one of the following:

- **true** enables logging. This option saves and displays the log files as standard output in the application server console.
- **false** disables logging. This option saves the log files in the **/domain/logs/BRMWebSvcMgr.log** file. Configure the *BRM_home***/deploy/web_services/lib/weblogic_ws_startup.jar** file to use this option.

9. Save and close the file.
10. (Optional) To configure logging if you are using Web Services Manager in standalone mode, set the appropriate parameters in the *BRM_home***/apps/brm_wsm/config/logging.properties** file.
11. If you are working in the *local_dir***/WEB-INF/classes** directory, regenerate the WAR file by running one of the following commands:
    - To regenerate the **infranetwebsvc.war** file:

      ```
      jar -cvf infranetwebsvc.war *
      ```

    - To regenerate the **BrmWebServices.war** file:

      ```
      jar -cvf BrmWebServices.war *
      ```

12. Deploy the regenerated **infranetwebsvc.war** or **BrmWebServices.war** file on the server. See "Deploying and Running Web Services Manager".

Connecting to a Different Instance of BRM

If you customized web services, use the custom **infranetwebsvc.war** or **BrmWebServices.war** file. Otherwise, you should use the default **infranetwebsvc.war** or **BrmWebServices.war** file. For more information about customizing web services, see "Customizing Web Services for WebLogic Server or Tomcat Deployments".

To change the instance of BRM to which Web Services Manager connects:

1. On your application server, copy the *BRM_home***/deploy/web_services/Infranet.properties** file to one of the following:
    - *local_dir***/WEB-INF/classes** directory, where *local_dir* is a directory on the machine on which you installed your application server.

      > **Note:**
      >
      > If you copy the **Infranet.properties** file to the *local_dir***/WEB-INF/classes** directory, extract the **infranetwebsvc.war** or **BrmWebServices.war** file to a local directory (*local_dir*) on the system on which your application server is installed.

    - The home directory on the machine on which you installed your application server.
2. Open the copied **Infranet.properties** file.
3. If your BRM server and Web Services Manager instances are running on the same server, update these parameters:

```
infranet.connection=pcp://root.0.0.0.1:password@ipAddress:port/0.0.0.1/service/
admin_client 1
infranet.login.type=1
```

where:

- *password* is the password for the BRM server.
- *ipAddress* is the IP address of the system on which BRM is installed.
- *port* is the port number used by the application server on which BRM is installed.

4. If your BRM server is running on different server than Web Services Manager, comment out the **infranet.connection** parameter and *add* the **infranet.wallet.location** parameter:

```
#infranet.connection=
infranet.wallet.location=wallet_location
```

where *wallet_location* is the PCP connection to your BRM server with the path to your client Oracle wallet. For example: **pcp://root.0.0.0.1:***password@ipAddress***:***port***/0.0.0.1/service/admin_client/scratch/pin00/WALLET**.

5. If SSL is enabled in the Connection Manager (CM), locate the following lines and update the parameters if necessary:

```
infranet.pcp.ssl.enabled=true
infranet.pcp.ssl.wallet.location=wallet_directory
```

where *wallet_directory* is the path to your client Oracle wallet. The client Oracle wallet contains the optional client SSL certificate and the private key, and it contains the Trusted CA certificate.

6. If you added custom opcodes or custom fields for web services, add the enum values of the custom fields.

   For example, if you created the **custom_fld_usage_id** custom field and the enum value for the **custom_fld_usage_id** field is 10001, add the following entry:

```
infranet.custom.field.10001=custom_fld_usage_id
```

   For information about mapping enum values, see "Creating Custom Fields" in *BRM Developer's Guide*.

7. (Optional) To configure the connection pool parameters, modify the following entries:

```
infranet.connectionpool.minsize=min_connections
infranet.connectionpool.maxsize=max_connections
infranet.connectionpool.timeout=connection_timeout
```

   where:

   - *min_connections* is the minimum number of connections allowed in the pool. The default number is **1**.
   - *max_connections* is the maximum number of connections allowed in the pool.
   - *connection_timeout* is the connection pool timeout in milliseconds.

8. (Optional) To configure logging for Web Services Manager, modify the following entry:

```
webservices.log.enabled=log_value
```

   where *log_value* is one of the following:

   - **true** enables logging. This option saves and displays the log files as standard output in the application server console.

- **false** disables logging. This option saves the log files in the **/domain/logs/BRMWebSvcMgr.log** file. Configure the *BRM_home***/deploy/web_services/lib/weblogic_ws_startup.jar** file to use this option.

9. (Optional) To configure searching in BRM using the PCM_OP_SEARCH opcode, restrict the PCM_OP_SEARCH opcode to pre-defined search templates by modifying the following entry:

   ```
   allowed.search.template.ids=template_id
   ```

   where *template_id* is the template ID of the search template that you want the PCM_OP_SEARCH opcode to use for searching. Use a comma (,) to separate multiple template IDs. If you do not want to restrict the PCM_OP_SEARCH opcode to any pre-defined search templates, set *template_id* to **None**.

   For a list of template IDs, connect to the BRM database and check the list of POIDS and the respective templates in the SEARCH_T table in the BRM database. For more information, see "Searching for Objects in the BRM Database" in *BRM Developer's Guide*.

10. If you added custom opcodes or custom fields for web services, add the enum values of the custom fields. For information about mapping enum values, see "Creating Custom Fields" in *BRM Developer's Guide*.

    For example, if you created the **custom_fld_usage_id** field and the enum value for the **custom_fld_usage_id** field is 10001, add the following entry:

    ```
    infranet.custom.field.10001=custom_fld_usage_id
    ```

11. Save and close the file.

12. If you are working in the *local_dir***/WEB-INF/classes** directory, regenerate the WAR file by running one of the following commands:

    - To regenerate the **infranetwebsvc.war** file:

      ```
      jar -cvf infranetwebsvc.war *
      ```

    - To regenerate the **BrmWebServices.war** file:

      ```
      jar -cvf BrmWebServices.war *
      ```

13. Deploy the regenerated **infranetwebsvc.war** or **BrmWebServices.war** file on the server. See "Deploying and Running Web Services Manager".

Configuring Security for Web Services Manager

By default, secure sockets layer (SSL) security for Web Services Manager is enabled. If you disabled SSL during the BRM server installation, you can enable SSL in Web Services Manager by configuring security parameters and enabling the SSL security feature in the application server on which Web Services Manager is deployed.

For more information, see the following topics:

- Configuring Security for Standalone Web Services Manager
- Configuring Security for Web Services Manager in WebLogic Server
- Configuring Security for Web Services Manager in Tomcat Server

Configuring Security for Standalone Web Services Manager

To configure security for Web Services Manager in WebLogic Server, do the following:

1. Obtain an SSL certificate and private key and convert them into PKCS12 (**.p12**) or JKS (**.jks**) format.
2. Edit the *BRM_home***/deploy/web_services/Infranet.properties** file.
3. Uncomment the following lines in the file and set them all to the appropriate values for your environment:

```
@HTTPS Socket
server.sockets.https.port=8081
server.sockets.https.host=0.0.0.0
server.sockets.https.tls.enabled=true
server.sockets.https.tls.endpoint-identification-algorithm=NONE
server.sockets.https.tls.client-auth=NONE
server.sockets.https.tls.private-key.keystore.passphrase=ABCD123#
server.sockets.https.tls.private-key.keystore.resource.path=/etc/example/server.p12
server.sockets.https.tls.trust.keystore.trust-store=true
server.sockets.https.tls.trust.keystore.passphrase=ABCD123#
server.sockets.https.tls.trust.keystore.resource.path=/etc/example/server.p12
```

Configuring Security for Web Services Manager in WebLogic Server

Before you configure security for Web Services Manager, ensure that WebLogic Server and Web Services Manager are installed and that Web Services Manager has been deployed on a WebLogic Server domain. See "Installing Web Services Manager" and "Deploying and Running Web Services Manager" for more information.

To configure security for Web Services Manager in WebLogic Server, do the following:

1. Configure authentication for Web Services Manager. See "Configuring Authentication for WebLogic Server".
2. Configure authorization for Web Services Manager. See "Configuring WebLogic Security Policy on BRM Web Services for JAX-WS in WebLogic Server".

Configuring Authentication for WebLogic Server

Before you configure authentication for Web Services Manager, create a user, group, and security realm for Web Services Manager in WebLogic Server. For more information about creating users and groups, see the discussion about users, groups, and security roles in *Fusion Middleware Securing Resources Using Roles and Policies for Oracle WebLogic Server*. For more information about security realms, see the discussion about security realms in WebLogic Server in *Fusion Middleware Securing Oracle WebLogic Server*.

To configure authentication for Web Services Manager in WebLogic Server:

1. Open the *local_dir***/infranetwebsvc.war/WEB-INF/weblogic.xml** file in a text editor, where *local_dir* is a directory on the WebLogic host where you copied the **infranetwebsvc.war** file.
2. Remove the comment from the following lines:

```
# <security-role-assignment>
   # <role-name>brmws</role-name>
   # <externally-defined/>
# </security-role-assignment>
```

3. Save and close the file.
4. Open the *local_dir***/infranetwebsvc.war/WEB-INF/web.xml** file in a text editor.
5. Remove the comment from the following lines:

```
# <security-constraint>
  # <web-resource-collection>
   #  <web-resource-name>restricted web services</web-resource-name>
    # <url-pattern>/*</url-pattern>
    # <http-method>GET</http-method>
    # <http-method>POST</http-method>
  # </web-resource-collection>
 # <auth-constraint>
    # <role-name>brmws</role-name>
  # </auth-constraint>
  # <user-data-constraint>
   # <transport-guarantee>CONFIDENTIAL</transport-guarantee>
  # </user-data-constraint>
# </security-constraint>

# <login-config>
 # <auth-method>BASIC</auth-method>
 # <realm-name>default</realm-name>
# </login-config>
# <security-role>
    # <role-name>brmws</role-name>
# </security-role>
```

6. Save and close the file.
7. Log in to WebLogic Server Remote Console.
8. Click **Edit Tree**, then **Environment**, then **Servers**.

   A table containing the list of servers in the domain is displayed.
9. Select the server for which you want to enable the SSL port.
10. In the **General** subtab, select **SSL Listen Port Enabled**.
11. In the **SSL Listen Port** field, enter a free port number. The default is **7002**. Make a note of the values in the **Listen Port** and **SSL Listen Port** fields.
12. Click **Save**.

    If you use a SOAP development application to generate a web service client and use port numbers other than the default port numbers, the URLs for the web services that take the payload as an XML element show port numbers that do not match the port numbers you configured in WebLogic Server Remote Console. Populate the correct port numbers in the URLs for the WSDL files that are generated dynamically by your SOAP development application by changing the port numbers manually in your SOAP development application request.

Configuring WebLogic Security Policy on BRM Web Services for JAX-WS in WebLogic Server

You define access restrictions for web services in security policies in WebLogic Server.

To configure WebLogic Security Policy on BRM Web Services for JAX-WS in WebLogic Server:

1. Determine the port binding name for each of the endpoints that you intend to secure. For each endpoint, do the following:
   a. Look at the WSDL file for the endpoint. See "Determining the WSDL URLs for Web Services Manager" for information about accessing the WSDL.
   b. In the WSDL file, find the port name. It may be near the end of the file. For example, the following line contains the port name for the BRMReadServices_v2:

```
<port binding="brm:BRMReadService_binding" name="BRMReadService_pt">
```

For this endpoint, the name is **BRMReadService_pt**.

c. Record the port name.

2. Determine which of the policies supplied with WebLogic Server you would like to implement. For example:
   - If you want to use the policy for HTTPS with basic authentication, you could use **Wssp1.2-2007-Https-BasicAuth.xml**.
   - If you want to use the policy for HTTPS without authentication, you could use **Wssp1.2-2007-Https.xml**.
3. Create the *BRM_home*/**apps/deploy/web_services/brm_wsm_ws_policy** and *BRM_home*/**apps/deploy/web_services/brm_wsm_ws_policy/WEB-INF** directories.
4. In the *BRM_home*/**apps/deploy/web_services/brm_wsm_ws_policy/WEB-INF** directory, create a **weblogic-webservices-policy.xml** file in the following format:

```
<webservice-policy-ref xmlns=http://xmlns.oracle.com/weblogic/webservice-policy-ref
    xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
    xsi:schemaLocation=http://xmlns.oracle.com/weblogic/webservice-policy-ref
                       http://xmlns.oracle.com/weblogic/webservice-policy-ref/1.1/
webservice-policy-ref.xsd>
  <port-policy>
      <port-name>portName1</port-name>
      <ws-policy>
          <uri>policy:policyFilename</uri>
          <direction>both</direction>
          <status>enabled</status>
      </ws-policy>
  </port-policy>
  <port-policy>
      <port-name>portName2</port-name>
      <ws-policy>
          <uri>policy:policyFilename</uri>
          <direction>both</direction>
          <status>enabled</status>
      </ws-policy>
  </port-policy>
. . .
  <port-policy>
      <port-name>portNamen</port-name>
      <ws-policy>
          <uri>policy:policyFilename</uri>
          <direction>both</direction>
          <status>enabled</status>
      </ws-policy>
  </port-policy></webservice-policy-ref>
```

where:

- *portName1* is the port name for the first endpoint, for example **BRMReadService_pt**.
- *portName2* is the port name for the second endpoint, for example **BRMCustService_pt**.
- *portNamen* is the port name for the *n*th endpoint, for example **BRMSubscriptionService_pt**.
- *policyFilename* is the name of the WebLogic Server policy file you are using, for example **Wssp1.2-2007-Https-BasicAuth.xml**.

5. If you are using the services in the **infranetwebsvc.war** file, do the following:
   a. Create the following **plan.xml** file and put it in the appropriate deployment directory:

```
<deployment-plan xmlns="http://xmlns.oracle.com/weblogic/deployment-plan"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://
xmlns.oracle.com/weblogic/deployment-plan http://xmlns.oracle.com/weblogic/
deployment-plan/1.0/deployment-plan.xsd" global-variables="false">
  <application-name>deploymentName</application-name>
  <module-override>
    <module-name>infranetwebsvc.war</module-name>
    <module-type>war</module-type>
    <module-descriptor external="true">
      <root-element>webservice-policy-ref</root-element>
      <uri>WEB-INF/weblogic-webservices-policy.xml</uri>
    </module-descriptor>
  </module-override>
  <config-root>policyPath</config-root>
</deployment-plan>
```

      where:

      - *deploymentName* is the name of the deployment in WebLogic that contains **infranetwebsvc.war**.
      - *policyPath* is the path to the **brm_wsm_ws_policy** directory you created above, that is, *BRM_home*/**apps/deploy/web_services/brm_wsm_ws_policy**.

   b. Redeploy the deployment in WebLogic that contains **infranetwebsvc.war**.

6. If you are using the services in the **BrmWebServices.war** file:
   a. Create the following **plan.xml** file and put it in the appropriate deployment directory:

```
<deployment-plan xmlns="http://xmlns.oracle.com/weblogic/deployment-plan"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://
xmlns.oracle.com/weblogic/deployment-plan http://xmlns.oracle.com/weblogic/
deployment-plan/1.0/deployment-plan.xsd" global-variables="false">
  <application-name>deploymentName</application-name>
  <module-override>
    <module-name>BrmWebServices.war</module-name>
    <module-type>war</module-type>
    <module-descriptor external="true">
      <root-element>webservice-policy-ref</root-element>
      <uri>WEB-INF/weblogic-webservices-policy.xml</uri>
    </module-descriptor>
  </module-override>
  <config-root>policyPath</config-root>
</deployment-plan>
```

      where:

      - *deploymentName* is the name of the deployment in WebLogic that contains **BrmWebServices.war**.
      - *policyPath* is the path to the **brm_wsm_ws_policy** directory you created above, that is, *BRM_home*/**apps/deploy/web_services/brm_wsm_ws_policy**.

   b. Redeploy the deployment in WebLogic that contains **BrmWebServices.war**.

7. To verify that the security policy has been added, access the WSDL for the endpoint again, and look for text similar to the following:

```
<wsp:UsingPolicy wssutil:Required="true"/>
<ns0:Policy xmlns:ns0="http://schemas.xmlsoap.org/ws/2004/09/policy"
wssutil:Id="DefaultReliability.xml">
```

```
<ns1:RMAssertion xmlns:ns1="http://schemas.xmlsoap.org/ws/2005/02/rm/policy">
<ns1:InactivityTimeout Milliseconds="600000"/>
<ns1:BaseRetransmissionInterval Milliseconds="3000"/>
<ns1:ExponentialBackoff/>
<ns1:AcknowledgementInterval Milliseconds="200"/>
<ns2:Expires xmlns:ns2="http://www.bea.com/wsrm/policy" Expires="P1D"/>
</ns1:RMAssertion>
</ns0:Policy>
```

If you have enabled SSL, add the following entry in the session-config element of the **BrmWebServices.war/WEB-INF/web.xml** file to enable cookie security:

```
<cookie-config>
    <secure>true</secure>
</cookie-config>
```

Configuring Security for Web Services Manager in Tomcat Server

Before you configure security for Web Services Manager, ensure that Tomcat server and Web Services Manager are installed and that Web Services Manager has been deployed on a Tomcat server domain. See "Installing Web Services Manager" and "Deploying and Running Web Services Manager" for more information.

To configure security for Web Services Manager in Tomcat server, do the following:

1. Configure authentication for Web Services Manager for JAX-WS in Tomcat server. See "Configuring Authentication for Web Services Manager for JAX-WS in Tomcat Server".
2. Enable SSL in Tomcat server. See "Enabling SSL in Tomcat Server".

Configuring Authentication for Web Services Manager for JAX-WS in Tomcat Server

To configure authentication for Web Services Manager for JAX-WS in Tomcat server:

1. Open the *local_dir***/BrmWebServices.war/WEB-INF/web.xml** file in a text editor.
2. Add the following lines:

```
# <security-constraint>
  # <web-resource-collection>
   #  <web-resource-name>restricted web services</web-resource-name>
    # <url-pattern>/*</url-pattern>
    # <http-method>GET</http-method>
    # <http-method>POST</http-method>
  # </web-resource-collection>
 # <auth-constraint>
    # <role-name>brmws</role-name>
  # </auth-constraint>
# <user-data-constraint>
    # <transport-guarantee>CONFIDENTIAL</transport-guarantee>
# </user-data-constraint>
# </security-constraint>

# <login-config>
 # <auth-method>BASIC</auth-method>
# </login-config>
# <security-role>
    # <role-name>brmws</role-name>
# </security-role>
```

3. Save and close the file.
4. Open the *local_dir***/apache-tomcat-***version***/conf/tomcat-users.xml** file in a text editor.

5. Locate the following lines and specify the login details of the user:

```
<role rolename="brmws"/>
<user username="username" password="password" roles="brmws"/>
```

where:

- *username* is the username for accessing web services.
- *password* is the password for accessing web services.

6. Save and close the file.
7. Open the **config/server.xml** file in a text editor.
8. In the **<Engine>** section, add the following class path:

```
<Realm className="org.apache.catalina.realm.MemoryRealm" />
```

9. Save and close the file.
10. Restart the Tomcat server.

Enabling SSL in Tomcat Server

To enable secure communication for Web Services Manager, enable secure sockets layer (SSL) in the Tomcat server domain on which you deploy Web Services Manager.

To enable SSL for Tomcat server:

1. Generate the KeyStore by running the following command:

```
keytool -genkey -alias mykeys -keyalg RSA -keystore mykeystore
```

where:

- *mykeys* is the alias.
- *mykeystore* is the name of the KeyStore.

2. Open the **conf/server.xml** file in a text editor.
3. Uncomment the following lines and specify the path for the KeyStore file:

```
# <Connector port="8443" protocol="org.apache.coyote.http11.Http11NioProtocol"
           # address="IPAddress"
           # maxThreads="150" SSLEnabled="true" scheme="https" secure="true"
           # clientAuth="false" sslProtocol="TLS"
           # keystoreFile="filepath"
           # keystorePass="password"/>
```

where:

- *IPAddress* is the IP address of the machine on which you installed the Apache Tomcat server.
- *filepath* is the KeyStore file path.
- *password* is the password for the KeyStore file.

4. Save and close the file.

Disabling the JarScanner Feature in Tomcat Server

The JarScanner feature in the Tomcat server is used to scan the web application for JAR files. To avoid unnecessary warnings displayed for optional JAR files, disable the JarScanner feature in the Tomcat server.

To disable the JarScanner feature in the Tomcat server:

1. Open the *local_dir*/**apache-tomcat-***version*/**conf/context.xml** in a text editor.
2. Search for the following entry:

```
<JarScanner  scanClassPath="true" scanAllFiles="false" scanAllDirectories="false"></
JarScanner>
```

3. Set the scanClassPath entry to **false**:

```
<JarScanner  scanClassPath="false" scanAllFiles="false" scanAllDirectories="false"></
JarScanner>
```

4. Save and close the file.

Configuring Java Logging for the Application Server

Depending on your configuration, you may wish to change the level of Java logging on the application server. To configure the Java logging level, do the following:

- For WebLogic Server, see "Configuring Java Logging for WebLogic Server" for Web Services Manager-specific configuration. For more information, see the discussion about application logging and WebLogic logging services in *Fusion Middleware Using Logging Services for Application Logging for Oracle WebLogic Server*.
- For Tomcat server, see the discussion about logging in Tomcat in *Tomcat User Guide*.

Configuring Java Logging for WebLogic Server

To configure Java logging in WebLogic Server:

1. Specify the Java Unified Logging (JUL) mechanism. See "Specifying the Java Unified Logging (JUL) Mechanism".
2. Create a startup class. See "Creating a Startup Class".

Specifying the Java Unified Logging (JUL) Mechanism

Specifying the JUL mechanism allows Web Services Manager to use JUL in addition to the WebLogic Server Remote Console logging.

To specify the JUL mechanism:

1. Open the *BRM_home*/**deploy/web_services/Infranet.properties** file in a text editor.
2. Uncomment the following entry:

```
# webservices.log.enabled = true
```

3. Change the value to **false**:

```
webservices.log.enabled = false
```

4. Save and close the file.

Creating a Startup Class

You define a startup class to enable JUL and create log files for the following web service classes:

- **com.portal.webservices.BRMFlistToXML**
- **com.portal.webservices.BRMXMLToFlist**
- **com.portal.webservices.OpcodeCaller**
- **com.portal.webservices.WebServicesUtilities**

To create a startup class:

1. Copy the *BRM_home***/deploy/web_services/weblogic_ws_startup.jar** file to the *domain_name***/lib** directory, where *domain_name* is the WebLogic Server domain in which Web Services Manager is deployed.
2. Log in to WebLogic Server Remote Console.
3. Click **Edit Tree**, then **Environment**, then **Startup Classes**.

   A list of any startup classes is displayed in a table.
4. Click **New**.
5. In the **Name** field, enter **BRMWSLoggerStartUpClass** and click **Create**.
6. In the **Class Name** field, enter **com.portal.webservices.BRMWSLoggerStartUp**.
7. In the **Arguments** field, set the log level. This field sets the log level for all the classes in Web Services Manager:
   - To log problems that require attention from the system administrator, enter **SEVERE**. This is the default.
   - To log the most detailed trace and debug messages, enter **FINEST**.
   - To log highly detailed trace and debug messages, enter **FINER**.
   - To log trace and debug messages for performance monitoring, enter **FINE**.
8. Turn on **Run Before Application Deployments** and **Run Before Application Activations**.
9. Click the **Targets** tab, move your administration server name in the **Targets** list to the **Chosen** area, and click **Save**.
10. Restart the WebLogic Server, which applies changes.
11. Redeploy any existing Web Services Manager deployments. See "Deploying and Running Web Services Manager".

By default, log files are created in the *WebLogic_home***/user_projects/domains/***domain_name***/logs/BRMWebServicesMgrLogs/BRMWebServicesMgr.log** file.

where:

- *WebLogic_home* is the directory in which WebLogic Server is installed.
- *domain_name* is the name of the domain you are configuring.

5

Securing Web Services Manager with OAuth 2.0

Learn how to secure Oracle Communications Billing and Revenue Management (BRM) Web Services Manager with the OAuth 2.0 authorization framework.

Topics in this document:

About the OAuth 2.0 Authorization Framework

Web Services Manager uses the OAuth 2.0 protocol to authenticate a client application's identity and to authorize the client application to access BRM web services. It does this by validating an OAuth access token that is passed in the header of the client's HTTP/HTTPS request to Web Services Manager.

Your client must pass this OAuth access token in the header of every HTTP/HTTPS request sent to Web Services Manager.

Setting Up Web Services Manager with OAuth 2.0

To set up your client application to use OAuth 2.0 authentication to access BRM web services:

1. (For deployments using Tomcat only) Download **commons-io-2.18.0.jar** and copy the file to *Tomcat_home***/lib**.
2. Install the Oracle Access Management software. For the list of supported versions, see "Additional BRM Software Requirements" in *BRM Compatibility Matrix*.

   For information about installing the Oracle Access Management software, see Oracle Fusion Middleware Installing and Configuring Oracle Identity and Access Management.
3. Create an identity domain in Oracle Access Management. See "Creating an OAuth Identity Domain".
4. Create a resource server in your identity domain. See "Creating a Resource Server".
5. Create an OAuth client in your identity domain. See "Creating an OAuth Client".
6. Validate that OAuth 2.0 is set up properly in Web Services Manager. See "Validating Your OAuth Setup".
7. Configure Web Services Manager to protect BRM web services through Oracle Access Management. See "Configuring Standalone Web Services Manager" or "Configuring Web Services Manager for WebLogic Server".

Creating an OAuth Identity Domain

You create an OAuth identity domain to control the authentication and authorization of users who can sign in to Web Services Manager, and what features they can access. You create all artifacts, such as the resource server and OAuth client, under the identity domain.

To create an identity domain, use cURL to send an HTTP/HTTPS request to the Oracle Access Management URL:

```
curl -i -H "Content-Type: application/json" \
-H "Accept: application/json" \
-H "Authorization:Basic encoded_admin" \
-X POST http://oam_adminHost:oam_adminPort/oam/services/rest/ssa/api/v1/
oauthpolicyadmin/oauthidentitydomain \
-d '{"name": "identity_domain", "description": "Description", "tokenSettings":
[ { "tokenType": "ACCESS_TOKEN", "tokenExpiry": 3600 } ] }'
```

where:

- *encoded_admin* is the Base64-encoded format of the client ID and client secret separated by a colon (*client_id*:*client_secret*).
- *oam_adminHost*:*oam_adminPort* is the host name and port for the Oracle Access Management administration server.
- *identity_domain* is the name of the Oracle Access Management identity domain that you want to create.

For more information about the Oracle Access Management endpoint, see "Add a new OAuth Identity Domain" in *REST API for OAuth in Oracle Access Manager*.

Creating a Resource Server

A resource server hosts the protected resources. It must be capable of accepting and responding to resource requests using OAuth access tokens.

To create a resource server, use cURL to send an HTTP/HTTPS request to the Oracle Access Management URL:

```
curl -i -H "Authorization:Basic encoded_admin" \
-H "Content-Type: application/json" \
-H "Accept: application/json" \
-X POST http://oam_admitHost:oam_adminPort/oam/services/rest/ssa/api/v1/
oauthpolicyadmin/application \
-d '{ "name": "resource_server", "idDomain": "identity_domain",
"description": "Description", "scopes":[ { "scopeName":"OAUTH1",
"description":"All Access" } ] }'
```

where *resource_server* is the name of the resource server that you want to create.

For more information about the Oracle Access Management endpoint, see "Add a new Resource Server" in *REST API for OAuth in Oracle Access Manager*.

Creating an OAuth Client

To create an OAuth client, use cURL to send an HTTP/HTTPS request to the Oracle Access Management URL:

```
curl -i -H "Authorization:Basic encoded_admin" \
-H "Content-Type: application/json" \
-H "Accept: application/json" \
-X POST http://oam_adminHost:oam_adminPort/oam/services/rest/ssa/api/v1/
oauthpolicyadmin/client \
-d '{ "secret": "client_secret", "id": "client_id", "name": "client_name",
"scopes": [ "BrmWebServices.OAUTH1" ], \
"clientType": "CONFIDENTIAL_CLIENT", "idDomain": "identity_domain",
"description": "Description", "grantTypes":[ "CLIENT_CREDENTIALS" ],
"defaultScope": "BrmWebServices.OAUTH1", \
"redirectURIs": [ { "url":"http://wsm_host:wsm_port/BrmWebServices",
"isHttps": false } ] }'
```

where:

- *client_id* and *client_secret* are the client ID and client secret.
- *client_name* is the name of the OAuth client that you want to create.
- *wsm_host*:*wsm_port* is the hostname and port number of the Web Services Manager server.

For more information about the Oracle Access Management endpoint, see "Add a new OAuth Client" in *REST API for OAuth in Oracle Access Manager*.

Validating Your OAuth Setup

To validate that Web Services Manager has been successfully secured with OAuth 2.0:

1. Generate an OAuth access token by submitting a POST request to the **Create Access Token Flow** endpoint in the Oracle Access Management OAuth REST API using cURL:

   ```
   curl -i -H 'Authorization: Basic encoded_admin' \
   -H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8" \
   -H "X-OAUTH-IDENTITY-DOMAIN-NAME: identity_domain" \
   --request POST http://oam_managedServerHost:oam_managedServerPort/oauth2/
   rest/token \
   -d 'grant_type=CLIENT_CREDENTIALS&scope=BrmWebServices.OAUTH1'
   ```

   where *oam_managedServerHost* and *oam_managedServerPort* port is the host name and port for the Oracle Access Management server.

   If successful, the response code 200 is returned with the access token and its expiration time in the response payload.

   For more information, see "Create Access Token Flow" in *REST API for OAuth in Oracle Access Manager*.

2. Validate the access token by submitting a GET request to the **Validate Access Token Flow** endpoint in the Oracle Access Management OAuth REST API using cURL:

   ```
   curl -i -H "X-OAUTH-IDENTITY-DOMAIN-NAME: identity_domain" \
   --request GET "http://oam_managedServerHost:oam_managedServerPort/oauth2/
   rest/token/info?access_token=access_token"
   ```

   where *access_token* is the access token returned in step 1.

   If successful, the response code 200 is returned with details about the access token in the response payload.

   For more information, see "Validate Access Token Flow" in *REST API for OAuth in Oracle Access Manager*.

Configuring Standalone Web Services Manager

Configure Web Services Manager to protect BRM web services through Oracle Access Management.

1. Extract the certificate from the OAuth server and put it in a directory to which the Web Services Manager has access.
2. Open the *BRM_home*/**apps/brm_wsm/config/Infranet.properties** file in a text editor and edit the following parameters:

   ```
   infranet.isOAuth=false
   infranet.certificatePath=certPath/certFile
   ```

   where:

   - *certPath* is the directory to which you copied the OAuth certificate.
   - *certFile* is the name of the certificate file.

Configuring Web Services Manager for WebLogic Server

Configure Web Services Manager to protect BRM web services through Oracle Access Management and enable OAuth validation.

To configure Web Services Manager for WebLogic Server:

1. Copy the *BRM_home*/**deploy/web_services/Infranet.properties** file to the *BRM_home*/**apps/brm_wsm/config/** directory.
2. Open the copied **Infranet.properties** file in a text editor.
3. Edit the following parameters:

   ```
   infranet.OAuthOldOAM=false
   infranet.OAuthAccessTokenUrl:http://oam_host:oam_port/oauth2/rest/ token/
   info
   infranet.OAuthDomainName:identity_domain
   ```

4. Save and close the file.
5. Restart Web Services Manager.

   When you restart the WebLogic Server, ensure that the **libportal.so** BRM library is set in LD_LIBRARY_PATH. For JRE on 64-bit environments, rename **libportal64.so** to **libportal.so**.

6. Go to the *BRM_home*/**deploy/web_services** directory and then extract the contents of the **BrmWebServices.war** file to a local directory (*local_dir*):

   ```
   jar -xvf BrmWebServices.war
   ```

7. Open the *local_dir*/**WEB-INF/web.xml** file in a text editor.

8. Uncomment these **filter** and **filter-mapping** tags:

   ```
   <filter>
         <filter-name>OAuthTokenValidationFilter</filter-name>
         <filter-class>com.portal.jax.OAuthTokenValidationFilter</filter-
   class>
   </filter>
   <filter-mapping>
      <filter-name>OAuthTokenValidationFilter</filter-name>
      <servlet-name>BrmWebServices</servlet-name>
      <url-pattern>/BrmWebServices/*</url-pattern>
      <url-pattern>/BRMPricingServices_v2/</url-pattern>
      <url-pattern>/BRMBalServices_v2/</url-pattern>
      <url-pattern>/BRMARServices_v2</url-pattern>
      <url-pattern>/BRMBillServices_v2</url-pattern>
      <url-pattern>/BRMCustServices_v2</url-pattern>
      <url-pattern>/BRMCustcareServices_v2</url-pattern>
      <url-pattern>/BRMInvServices_v2</url-pattern>
      <url-pattern>/BRMPymtServices_v2</url-pattern>
      <url-pattern>/BRMCollectionServices_v2</url-pattern>
      <url-pattern>/BRMReadServices_v2</url-pattern>
      <url-pattern>/BRMActServices_v2</url-pattern>
      <url-pattern>/BRMSubscriptionServices_v2</url-pattern>
      <dispatcher>FORWARD</dispatcher>
      <dispatcher>REQUEST</dispatcher>
   </filter-mapping>
   ```

9. Save and close the file.

10. Regenerate the **BrmWebServices.war** file.

    a. Go to *local_dir* and delete the existing **BRMWebServices.war** file:

       ```
       cd local_dir
       rm BrmWebServices.war
       ```

    b. Create a new **BrmWebServices.war** archive file:

       ```
       jar -cvf BrmWebServices.war .
       ```

Sending SOAP Requests to BRM Web Services

After you have set up OAuth 2.0 authentication in Web Services Manager, you can start submitting SOAP requests to the BRM web services.

To send a request to a BRM web service:

1. Submit a GET request to the BRM web service that you want to use:

```
curl -i -H "X-OAUTH-IDENTITY-DOMAIN-NAME: identity_domain" \
-H "Authorization:Bearer access_token" \
--request GET http://wsmHost:wsmPort/BrmWebServices/webServicesName?wsdl
```

   where:

   - *wsmHost*:*wsmPort* is
     - For the standalone server: The host name and port for Web Services Manager. You can find these values at the top of the **infranet.properties** file.
     - For Web Services Manager deployed in another server (for example, WebLogic Server): the host name and port number for the external server that contains BRM Web Services Manager.
   - *webServicesName* is the name of the web service such as **BRMACTServices_v2**, **BRMCUSTServices_v2**, or **BRMPYMTServices_v2**. For the web service names, see "About WSDL Files and BRM Opcodes".

2. Submit a request to the target SOAP operation, ensuring that you send the OAuth access token in the header request:

```
curl -i -H "Content-Type: text/xml;charset=UTF-8"
--request POST http://wsmHost:wsmPort/BrmWebServices/webServicesName
-d '<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/
envelope/" xmlns:bus="http://xmlns.oracle.com/BRM/schemas/
BusinessOpcodes">
<soapenv:Header>
   <code>access_token</code>
</soapenv:Header>
<soapenv:Body>
   <bus:operationNameRequest>
      <bus:opcode_Request>
         <bus:opcode_inputFlist>
         </bus:opcode_inputFlist>
      </bus:opcode_Request>
   </bus:pcmOpSearchRequest>
</soapenv:Body>
</soapenv:Envelope>'
```

   where:

   - *operationName* is the name of the SOAP operation to call in the web service interface.
   - *opcode* is the name of the BRM opcode to call.

   For example, to search for accounts that have purchased a particular package, you would submit this request to the **pcmOpSearch** operation in the **BRMReadServices_v2** interface:

```
curl -i -H "Content-Type: text/xml;charset=UTF-8"
--request POST http://wsmHost:wsmPort/BrmWebServices/BRMReadServices_v2
-d '<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/
envelope/" xmlns:bus="http://xmlns.oracle.com/BRM/schemas/BusinessOpcodes">
<soapenv:Header>
   <code>access_token</code>
</soapenv:Header>
```

```
<soapenv:Body>
   <bus:pcmOpSearchRequest>
      <bus:PCM_OP_SEARCH_Request>
         <bus:flags>0</bus:flags>
         <bus:PCM_OP_SEARCH_inputFlist>
            <bus:FLAGS>256</bus:FLAGS>
            <bus:ARGS elem="1">
               <bus:POID>0.0.0.1 /plan -1 0</bus:POID>
            </bus:ARGS>
            <bus:ARGS elem="2">
               <bus:ACCOUNT_OBJ>0.0.0.1 /account 1 0</bus:ACCOUNT_OBJ>
            </bus:ARGS>
            <bus:ARGS elem="3">
               <bus:NAME>Plan 1 - Measured Web Access with Discounts</
bus:NAME>
            </bus:ARGS>
            <bus:POID>0.0.0.1 /search/pin 45 0</bus:POID>
            <bus:RESULTS elem="0"></bus:RESULTS>
            <bus:TEMPLATE>select X from /plan where F1 like V1 and F2 = V2
and F3 =V3</bus:TEMPLATE>
         </bus:PCM_OP_SEARCH_inputFlist>
      </bus:PCM_OP_SEARCH_Request>
   </bus:pcmOpSearchRequest>
</soapenv:Body>
</soapenv:Envelope>'
```

6

Customizing Web Services for a Standalone Server

Learn how to customize Oracle Communications Billing and Revenue Management (BRM) Web Services Manager to expose your customized opcodes or support custom web services when running on a standalone server.

Topics in this document:

- Setting Up Web Services Manager to Support Custom Fields in Opcodes
- Setting Up Web Services Manager to Support Unexposed Opcodes for XML-Element Services
- Setting Up Web Services Manager to Support Custom Opcodes

For information about enabling or disabling validation of input and output XML data, see "Validating Input and Output XML Data".

Setting Up Web Services Manager to Support Custom Fields in Opcodes

You can expose custom fields in BRM opcodes that are exposed by default as web services by doing the following:

1. Add the custom field to the opcode in BRM. See "Creating Custom Fields and Storable Classes" in *BRM Developer's Guide* for instructions.
2. If you are using the standalone server, update the appropriate **.xsd** file with the new fields. The location of the schema files is specified in the **webservices.schema.location** parameter in the *BRM_home***/apps/brm_wsm/config/Infranet.properties** file.
3. If you want to control the XML validation, see "Validating Input and Output XML Data" for more information.

Setting Up Web Services Manager to Support Unexposed Opcodes for XML-Element Services

You can expose BRM opcodes that are not exposed by default as web services by following the procedure below.

See "About WSDL Files and BRM Opcodes" for information about the opcodes that are exposed by default.

To enable Web Services Manager to support these opcodes:

1. Generate the XSD file for your system. See "Generating the Schema Files for Your System" for instructions.
2. Copy the generated XSD file to the location specified in **webservices.schema.location** in the *BRM_home***/apps/brm_wsm/config/Infranet.properties** file.

3. Create a WSDL file for the web service. See "Generating WSDL Files for Web Services" in *BRM JCA Resource Adapter*.
4. Generate the **sun-jaxws.xml** file using one of the following endpoints:
   - **/configurations/endpoints/default** (to use the default deployment descriptor)
   - **/configurations/endpoints** (to use a custom deployment descriptor)
5. Update the corresponding <endpoint> tag in the generated **sun-jaxws.xml** file for your implementation so that it points to your customized endpoint:

   ```
   <endpoint name="endpointName"
       implementation="pathToClass"
       url-pattern="urlPattern"
       wsdl="WEB-INF/wsdl/customDir/customWsdl" />
   ```

   where:
   - *endpointName* is the logical name of the endpoint for the web service.
   - *pathToClass* is the fully qualified path to the implementation class.
   - *urlPattern* is the URL pattern to access the web service.
   - *customDir* is a custom directory created to avoid conflicts with the standard Web Services Manager WSDL files. Ensure that your custom WSDL is located in this directory.
   - *customWsdl* is the name of the WSDL file you created in step 3. Ensure that you add the same custom WSDL file to the .jar file later in this procedure.

   For example:

   ```
   <endpoint name="JAXWS_BRMCustomService"

   implementation="com.oracle.communications.brm.wsm.jaxws.services.cust.BRMCustomServicePttImpl"
       url-pattern="/jaxws/BRMCustomService"
       wsdl="WEB-INF/wsdl/customDir/BRMCustomServices_v2.wsdl" />
   ```

6. Create a JAR file that contains your WSDL file and your edited **sun-jaxws.xml** file in the following structure:
   - **WEB-INF/wsdl/***customDir***/***customWsdl*
   - *sunjavawxPath***/sun-jaxws.xml**

   where:
   - *customDir* is the custom folder used to distinguish your custom WSDL from the standard WSDL files.
   - *customWsdl* is the name of your WSDL file.
   - *sunjavawxPath* is the path to the **sun-jaxws.xml** you created during this procedure.
7. Update the Infranet.properties file with the following new values:

   ```
   webservices.descriptor=sunjavawxPath/sun-jaxws.xml
   webservices.loadcustomschema=true
   ```

   where *sunjavawxPath* is the path to the **sun-jaxws.xml** you created during this procedure.
8. Set the BRM_WSM_CLASSPATH_EXT environment variable to the full path of the jar file you created above.

Setting Up Web Services Manager to Support Custom Opcodes

You can expose your custom opcodes as web services by following the procedures below.

See "About WSDL Files and BRM Opcodes" for information about the opcodes that are exposed by default.

See the following sections for details of how to support custom opcodes:

- Supporting Custom Opcodes for XML-Element Services
- Supporting Custom Opcodes for XML-String Services

Supporting Custom Opcodes for XML-Element Services

To enable Web Services Manager to support these opcodes for the web services that expect an XML element payload:

1. Generate the XSD file for your system. See "Generating the Schema Files for Your System" for instructions.
2. Copy the generated XSD file to the location specified in **webservices.schema.location** in the *BRM_home***/apps/brm_wsm/config/Infranet.properties** file.
3. Create a WSDL file for the web service. See "Generating WSDL Files for Web Services" in *BRM JCA Resource Adapter*.
4. Do one of the following:
   - Create the **CustomOp.java** file by entering the following command:

     ```
     parse_custom_ops_fields -L pcmjava -I input -O output -P java_package
     ```

     where:
     - *input* is the header file you create for your custom opcodes and fields.
     - *output* is the memory-mapped file or directory for the output of the script. *output* must be a directory having some correspondence with the Java package. For example, if *java_package* is in **com.portal.classFiles**, *output* must be **f:/mysource/com/portal/classFiles**.
     - *java_package* is the Java package in which to put the generated classes.

     For more information, see "parse_custom_ops_fields" in *BRM Developer's Guide*.
   - Manually create the **CustomOp.java** file.
5. Verify that the **CustomOp.java** file contains the following:
   - The mapping between opcode names and opcode numbers for all the custom opcodes in the file.

     > **Note:**
     >
     > Verify that the mapping includes the full name of each opcode. If any opcode name is truncated, replace the truncated name with the full name.

   - The **opToString** method, which converts opcode numbers to opcode names.

- The **stringToOp** method, which converts opcode names to opcode numbers.

The following is a sample **CustomOp.java** file:

```
public class CustomOp {
   public static final int CUSTOM_OP_ACT_INFO= 100000;
   public static final int CUSTOM_OP_READ_ACT_PRODUCT = 100001;
   public static String opToString( int op ) {           try {
                java.lang.reflect.Field[] flds =
CustomOp.class.getFields();
           for( int i = 0; i < flds.length; i++ ) {
                 try {
                    int val = flds[i].getInt(null);
                    if( val == op ) {
                        return flds[i].getName();
                    }
            } catch( IllegalAccessException e ) { continue;
            } catch( IllegalArgumentException e ) { continue; }
                    }
             } catch( SecurityException e ) {}
              return null;
}
public static int stringToOp( String op ) {
              try {
                    java.lang.reflect.Field[] flds =
CustomOp.class.getFields();
                for( int i = 0; i < flds.length; i++ ) {
                   try {
                       String name = flds[i].getName();
                       if( name.equals(op) ) {
                         return flds[i].getInt(null);
                        }
                    } catch( IllegalAccessException e ) { continue;
}
                       catch( IllegalArgumentException e ) { continue; }
                }
           } catch( SecurityException e ) {}
       return -1;
     }
}
```

6. Compile the **CustomOp.java** file into the **CustomOp.class** file by entering the following command:

   javac -d . *path*/**CustomOp.java**

   For example:

   javac -d . com/portal/classFiles/CustomOp.java

7. Package the **CustomOp.class** file into the **CustomFields.jar** file by entering the following command:

   jar -cvf CustomFields.jar *path*/**CustomOp.class**

   For example:

   jar -cvf CustomFields.jar com/portal/classFiles/CustomOp.class

8. Make the **CustomFields.jar** file available to Web Services Manager by doing the following:

   a. Copy the *path*/**CustomFields.jar** file to the *CustJarLoc* directory, where *path* is the path to the **CustomFields.jar** file (for example, *BRM_home*/**apps/brm_wsm**) and *CustJarLoc* is a directory that the Web Services Manager has permission to access.

b. Open the *BRM_home*/**apps/brm_wsm/config/Infranet.properties** file in a text editor.

c. Add or modify the following entry:

```
infranet.custom.field.package = package
```

where *package* is the name of the package that contains the **CustomOp.java** file; for example, **com.portal.classFiles**.

d. Add all the custom fields to the **Infranet.properties** file.

e. Save and close the file.

f. Add the full path of *CustJarLoc* to the BRM_WSM_CLASSPATH_EXT environment variable.

g. Stop and restart the Web Services Manager. See "Running and Stopping Standalone Web Services Manager" for more information.

9. Generate the web service implementation class for the custom service by doing the following:

a. Create an implementation class for your new web service. Following is a sample implementation class. Ensure that you update the *packageName*, *ServiceName*, *PortName*, *ServiceClassName*, and *BindingSOAPVersion* with your custom values.

```
package packageName;

// Copyright (c) 2024, 2025, Oracle and/or its affiliates.

import java.util.List;
import java.util.logging.Logger;
import java.util.logging.Level;
import java.util.concurrent.ConcurrentHashMap;
import java.time.Duration;
import io.helidon.metrics.api.MeterRegistry;
import io.helidon.metrics.api.Metrics;
import io.helidon.metrics.api.Timer;
import io.helidon.metrics.api.Tag;
import io.helidon.tracing.Span;
import jakarta.annotation.Resource;
import jakarta.jws.HandlerChain;
import jakarta.jws.soap.SOAPBinding;
import jakarta.xml.ws.Provider;
import jakarta.xml.ws.ServiceMode;
import jakarta.xml.ws.BindingType;
import jakarta.xml.ws.WebServiceContext;
import jakarta.xml.ws.WebServiceProvider;
import jakarta.xml.ws.soap.SOAPFaultException;
import jakarta.xml.soap.SOAPMessage;
import jakarta.xml.soap.SOAPException;
import jakarta.xml.soap.SOAPElement;
import jakarta.xml.soap.SOAPFactory;
import jakarta.xml.soap.Detail;
import jakarta.xml.soap.SOAPFault;
import jakarta.xml.soap.SOAPConstants;
import javax.xml.namespace.QName;
import org.w3c.dom.Document;
import com.portal.pcm.DeterminateException;
import com.portal.pcm.EBufException;
import com.oracle.communications.brm.wsm.jaxws.ApiRequestProcessor;
import com.oracle.communications.brm.wsm.utils.WebServicesUtilities;
import com.oracle.communications.brm.wsm.utils.TracerHandler;
import com.oracle.communications.brm.wsm.utils.SOAPFaultResponse;
```

```
import com.oracle.communications.brm.wsm.utils.ApplicationException;

@WebServiceProvider(targetNamespace = "http://xmlns.oracle.com/BRM/schemas/
BusinessOpcodes", serviceName = "ServiceName", portName = "PortName")
@ServiceMode(value = jakarta.xml.ws.Service.Mode.MESSAGE)
@SOAPBinding(style = SOAPBinding.Style.DOCUMENT, parameterStyle =
SOAPBinding.ParameterStyle.BARE)
@BindingType(jakarta.xml.ws.soap.SOAPBinding.BindingSOAPVersion)
@HandlerChain(file = "handler-chain.xml")
public class ServiceClassName implements Provider<SOAPMessage> {
    private static Logger m_logger =
Logger.getLogger(ServiceClassName.class.getName());
    private final MeterRegistry registry = Metrics.globalRegistry();

    private final Tag successTag = Tag.create("status", "success");
    private final Tag failTag = Tag.create("status", "fail");

    private final ConcurrentHashMap<Integer,Timer> successResponseTimerMap = new
ConcurrentHashMap<Integer,Timer>();
    private final ConcurrentHashMap<Integer,Timer> failedResponseTimerMap = new
ConcurrentHashMap<Integer,Timer>();

    @Override
    public SOAPMessage invoke(SOAPMessage request) {
        m_logger.entering(ServiceClassName.class.getName(), "Starting invoke
method execution for request");
        Span span = TracerHandler.startSpan(null,
ServiceClassName.class.getName() + "." + "invoke");
        Exception spanException = null;

        String opcodeName = null, soapVersion = null, soapNamespaceUri = null;
        int opcodeNumber = -1;
        long startTime = System.nanoTime();
        SOAPMessage response = null;
        Timer currentTimer;

        try{
            // Determining SOAP request protocol version (1.1 or 1.2)
            soapNamespaceUri =
request.getSOAPPart().getEnvelope().getNamespaceURI();

            if (SOAPConstants.URI_NS_SOAP_1_1_ENVELOPE.equals(soapNamespaceUri))
{
                soapVersion = SOAPConstants.SOAP_1_1_PROTOCOL;
            }
            else if
(SOAPConstants.URI_NS_SOAP_1_2_ENVELOPE.equals(soapNamespaceUri)) {
                soapVersion = SOAPConstants.SOAP_1_2_PROTOCOL;
            }
            else {
                soapVersion = SOAPConstants.DEFAULT_SOAP_PROTOCOL;
                soapNamespaceUri = SOAPConstants.URI_NS_SOAP_ENVELOPE;
            }
            m_logger.log(Level.FINEST, "SOAP Protocol Version: " + soapVersion +
", SOAP Namespace URI: " + soapNamespaceUri);
            Document reqDoc = request.getSOAPBody().extractContentAsDocument();

            opcodeName = WebServicesUtilities.getOpcodeName(span.context(),
reqDoc);
            opcodeNumber = WebServicesUtilities.getOpcode(span.context(),
opcodeName);
```

```
            m_logger.log(Level.FINEST, "Opcode name obtained from request
payload: " + opcodeName);

            if(!successResponseTimerMap.containsKey(opcodeNumber)){
                m_logger.log(Level.FINEST, "Initializing success & failure
metric timers");

                Tag apiTag = Tag.create("api", opcodeName);

                successResponseTimerMap.put(
                        opcodeNumber,

registry.getOrCreate(Timer.builder("http_request_duration").tags(List.of(apiTag,
successTag)))
                );
                failedResponseTimerMap.put(
                        opcodeNumber,

registry.getOrCreate(Timer.builder("http_request_duration").tags(List.of(apiTag,
failTag)))
                );
            }
            ApiRequestProcessor processor = new ApiRequestProcessor();
            response = processor.processApiRequest(span.context(), reqDoc,
opcodeName, opcodeNumber, soapVersion, soapNamespaceUri);

            currentTimer = successResponseTimerMap.get(opcodeNumber);
            currentTimer.record(Duration.ofNanos((System.nanoTime() -
startTime)));
        } catch (ApplicationException e) {
            spanException = e;
            if (opcodeNumber != -1) {
                currentTimer = failedResponseTimerMap.get(opcodeNumber);
                currentTimer.record(Duration.ofNanos((System.nanoTime() -
startTime)));
            }
            throw SOAPFaultResponse.createErrorResponse(e, soapVersion,
soapNamespaceUri);
        } catch (SOAPException ex) {
            spanException = ex;
            if (opcodeNumber != -1) {
                currentTimer = failedResponseTimerMap.get(opcodeNumber);
                currentTimer.record(Duration.ofNanos((System.nanoTime() -
startTime)));
            }
            throw SOAPFaultResponse.createErrorResponse(
                new ApplicationException(ex,
ApplicationException.SERVER_EXCEPTION), soapVersion, soapNamespaceUri
            );
        } finally {
            TracerHandler.endSpan(span, spanException);
            m_logger.exiting(ServiceClassName.class.getName(), "Completed invoke
method execution for request");
        }
        return response;
    }
}
```

b. Create **BRMCustomServicePttImpl.java** and compile it using the following command:

javac -d . -cp '.:$PIN_HOME/jars/*' *packageName*/**BRMCustomServicePttImpl.java**

where *packageName* is the package name configured in your implementation class.

10. Generate the **sun-jaxws.xml** file using one of the following endpoints:
    - **/configurations/endpoints/default** (to use the default deployment descriptor)
    - **/configurations/endpoints** (to use a custom deployment descriptor)
11. Add the new endpoint in the generated **sun-jaxws.xml** file for your implementation:

```
<endpoint name="endpointName"
    implementation="pathToClass"
    url-pattern="urlPattern"
    wsdl=wsdl="WEB-INF/wsdl/customDir/customWsdl" />
```

    where:
    - *endpointName* is the logical name of the new endpoint for the web service.
    - *pathToClass* is the fully qualified path to the implementation class.
    - *urlPattern* is the URL pattern to access the web service.
    - *customDir* is a custom directory created to avoid conflicts with the standard Web Services Manager WSDL files. Ensure that your custom WSDL is located in this directory.
    - *customWsdl* is the name of the WSDL file you created in step 3. Ensure that you add the same custom WSDL file to the .jar file later in this procedure.

    For example:

```
<endpoint name="JAXWS_BRMCustomService"
    implementation="CustomPackage.BRMCustomServicePttImpl"
    url-pattern="/jaxws/BRMCustomService"
    wsdl=""WEB-INF/wsdl/customDir/BRMCustomServices_v2.wsdl" />
```

12. Create a JAR file that contains your implementation class, its WSDL file, and your edited **sun-jaxws.xml** file in the following structure:
    - **WEB-INF/wsdl/***customDir***/***customWsdl*
    - *sunjavawxPath***/sun-jaxws.xml**
    - *packageName***/BRMCustomServicePttImpl.class**

    where:
    - *customDir* is the custom folder used to distinguish your custom WSDL from the standard WSDL files.
    - *customWsdl* is the name of your WSDL file.
    - *sunjavawxPath* is the path to the **/sun-jaxws.xml** you created during this procedure.
    - *packageName* is the package name configured in your implementation class.
13. Update the **Infranet.properties** file with the following new values:

```
webservices.descriptor=sunjavawxPath/sun-jaxws.xml
webservices.loadcustomschema=true
infranet.custom.field.package=CustomOp_package
```

    where:
    - *sunjavawxPath* is the path to the **/sun-jaxws.xml** you created during this procedure.
    - *CustomOp_package* is the path to the **CustomOp.class** file you created earlier in this procedure.
14. Add the full paths of **CustomFields.jar** file and the JAR file you created in step 12 to the BRM_WSM_CLASSPATH_EXT environment variable.

Supporting Custom Opcodes for XML-String Services

To enable Web Services Manager to support these opcodes for the web services that expect an XML string payload:

1. Do one of the following:
   - Create the **CustomOp.java** file by entering the following command:

     parse_custom_ops_fields -L pcmjava -I *input* **-O** *output* **-P** *java_package*

     where:
     - *input* is the header file you create for your custom opcodes and fields.
     - *output* is the memory-mapped file or directory for the output of the script. *output* must be a directory having some correspondence with the Java package. For example, if *java_package* is in **com.portal.classFiles**, *output* must be **f:/mysource/com/portal/classFiles**.
     - *java_package* is the Java package in which to put the generated classes.

     For more information, see "parse_custom_ops_fields" utility in *BRM Developer's Guide*.
   - Manually create the **CustomOp.java** file.
2. Verify that the **CustomOp.java** file contains the following:
   - The mapping between opcode names and opcode numbers for all the custom opcodes in the file.

     > **Note:**
     >
     > Verify that the mapping includes the full name of each opcode. If any opcode name is truncated, replace the truncated name with the full name.

   - The **opToString** method, which converts opcode numbers to opcode names.
   - The **stringToOp** method, which converts opcode names to opcode numbers.

   The following is a sample **CustomOp.java** file:

```
public class CustomOp {
   public static final int CUSTOM_OP_ACT_INFO= 100000;
   public static final int CUSTOM_OP_READ_ACT_PRODUCT = 100001;
   public static String opToString( int op ) {           try {
                java.lang.reflect.Field[] flds =
CustomOp.class.getFields();
           for( int i = 0; i < flds.length; i++ ) {
                 try {
                    int val = flds[i].getInt(null);
                    if( val == op ) {
                        return flds[i].getName();
                    }
             } catch( IllegalAccessException e ) { continue;
             } catch( IllegalArgumentException e ) { continue; }
                    }
              } catch( SecurityException e ) {}
               return null;
}
```

```
public static int stringToOp( String op ) {
              try {
                    java.lang.reflect.Field[] flds =
CustomOp.class.getFields();
                for( int i = 0; i < flds.length; i++ ) {
                   try {
                       String name = flds[i].getName();
                       if( name.equals(op) ) {
                         return flds[i].getInt(null);
                        }
                    } catch( IllegalAccessException e ) { continue;
}
                       catch( IllegalArgumentException e ) { continue; }
               }
          } catch( SecurityException e ) {}
       return -1;
     }
}
```

3. Compile the **CustomOp.java** file into the **CustomOp.class** file by entering the following command:

   javac -d . *path*/**CustomOp.java**

   For example:

   javac -d . com/portal/classFiles/CustomOp.java

4. Package the **CustomOp.class** file into the **CustomFields.jar** file by entering the following command:

   jar -cvf CustomFields.jar *path*/**CustomOp.class**

   For example:

   jar -cvf CustomFields.jar com/portal/classFiles/CustomOp.class

5. Make the **CustomFields.jar** file available to Web Services Manager by doing the following:

   a. Copy the *path*/**CustomFields.jar** file to the *CustJarLoc* directory, where *path* is the path to the **CustomFields.jar** file (for example, *BRM_home*/**apps/brm_wsm/**) and *CustJarLoc* is a directory that the Web Services Manager has permission to access.

   b. Open the *BRM_home*/**apps/brm_wsm/config/Infranet.properties** file in a text editor.

   c. Add or modify the following entry:

      infranet.custom.field.package = *package*

      where *package* is the name of the package that contains the **CustomOp.java** file; for example, **com.portal.classFiles**.

   d. Add all the custom fields to the *BRM_home*/**apps/brm_wsm/config/Infranet.properties** file.

   e. Save and close the file.

   f. Set the BRM_WSM_CLASSPATH_EXT environment variable to the full path of *CustJarLoc*.

6. Generate the web service implementation class for the custom service by doing the following:

   a. Create an implementation class for your new web service. Following is a sample implementation class. Ensure that you update the *packageName* with your custom value.

```
package packageName;

// Copyright (c) 2025, Oracle and/or its affiliates.

import com.oracle.communications.brm.wsm.utils.ApplicationException;
import com.oracle.communications.brm.wsm.utils.OpcodeCaller;
import com.oracle.communications.brm.wsm.utils.SOAPFaultResponse;

import jakarta.jws.WebMethod;
import jakarta.jws.WebParam;
import jakarta.jws.WebResult;
import jakarta.jws.WebService;
import jakarta.jws.soap.SOAPBinding;
import jakarta.xml.soap.SOAPConstants;
import jakarta.jws.HandlerChain;

/**
 * Class that implements OOB Infranet CUST Web Service.
 * Implementation detail: This class delegates to OpcodeCaller all functionality
 * related to communicating with Infranet and calling opcodes. This allows
 * flexibility in reimplementing the web service, for a different app server,
 * for example. In that case, the new implementation can simply call the
 * OpcodeCaller utility functions.
 *
 * CUSTOM_OP_GENERIC<br/>
 */

@WebService (name = "BRMCustomService",
            targetNamespace = "http://xmlns.oracle.com/BRM/schemas/
BusinessOpcodes/",
            portName = "BRMCustomService_ptt")
@SOAPBinding(style = SOAPBinding.Style.RPC)
@HandlerChain(file = "handler-chain.xml")
public class BRMCustomInfra {
@WebMethod(operationName = "customOpGeneric")
    public  @WebResult(name = "CUSTOM_OP_GENERIC_response") String customGeneric(
        @WebParam(name = "flags")int  flag,
        @WebParam(name = "CUSTOM_OP_GENERIC_request")String inFlist)
    {
        String    CUSTOM_OP_GENERIC_response = null;
        OpcodeCaller oc = new OpcodeCaller();
        try {
             CUSTOM_OP_GENERIC_response =
oc.opcodeWithFlags("CUSTOM_OP_GENERIC", flag, inFlist,"CUSTOM_OP_GENERIC.xsd");
             return CUSTOM_OP_GENERIC_response;
        } catch (ApplicationException ex) {
             throw SOAPFaultResponse.createErrorResponse(ex,
SOAPConstants.SOAP_1_1_PROTOCOL, SOAPConstants.URI_NS_SOAP_1_1_ENVELOPE);
        }
    }
}
```

b. Compile it using the following command:

```
javac -d . -cp '.:$PIN_HOME/jars/*' packageName/BRMCustomInfra.java
```

where *packageName* is the package name configured in your implementation class.

7. Generate the **sun-jaxws.xml** file using one of the following endpoints:
   - **/configurations/endpoints/default** (to use the default deployment descriptor)
   - **/configurations/endpoints** (to use a custom deployment descriptor)

8. Add the new endpoint in the generated **sun-jaxws.xml** file for your implementation:

```
<endpoint name="endpointName"
    implementation="pathToClass"
    url-pattern="urlPattern />
```

where:

- *endpointName* is the logical name of the new endpoint for the web service.
- *pathToClass* is the fully qualified path to the implementation class.
- *urlPattern* is the URL pattern to access the web service.

For example:

```
<endpoint
    name="JAXWS_STR_BRMCustomService"
    implementation="CustomPackage.BRMCustomInfra"
    url-pattern="/jaxws_str/BRMCustomService" />
```

9. Create a JAR file that contains your implementation class and your edited **sun-jaxws.xml** file in the following structure:

- *sunjavawxPath*/**sun-jaxws.xml**
- *packageName*/**BRMCustomInfra.class**

where:

- *sunjavawxPath* is the path to the **sun-jaxws.xml** you created during this procedure.
- *packageName* is the package name configured in your implementation class.

10. Update the Infranet.properties file with the following new values:

```
webservices.descriptor=sunjavawxPath/sun-jaxws.xml
```

where *sunjavawxPath* is the path to the **sun-jaxws.xml** you created during this procedure.

11. Add the full paths of **CustomFields.jar** file and the JAR file you created in step 9 to the BRM_WSM_CLASSPATH_EXT environment variable.

12. Stop and restart the Web Services Manager. See "Running and Stopping Standalone Web Services Manager" for more information.

7

Customizing Web Services for WebLogic Server or Tomcat Deployments

Learn how to customize Oracle Communications Billing and Revenue Management (BRM) Web Services Manager to expose your custom opcodes or support custom web services in an instance of Web Services Manager deployed on Oracle WebLogic Server or Tomcat.

Topics in this document:

For information about enabling or disabling validation of input and output XML data, see "Validating Input and Output XML Data".

Setting Up Web Services Manager to Support Custom Opcodes

To expose custom opcodes as web services, first implement the custom opcode. For more information on custom opcodes, see "Using Custom Opcodes" in *BRM Developer's Guide*. Then, enable Web Services Manager to support custom opcodes as described below.

To enable Web Services Manager to support custom opcodes:

1. Do one of the following:
   - Create the **CustomOp.java** file by entering the following command:

     ```
     parse_custom_ops_fields -L pcmjava -I input -O output -P java_package
     ```

     where:

     – *input* is the header file you create for your custom opcodes and fields.

     – *output* is the memory-mapped file or directory for the output of the script. *output* must be a directory having some correspondence with the Java package. For example, if *java_package* is in **com.portal.classFiles**, *output* must be **f:/mysource/com/portal/classFiles**.

     – *java_package* is the Java package in which to put the generated classes.

     For more information, see the discussion about the **parse_custom_ops_fields** utility in *BRM Developer's Guide*.
   - Manually create the **CustomOp.java** file.
2. Verify that the **CustomOp.java** file contains the following:
   - The opcode-name-to-opcode-number mapping for all the custom opcodes in the file.

> **Note:**
>
> Verify that the mapping includes the full name of each opcode. If any opcode name is truncated, replace the truncated name with the full name.

- The **opToString** method, which converts opcode numbers to opcode names.
- The **stringToOp** method, which converts opcode names to opcode numbers.

The following is a sample **CustomOp.java** file:

```
public class CustomOp {
   public static final int CUSTOM_OP_ACT_INFO= 100000;
   public static final int CUSTOM_OP_READ_ACT_PRODUCT = 100001;
   public static String opToString( int op ) {           try {
                java.lang.reflect.Field[] flds =
CustomOp.class.getFields();
           for( int i = 0; i < flds.length; i++ ) {
                try {
                   int val = flds[i].getInt(null);
                   if( val == op ) {
                       return flds[i].getName();
                   }
            } catch( IllegalAccessException e ) { continue;
            } catch( IllegalArgumentException e ) { continue; }
                   }
             } catch( SecurityException e ) {}
              return null;
}
public static int stringToOp( String op ) {
               try {
                     java.lang.reflect.Field[] flds =
CustomOp.class.getFields();
                  for( int i = 0; i < flds.length; i++ ) {
                     try {
                         String name = flds[i].getName();
                         if( name.equals(op) ) {
                           return flds[i].getInt(null);
                          }
                      } catch( IllegalAccessException e ) { continue;
}
                         catch( IllegalArgumentException e ) { continue; }
                }
          } catch( SecurityException e ) {}
       return -1;
     }
}
```

3. Compile the **CustomOp.java** file into the **CustomOp.class** file by entering the following command:

   javac -d . *path*/**CustomOp.java**

   For example:

   javac -d . com/portal/classFiles/CustomOp.java

4. Package the **CustomOp.class** file into the **CustomFields.jar** file by entering the following command:

> **Note:**
>
> Make sure the JRE version that was used to generate the **CustomFields.jar** file is the same or lower than the version of the WebLogic Server JRE.

```
jar -cvf CustomFields.jar path.CustomOp.class
```

For example:

```
jar cvf CustomFields.jar com.portal.classFiles.CustomOp.class
```

5. Make the **CustomFields.jar** file available to Web Services Manager by doing one of the following:

   - If you have not deployed Web Services Manager, do the following:

     a. Copy the *path*/**CustomFields.jar** file to the *local_dir*/**WEB-INF/lib** directory, where *path* is the path to the **CustomFields.jar** file (for example, **com/portal/classFiles**).

     b. Open the *BRM_home*/**deploy/web_services/Infranet.properties** file in a text editor.

     c. Add or modify the following entry:

        ```
        infranet.custom.field.package = package
        ```

        where *package* is the name of the package that contains the **CustomOp.java** file; for example, **com.portal.classFiles**.

     d. Add all the custom fields to the **Infranet.properties** file.

     e. Save and close the file.

     f. Copy the *BRM_home*/**deploy/web_services/Infranet.properties** file to the *local_dir*/**WEB-INF/classes** directory or the home directory on the machine on which you installed WebLogic Server.

   - If you have deployed Web Services Manager, do the following:

     a. Copy the *path*/**CustomFields.jar** file to the *local_dir*/**WEB-INF/lib** directory.

        where *local_dir* is the directory in which you deployed Web Services Manager on your application server.

     b. Open the *Webservices_deployment_dir*/**WEB-INF/classes/Infranet.properties** file in a text editor.

     c. Add or modify the following entry:

        ```
        infranet.custom.field.package = package
        ```

        where *package* is the name of the package that contains the **CustomOp.java** file; for example, **com.portal.classFiles**.

     d. Add all the custom fields to the **Infranet.properties** file.

     e. Save and close the file.

Creating a Custom Web Service

You can extend Web Services Manager to support custom web services. Before you create a custom web service or customize an existing web service in Web Services Manager, implement your custom opcodes in the BRM system. For more information, see "Creating Custom Fields and Storable Classes" in *BRM Developer's Guide*.

To create a custom web service:

1. If you created an opcode with custom fields for your custom web service, configure BRM to recognize the custom fields. See "Creating Custom Fields and Storable Classes" in *BRM Developer's Guide*.
2. Create a WSDL file for the web service. See "Generating WSDL Files for Web Services" in *BRM JCA Resource Adapter*.

   To create a WSDL file manually, do the following:

   - For web services that support payload as XML string, see the **deploy/web_services/wsdl** sample file and create the WSDL file.
   - For web services that support payload as XML element, see the **deploy/web_services/BrmWebServices.war/WEB-INF/wsdl** sample file and create the WSDL file.
3. Create the XML specifications for your custom opcodes. See "Creating Opcode Specification Schema Files".
4. Generate web service classes for your custom service by doing the following:

   a. Create the following directory structure in a local directory (*local_dir*) on the machine on which your application server is installed.

      ```
      /wsdl
      /src
      /classes
      /jar
      ```

   b. Copy your custom WSDL files and schema (XSD) files into the *local_dir***/wsdl** directory.

   c. Copy the **BrmWebServices.war/WEB-INF/wsdl/BRMWebServiceException.xsd** file into the *local_dir***/wsdl** directory.

   d. Create the **custom_services.xml** as an Ant build file.

      The following is a sample **custom_services.xml** file:

      ```
      <?xml version="1.0"?>
      <project name="Custom BRM WebServices build file" default="all" basedir=".">
      <property name="buildDir"      value="classes"/>
      <property name="srcDir"     value="src"/>
      <property name="wsdlDir"       value="wsdl"/>
      <property name="pinwsgen"      value="pin_wsgen"/>

      <!-- define the classpath -->
      <path id="classpath">
              <pathelement path="${buildDir}"/>
              <pathelement path="jar/web_services.jar"/>
              <pathelement path="jar/webServicesUtils.jar"/>
      </path>

      <!-- create Source files from WSDL and XSDs -->
      <target name="custom_service_gen" description="Create java source files from
      ```

```
wsdl"  >
        <exec executable="BRM_home/deploy/web-services/pin_wsgen/pin_wsgen"
failonerror="true">
            <arg value="-s"/>
            <arg value="src"/>
            <arg value="-d"/>
            <arg value="${buildDir}"/>
            <arg value="-p"/>
            <arg value="com.portal.jax.'yourpackagesubdirname' "/>
            <arg value="${wsdlDir}/'YourCustomServices_v2.wsdl'/>
        </exec>
</target>

<target name="all" depends="custom_service_gen, custom_jar" description="build
everything" />

<!-- compile task -->
<target name="compile" depends="custom_service_gen" description="compile source
files" >
        <echo>" Compiling JAX-WS impl classes"</echo>
        <javac  srcdir="${srcDir}"
                destdir="${buildDir}"
                classpathref="classpath"
                debug="on"
                source="1.5"/>
</target>
<!--Create custom service jar -->
<target name="custom_jar" depends="custom_service_gen, compile"
description="generate jar file" >
        <jar jarfile="custom_services.jar" basedir="${buildDir}" >
        </jar>
</target>
<!--ant clean task -->
<target name="clean" description="remove derived objects" >
        <delete dir="classes/com"/>
        <delete dir="custom_service.jar"/>
</target>
</project>
```

where:

- *BRM_home* is the directory in which BRM is installed.
- *YourCustomServices_v2* is the custom service WSDL file name.
- *yourpackagesubdirname* is the package directory for your custom service.

5. Generate and build your custom web services by running the following command:

   `ant -file custom_services.xml`

6. Add all the custom field **enum** constants to the **Infranet.properties** file. See "Connecting Web Services Manager to the BRM System" for more information.

7. Package your custom web service with the **BrmWebServices.war** file by doing the following:

   a. Extract the **BrmWebServices.war** file to a local directory (*local_dir*) on the machine on which you installed your application server.

   b. Do one of the following:

   - (For WebLogic Server) Modify the *local_dir***/WEB-INF/Web.xml** file to include your custom service URL mapping similar to existing URL mapping.

- (For Apache Tomcat server) Modify the *local_dir***/WEB-INF/sun.jaxws.xml** file to add your custom service implementation class.

c. Copy your custom WSDL files and schema (XSD) files into the *local_dir***/WEB-INF/wsdl/** directory.

d. Copy your **custom_services.jar** into the *local_dir***/WEB-INF/classes** directory.

e. Copy your **CustomFields.jar** into the *local_dir***/WEB-INF/lib** directory.

f. Delete the existing **BrmWebServices.war** file.

g. Create a new **BrmWebServices.war** file by running the following command:

```
jar -cvf BrmWebServices.war *
```

Generating the Schema Files for Your System

Web Services Manager uses schema files to validate data it sends to or receives from BRM.

To generate the schema files for your system, do the following:

1. If you modified any opcodes, generate schemas for the opcodes in your BRM system. See "Generating the Schema for an Existing Opcode".
2. Generate schemas for the storable classes and subclasses in your BRM system. See "Generating the Schema for Your Storable Classes and Subclasses" in *BRM JCA Resource Adapter*.
3. In your opcode schema files, specify the location of your storable class schema files. See "Specifying the Location of the Storable Class Schema Files in the Opcode Schema Files" in *BRM JCA Resource Adapter*.

> **Note:**
>
> After generating the opcode and storable class schema files, copy the schema files to a location that is accessible to the Web Services Manager. Make sure that this location is the same as the location that is specified in the **include** section of the opcode schema files and in the opcode schema **InteractionSpec** attribute in the WSDL files. See "Specifying the Location of the Storable Class Schema Files in the Opcode Schema Files" and "Generating the WSDL Files for Your System" in *BRM JCA Resource Adapter*.

8

Generating the Schema for Your Opcodes

Learn how to use the Oracle Communications Billing and Revenue Management (BRM) Web Services Manager package with the opcode schemas and flist specifications you need for a default integration.

Topics in this document:

- Generating the Schema for an Existing Opcode
- Creating Opcode Specification Schema Files
- Specifying the XSL Rules to Create the Opcode Schema

If you customized any of the opcodes that are supported by Web Services Manager or if you added support for new opcodes, you must generate XSD schema files for the opcodes.

> **Note:**
>
> - Before you customize an existing opcode specification, ensure that you update the opcode specification in the BRM system.
> - After you customize web services, copy the customized schema files and the WSDL files to the **infranetwebsvc.war** file.

Generating the Schema for an Existing Opcode

To generate schema files for an opcode that you customized and Web Services Manager already supports:

1. Modify the opcode's XML specification file. By default, the opcode specification XML files are installed in the *BRM_home***/apps/brm_integrations/opspecs** directory, where *BRM_home* is the directory in which you installed the BRM components.
2. Do one of the following:
   - For web services that take payload as XML string:
     - Run the **pin_opspec_to_schema** utility. See "Creating Opcode Specification Schema Files".
     - Copy the customized XSD files to the *BRM_home*/**deploy/web_services/schemas** directory.
   - For web services that take payload as XML element:
     - Run the **pin_opspec_to_schema_v2** utility. See "Creating Opcode Specification Schema Files".
     - Copy the customized XSD files to the **infranetwebsvc/WEB-INF/services/InfranetWebservices.aar/META-INF** directory.

Creating Opcode Specification Schema Files

You must create opcode flist specification files for opcodes that you customize or add to the Web Services Manager. Create the specification XML files by following the *BRM_home*/**apps/brm_integrations/stylesheets/opspec.xsd** file.

You then convert the opcode flist specification XML files into XSD schema by using the **pin_opspec_to_schema** and **pin_opspec_to_schema_v2** utilities.

To convert opcode flist specification XML files into XSD schema, go to the *BRM_home*/**apps/brm_integrations** directory and do the following:

- For web services that take payload as XML string, run the following command:

  ```
  pin_opspec_to_schema -i input_file [-o output_file]
  ```

- For web services that take payload as XML element, run the following command:

  > **Note:**
  >
  > Before you run the following command, specify the BRM installation directory in the **pin_opspec_to_schema_v2** utility by replacing $PIN_HOME with *BRM_home*.

  ```
  pin_opspec_to_schema_v2 -i input_file > output_file
  ```

  where:

  – *input_file* specifies the name and location of the opcode's XML flist specification. By default, the utility looks for the file in the current directory.

  – *output_file* creates the XSD schema output file using the name you specify. By default, the utility creates a file named *opcodename***.xsd** in the directory from which you run the utility.

You can also create XSD schema for web services that take payload as XML element by using the **pin_opspec_to_schema_v2** XSD generator utility that is located in the *BRM_home*/**bin** directory.

To create the XSD schema file by using the **pin_opspec_to_schema_v2** utility, run the following command using Groovy:

```
groovy pin_opspec_to_schema_v2 -i input.xml > output.xsd
```

where:

- *input.xml* specifies the name of the opcode's XML flist specification
- *output.xsd* creates the XSD schema output file using the name you specify

Specifying the XSL Rules to Create the Opcode Schema

The **pin_opspec_to_schema** utility uses the *BRM_home*/**brm_integrations/stylesheets/pin_opspec_to_schema.xsl** style sheet to generate the schema for BRM opcodes. If your opcode references custom fields, you must customize the **pin_opspec_to_schema.xsl** style sheet to handle your custom fields.

For a list of the supported BRM data types, see "Understanding the BRM Data Types" in *BRM Developer's Guide*.