ORACLE INSURANCE

Oracle Health Insurance Back Office

Oracle Health Insurance Web Services

Installation Guide for WLS

version 1.5

Part number: E51467_01

December 2013

CHANGE HISTORY

Release	Version	Changes
10.12.2.0	1.1	• Added Appendix B for functional testing.
10.13.1.0	1.2	• Corrected some small typing errors (especially the minus sign in front of the –Dlog4j... setting which is not correct and resulted in errors when copying the line to a WebLogic configuration).
10.13.1.1	1.3	• Adjusted references for FRS11G1 to FRS11G2.
10.13.2.0	1.4	• Added additional information regarding English version
10.13.2.0	1.5	• Added an English example message

RELATED DOCUMENTS

Doc[1]: Oracle Health Insurance Back Office Service Layer Installation & Configuration Manual (CTA13651)

Contents

1. Introduction

This document describes the installation and configuration of the OHI Connect to Back Office Web service released by OHI Back Office.

1.1. Purpose

Describes the installation and configuration of the OHI Connect to Back Office web service in an Oracle Weblogic Server 11G environment (part of Oracle Fusion Middleware 11g).

1.2. Audience

This document is intended for administrators of Oracle WebLogic Server. Required knowledge:

- Working knowledge of Oracle WebLogic Server 11g version 10.3.3 or higher.
- The creation of Domain and Managed Server in WebLogic Server.
- The creation of JDBC Data Sources in WebLogic Server
- The creation of JMS queues and connection factories in WebLogic Server
- The deployment for EAR files in WebLogic Server.

Administrators of Oracle10g AS may find the following comparison between Oracle10g AS and WebLogic Server useful:

http://download.oracle.com/docs/cd/B31017_01/migrate.1013/b31269/compare_weblogic.htm

1.3. Document structure

This document is organized as follows:

- Overview of the OHI Back Office web services
  Describes the SOAP/JMS and SOAP/HTTP web services on a conceptual level.
- Installation preparation
  Applies to both SOAP/JMS and SOAP/HTTP implementations, unless stated otherwise.
- Installation of SIC_OOZCLAIMSDATAS (SOAP/HTTP)
- Installation of SIC_OOZWEBSERVICEJ (SOAP/JMS)
  Describes the minimum procedure for configuring the queues needed by SIC_OOZWEBSERVICEJ.
- Administration

Finally, appendix A describes to deploy the synchronous version of SIC_OOZWEBSERVICE.

1.4. Release

OHI offers the following web services for OHI Connect To Back Office:

- SIC_OOZClaimsData (synchronous)
  Retrieve claims information.

- SIC_OOZWebservice (synchronous and asynchronous)
  Multiple services to update / retrieve data in OHI Back Office.

When invoked, the web service connects to the OHI Back Office database to perform the required action and returns a response message to the calling application.
The action taken by an OHI web service either retrieves data from the OHI Back Office database or puts data into the OHI Back Office database. In both cases the OHI Back Office database must be online in order to complete the required action.

In some cases the calling application does not depend on an immediate response from the web service. In those cases, an asynchronous web service can be used.

The SIC_OOZWebservice has both a synchronous (SOAP/HTTP) and an asynchronous interface (SOAP/JMS). Apart from the interface, both variants use the same code.

All OHI web services are released as EAR files. Whether a web service is asynchronous or synchronous can be derived from the file name:

- asynchronous web service: <WEBSERVICE NAME>J.ear

- synchronous web service: <WEBSERVICE NAME>S.ear

So we have SIC_OOZCLAIMSDATAS.ear for the synchronous web service to retrieve claims data and we have SIC_OOZWEBSERVICES.ear and SIC_OOZWEBSERVICEJ.ear as two variants of our general purpose web services.

1.4.1. Dutch versus English

As these web services use Dutch XML messages it is important to know that there are also English versions of the OOZWEBSERVICE variants. These can be recognized by having '_EN' just in front of the ...J.ear or ...S.ear, so SIC_OOZWEBSERVICE_ENJ.ear and SIC_OOZWEBSERVICE_ENS.ear.

Only when needed this documentation distinguishes between the Dutch and English version. Otherwise simply replace the file names for the Dutch versions in the instructions by the file names used for the English version.

1.5. Software versions

The following software must be installed before OHI Connect to Back Office web services can be installed:

- OHI Back Office (including database software)

- Oracle WebLogic Server

Note that this installation assumes that the OHI web services are installed on the same node as the OHI Back Office application.

1.5.1. OHI Back Office

The interface works with OHI Back Office release 2006.02.4.0000 and above.

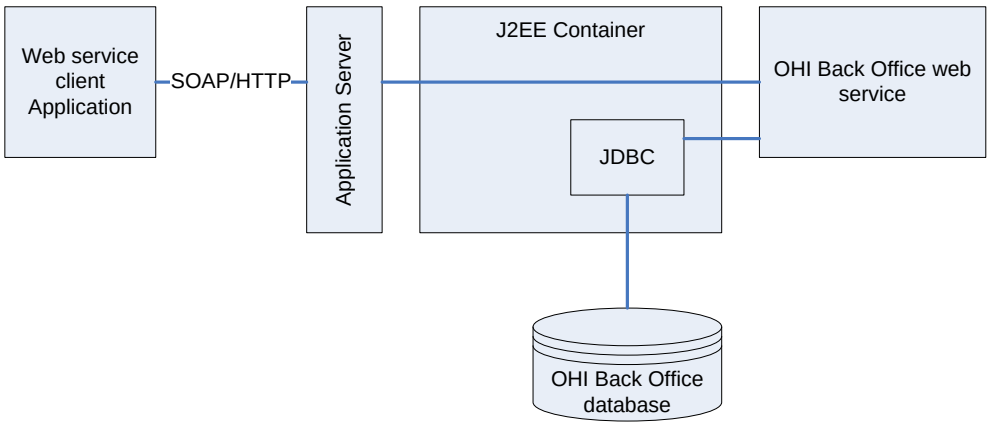
1.5.2. Oracle WebLogic Server

Starting with release 2011.01.0.0000 of OHI Back Office, the OHI Connect to Back Office must be deployed on Oracle WebLogic Server 11g. The version of the web services in the 2011.01.0.0000 release has been tested with Oracle WebLogic Server 11g version 10.3.3.

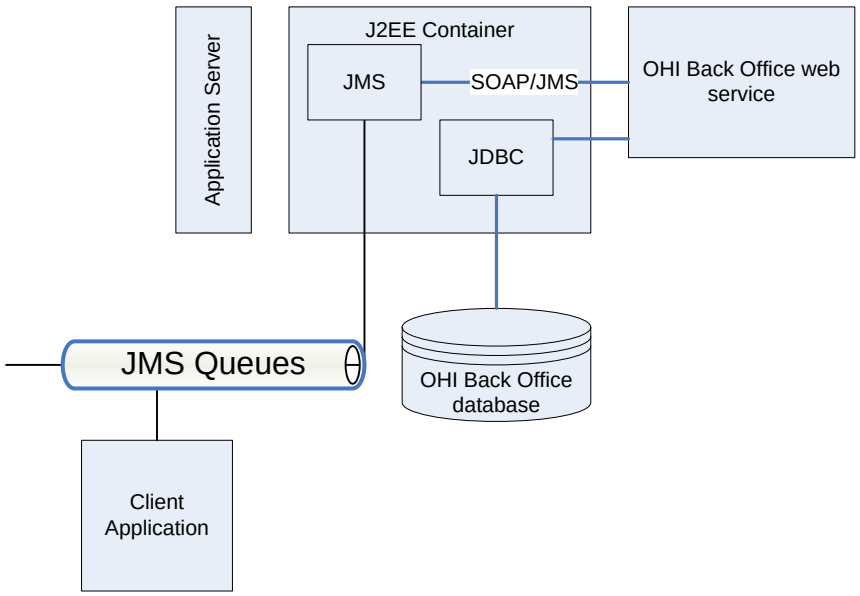
2. Overview of the OHI Back Office web services

As indicated before, there are synchronous and asynchronous OHI Back Office web services.

The diagram below shows the components for a synchronous (SOAP/HTTP) web service:



In the case of an asynchronous web service, the request is stored in a message queue. The message remains in the queue until it can be processed by the OHI Back Office web service. The web service creates a response which is stored in a different queue until it is processed by the calling application:



The advantage of asynchronous web services is that the server application can be offline when the request is created and that the client application can be offline when the request is processed. The price for this is that all communication between client and server is temporarily stored in message queues.

2.1. SOAP/JMS web service

The asynchronous web service (SOAP/JMS) is installed within a J2EE container (a J2EE runtime environment within the application server).

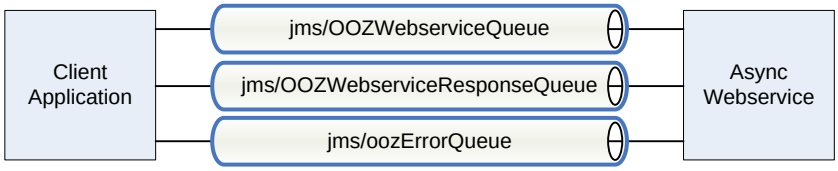
The communication with this web service is handled by JMS queues.

The invocation is triggered by a message on the inbound queue, jms/OOZWebserviceQueue.

The payload of this message is an XML document. The XML document contains the data to create PL/SQL calls in OHI BackOffice through a JDBC database connection.

If the request is handled successfully, the response is also an XML document which is put in a message on the outbound queue, jms/OOZWebserviceResponseQueue.

In case of an error, a message is created in the error queue, jms/oozErrorQueue.



The standard queue connection factory for accessing these queues is jms/oozQueueConnectionFactory.

2.1.1. English queue names

When you use the _EN (English) versions of the OOZWebService the names of the request and response queue contain 'C2B' instead of 'ooz'.

2.2. SOAP/HTTP web service

The SOAP/HTTP web service (OOZCLAIMSDATAS) is a Stateless Session Bean. This receives a SOAP message, processes the request to OHI Back Office and returns the response.

The internals of a SOAP/HTTP service are identical to the SOAP/JMS service.

3. Installation preparation

The installation preparation contains the following steps:

- Verification of required files
- Database preparation
- WebLogic Server Preparation

3.1. Verification of required files

3.1.1. Required Files

Depending on the agreements made, the following files are supplied by Oracle on Beehive Online (in future eDelivery) as part of a release delivery:

- SIC_OOZWEBSERVICEJ.EAR
  (SOAP/JMS web service containing various services for Connect 2 Back Office)
- SIC_OOZCLAIMSDATAS.EAR
  (SOAP/HTTP web service for claims enquiries)

3.2. Database preparation

3.2.1. Database objects

Before the installation the required database objects in the OHI Back Office must be installed. This task is usually carried out by the OHI Back Office DBA.

3.2.2. Oracle account to be used

Before the start of the installation, determine how the web service should connect with the OHI Back Office database.

Item	Description
Host	Host of the TNS listener for the OHI Back Office database runs. Normally this is the database server.
Port	Port number of the TNS listener Typically this is 1521 or 1526
Sid	The ORACLE SID of the OHI Back Office database
Oracle account	The Oracle account used to access the OHI Back Office database
Password	Password for the Oracle account

These data will be used to create a JDBC data source.

Note: the most obvious scenario would be to use the owner of the database objects as the oracle account, e.g. ozg_owner.
However, for security reasons it is desirable to use an alternative Oracle account to

which only the most strictly necessary rights have been granted (there is no standardized grant script available for this).

3.3. WebLogic Preparation

A certified version of the Weblogic application server must be installed prior to this step.

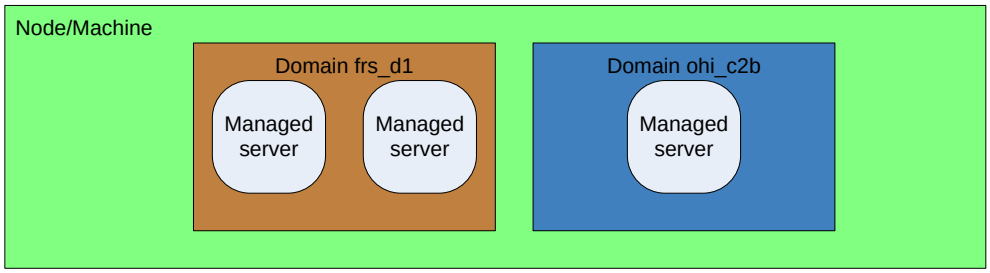
The Weblogic preparation consists of the following actions

- Create domain, managed server, machine
- Create of data source

3.3.1. Create domain, managed server, machine

Instead of adding the web services to the domain you are already using for OHI Back Office, we strongly advise to create a new domain for the Connect To Back Office web services.

We must now create managed server and machine definitions for this new domain. Note in the diagram below how the existing frs_d1 and new ohi_c2b domain coexist:



The steps for creating domain, managed server and machine are documented in paragraph 3.2 of **Doc[1]** but can also be found in the WebLogic documentation.

3.3.1.1. Create domain

Start the configuration wizard for creating a new domain:

- Log in as the UNIX account running WebLogic (eg. oracle)
- `. ozg_init.env $OZG_ORATAB_FRS11G2`
- `$WL_HOME/common/bin/config.sh`

Generate a new domain for supporting web services with its own administration server as per paragraph 3.2.1 of **Doc[1]**.

Do not use port 7001 for the administration server but, for example, port 7070.

Start the administration server for this domain (suppose it is named ohi_c2b):

```
$WL_HOME/../user_projects/domains/ohi_c2b/startWebLogic.sh
```

3.3.1.2. Create managed server, machine etc.

The remainder of the procedure is done through the HTML interface of the administration server.

Follow the steps in chapter 3.2 of **Doc[1]** to create a new domain with a single managed server. Ensure that you configure an admin server within the new domain and that the port number for the admin server has not been used (we chose 7070).

Suggested names:

- Domain: ohi_c2b
- Machine: machine1
- Managed Server: MS1 (we used port 7071)

3.4. Creating JDBC Data Source

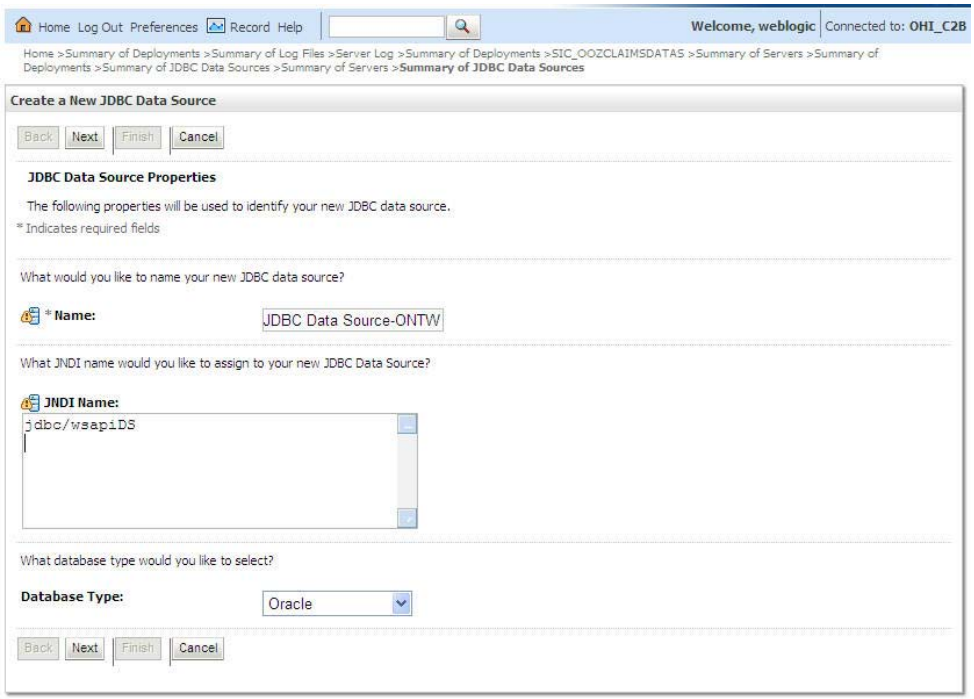
In WebLogic Server, you can configure database connectivity in two steps:

- Define a JDBC data source
- 'Target' the JDBC data source to the domain or to a managed server.

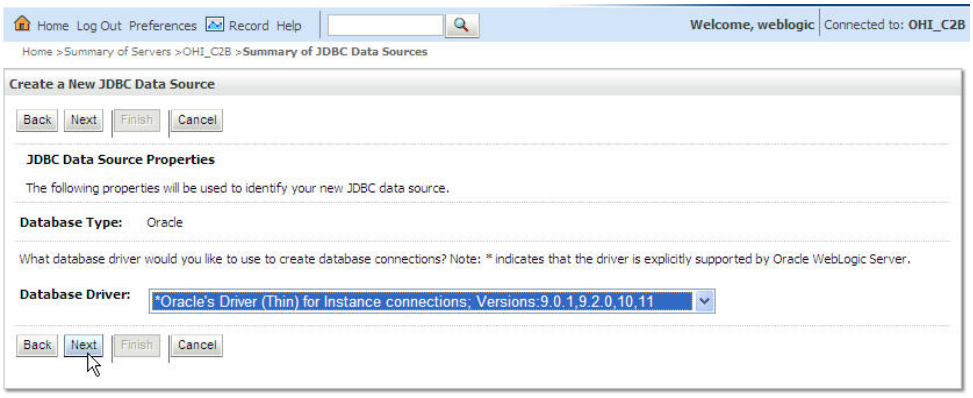
Name	Value
JNDI Name	jdbc/wsapiDS
Database	Oracle
Database Driver	Oracle's Driver (Thin) for Instance connection; Version: 9.0.1,9,2,0,10,11

3.4.1.1. Create data source using JNDI name

Create a generic data source.

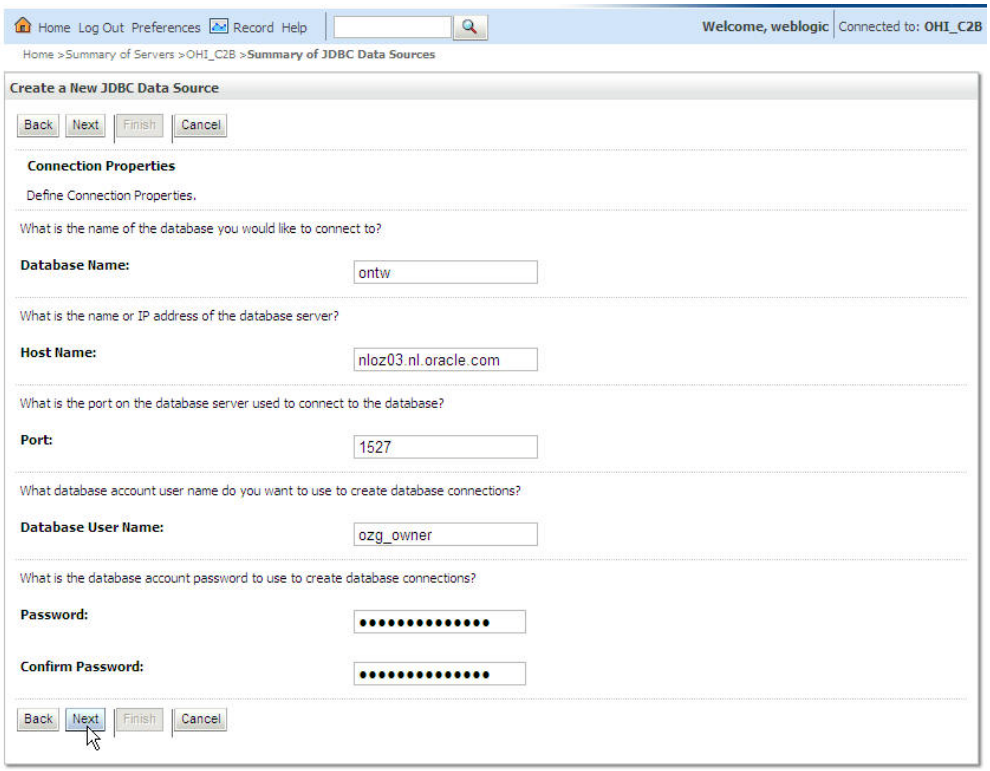


3.4.1.2. Select database driver



3.4.1.3. Set up connection properties

Define the Connection Properties using the values described in 3.2.2.



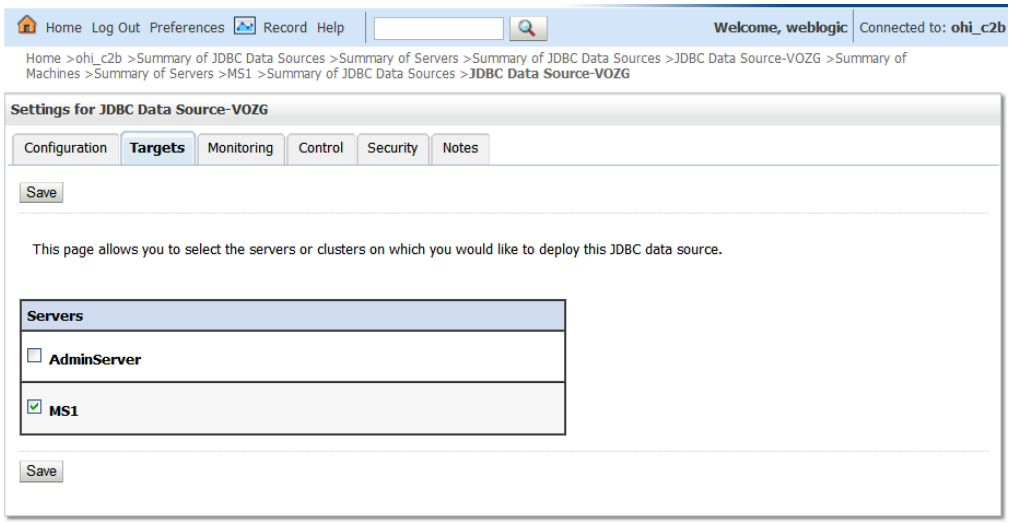
3.4.1.4. Test database connection

Repeat previous step until the database connection test is successful.

3.4.1.5. Target the JDBC data source

By 'targetting' the JDBC data source you make it available for use.

Select a Managed Server as target.



3.5. Start the managed server

4. Installation of SIC_OOZCLAIMSDATAS

The installation of a SOAP/HTTP web service is relatively straightforward because the service only depends on the JDBC data source which is used to connect to the OHI Back Office database.

First, carry out the preparatory activities as described in Ch 3.

Now you can deploy the SIC_OOZCLAIMSDATAS.EAR file.

5. Installation of SIC_OOZWEBSERVICEJ

This chapter describes how to install the SIC_OOZWEBSERVICEJ service. The actual deployment of the EAR file is a minor step compared to configuring the JMS queues. For this procedure we tried to simplify the configuration.

The main characteristics are:

- Non-clustered installation
- Native JMS, using in-memory queues.

The installation procedure contains the following steps:

- Create JMS Server
  This process will implement the queuing mechanism and persist queue data.
- Create JMS Module
  This container will hold the resources (queue connection factory and queues)
- Create queue connection factory
  This is used to access the queues handled by the JMS Server
- Create subdeployment
  This is a mechanism to link queues to the JMS server
- Create JMS queues
- Deploy application (EAR file).

5.1. Before you start

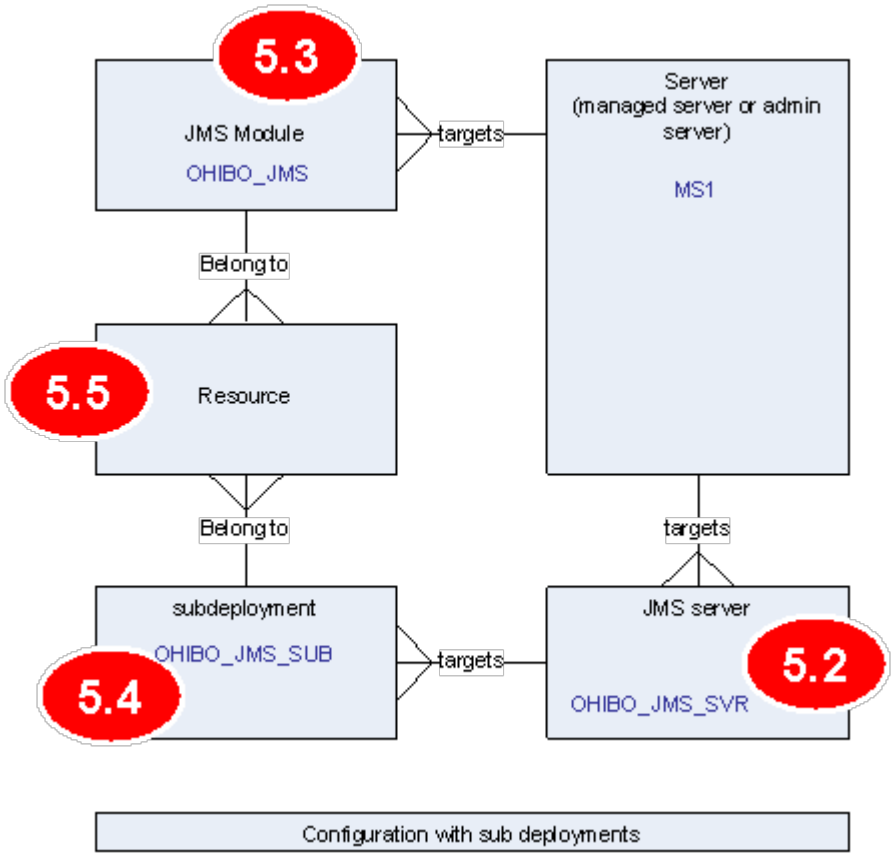
The following JMS queues must be created:

- jms/OOZWebServiceQueue
  (for the English version: jms/C2BWebServiceQueue)
- jms/OOZWebServiceResponseQueue
  (for the English version: jms/C2BWebServiceResponseQueue)
- jms/oozErrorQueue

The queues are accessed through a connection factory called jms/oozQueueConnectionFactory.

In Weblogic, JMS queues and JMS connection factories are called JMS resources. Resources can only be created within the context of a JMS module. The JMS module itself must belong to a (managed) server, while its resources are handled by a JMS server.

The diagram below visualizes how the different components in this configuration procedure relate to each other (shown as an entity relation diagram). Note that the numbering of the components corresponds with the paragraphs which describe the configuration of that particular component. The names used for the components are indicated in blue:



5.1.1. Alternative implementations

The installation procedure assumes a memory based, non-clustered implementation of the JMS queues.

Generally speaking, all queues will be (nearly) empty during normal operation. Note however that if the application server is shut down by force or as a result of a crash, the contents of the queues are lost, and new requests can not be queued by the calling application.

To avoid this, you can do the following:

- Use persistent storage for the queue messages (either in the database or in a File Store).
- Create a multi-node, clustered configuration. This allows you to take an application server instance down without interrupting the service.

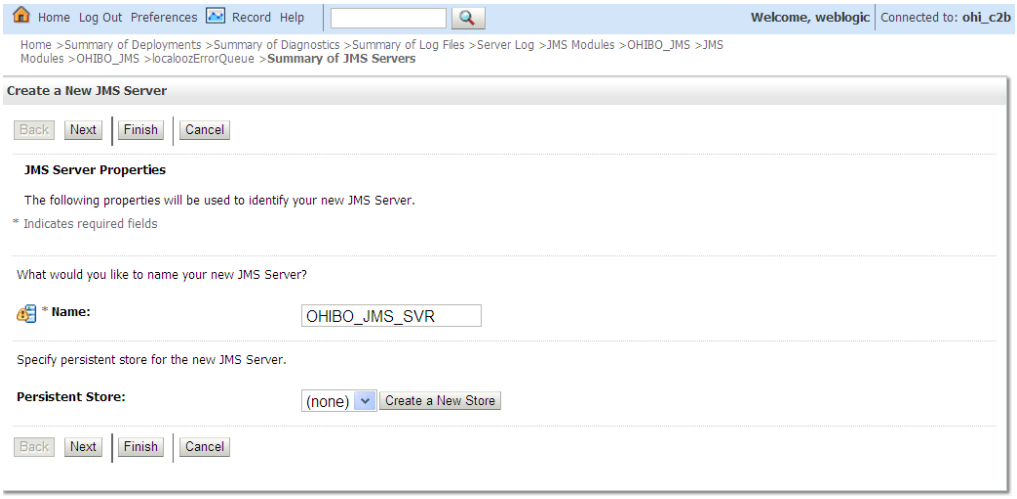
If you choose to take this approach, the following changes apply to the installation procedure:

- Create Persistent Stores (eg. File Stores) for each managed server and associate the JMS server of each managed server with its Persistent Store.
- Create Distributed Queues instead of normal queues. Instead of linking the queues to a subdeployment, you can now use default targeting.

Refer to your Weblogic documentation for more detailed instructions.

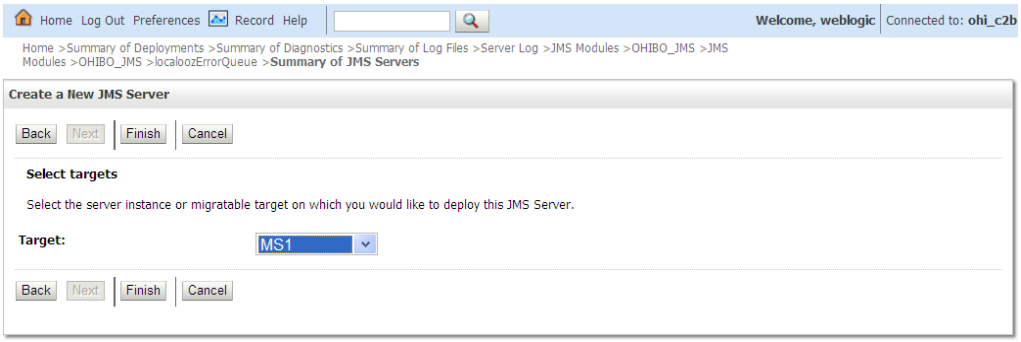
5.2. Create JMS Server

Choose in the Domain Structure pane Services > Messaging > JMS Servers and create a new JMS Server.



5.2.1. Target JMS Server

The JMS server must be targeted to a managed server.



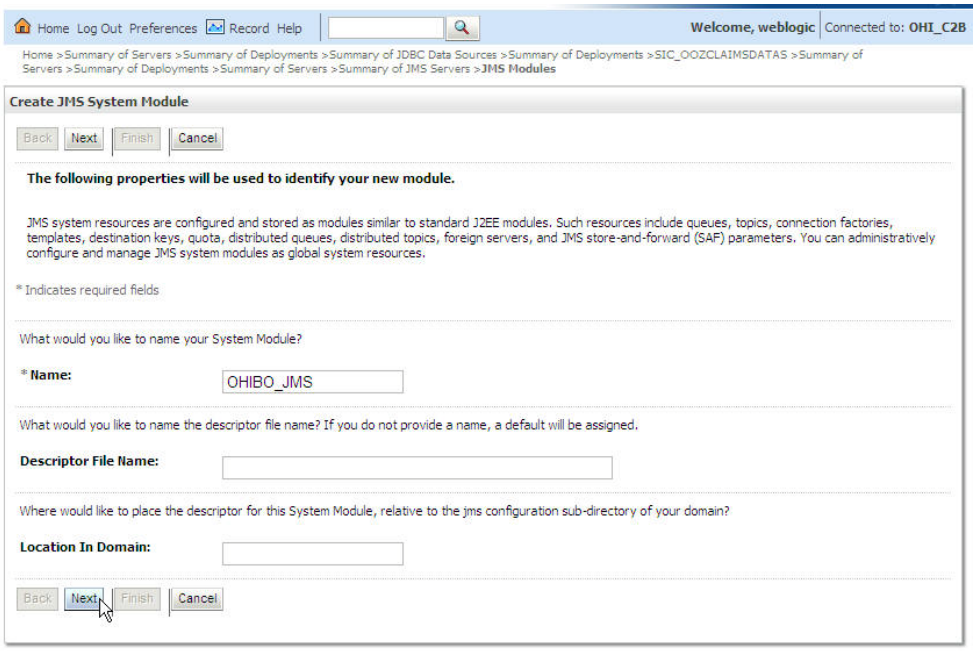
5.3. Create and configure JMS Module

A JMS Module is simply a container to hold JMS resources. Apart from the JMS queues and connection factory, a JMS Module may also hold a 'Foreign Server'.

A Foreign Server represents JMS provider that is outside the local WebLogic Server. It contains information that allows local server instance to reach a remote JNDI provider.

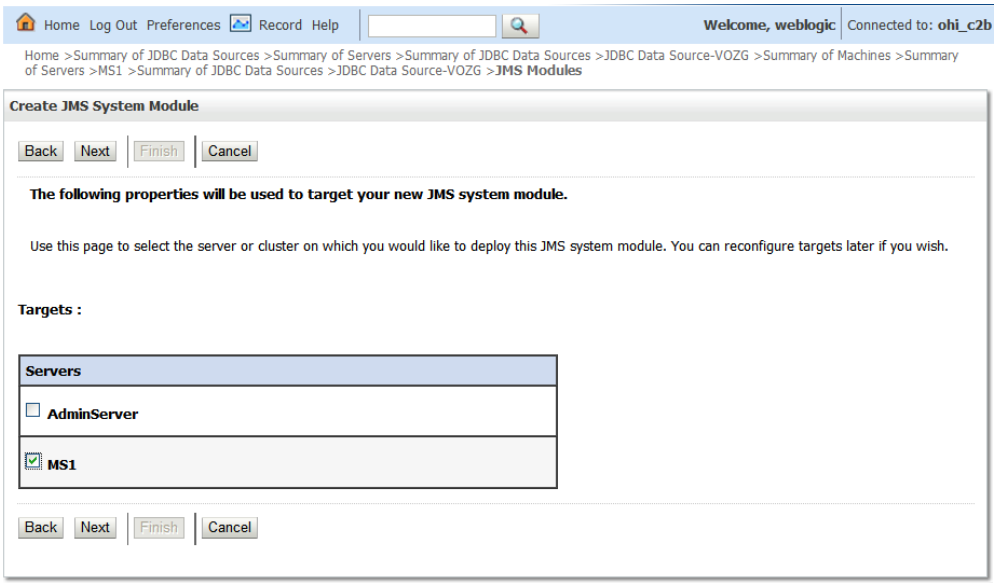
5.3.1. Create System Module

Choose in the Domain Structure pane Services > Messaging > JMS Modules



5.3.2. Target the JMS module

Next you must 'target' the new JMS module to a server. Choose the managed server of the ohi_c2b domain.



5.3.3. Finish JMS Module creation

Finish the JMS Module creation. There is no need to continue to create the resources because first we will create a subdeployment first.

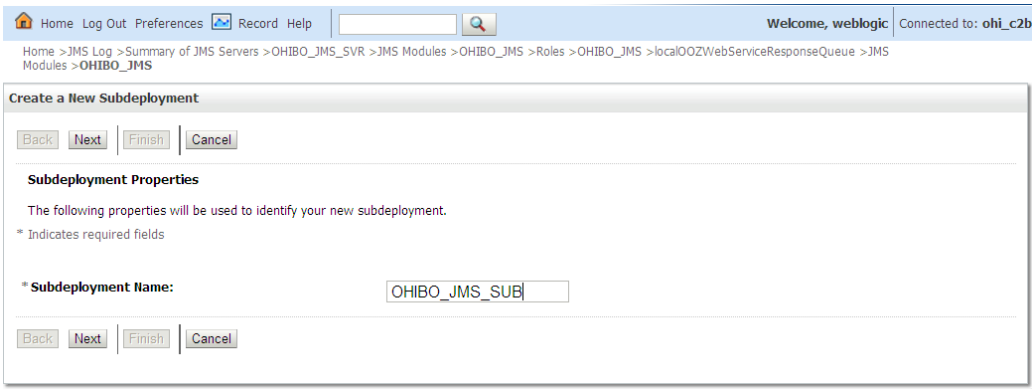
5.4. Create subdeployment

According to the Weblogic documentation, standalone queues must be targeted to JMS servers. Since this cannot be done directly, we use a subdeployment as a go-between.

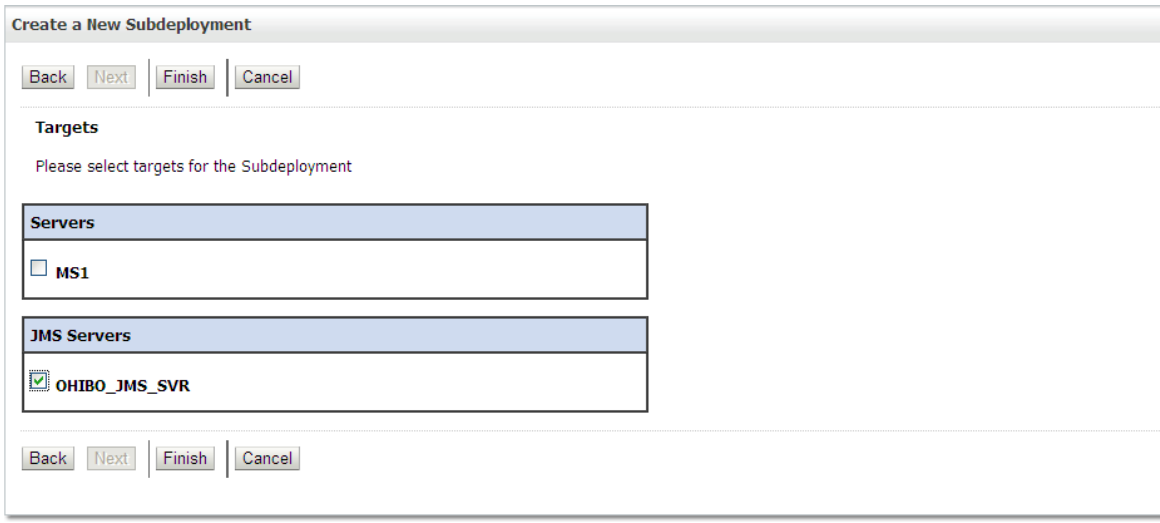
More information can be found in:

http://download.oracle.com/docs/cd/E12839_01/web.1111/e13738/basic_config.htm#i1151524

Navigate to the JMS Module you just created and choose the Subdeployments tab.

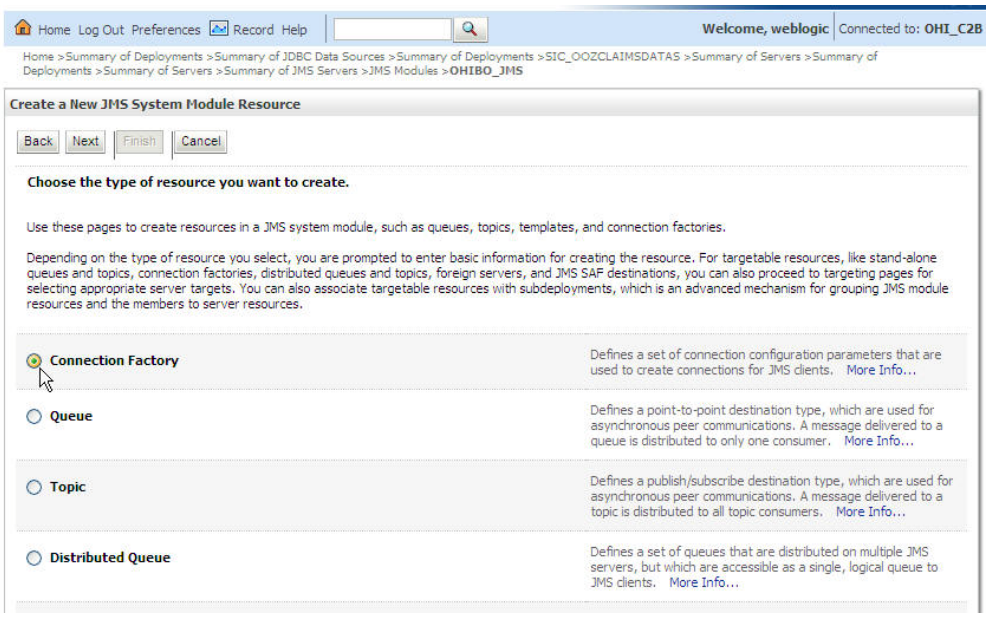


5.4.1. Target subdeployment to JMS server

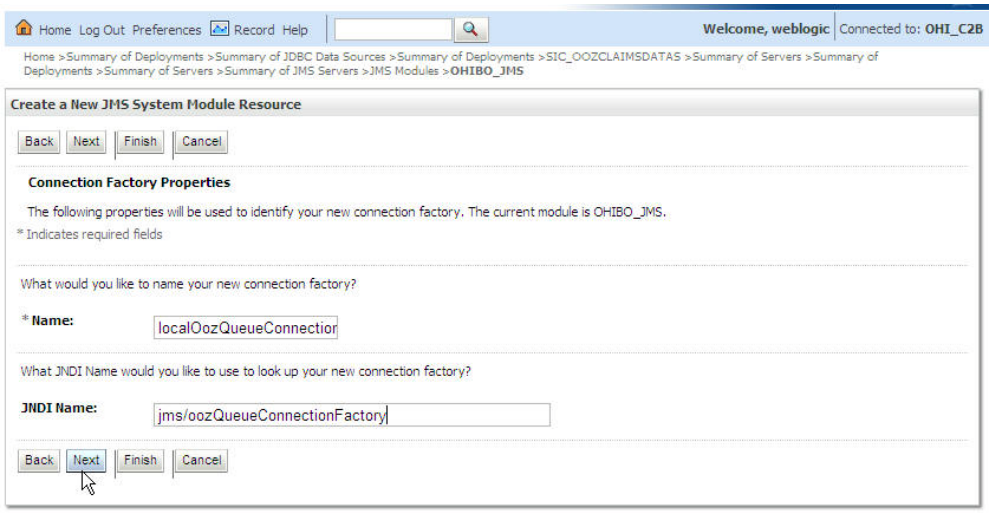


5.5. Create Connection Factory

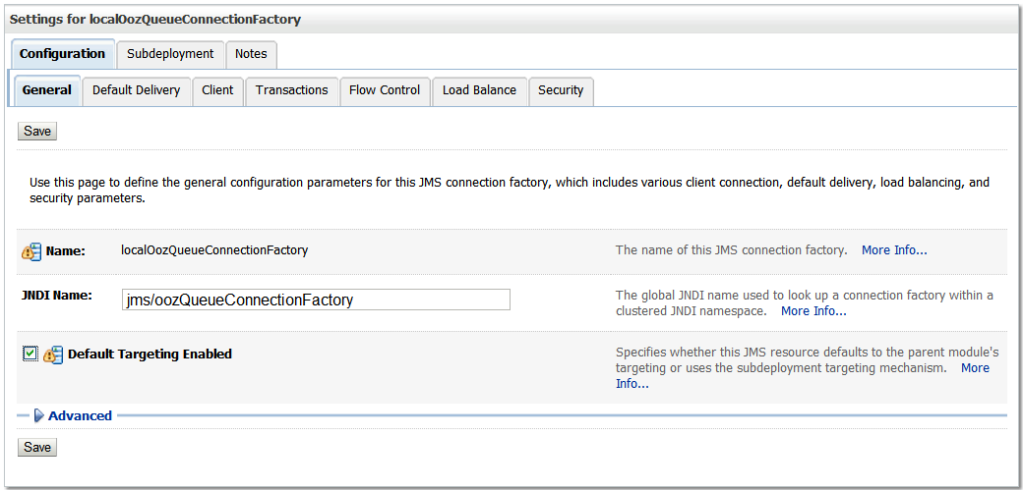
In the webservice program, the queue connection factory is used to access the JMS queues. Create a JMS System Module Resource of type 'Connection Factory'. For this select the create JMS Module and use the New button to create a new resource.



Set the JNDI name to jms/oozQueueConnectionFactory. This JNDI name is used by the webservice program.



Ensure that default targeting is enabled.

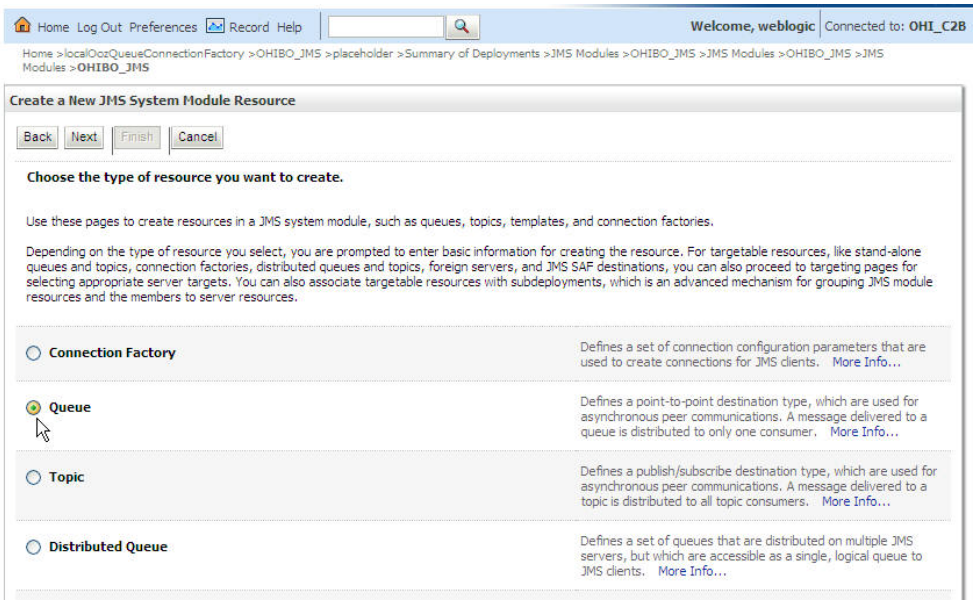


5.6. Create Queues

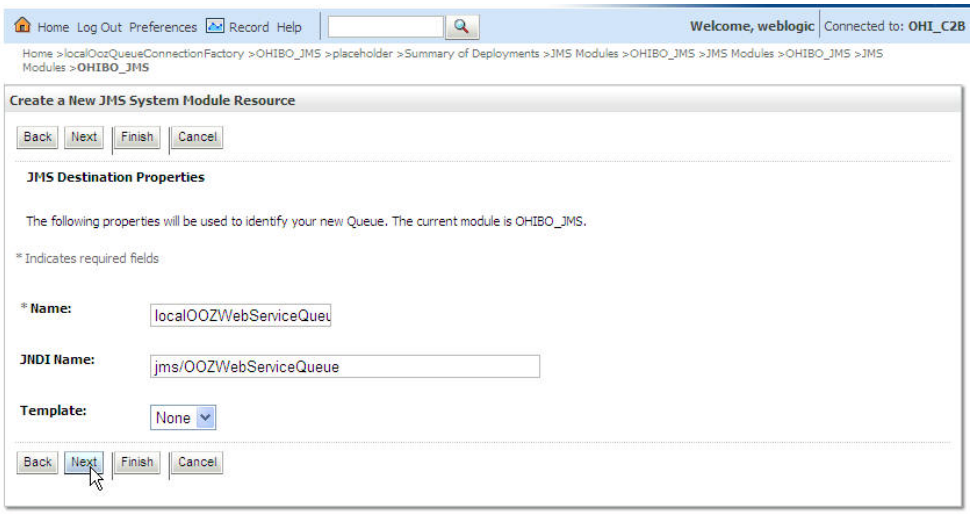
Create JMS queues with the following JNDI names (replace OOZ by C2B if you use the English version!):

Queue	JNDI Name
Request queue	jms/OOZWebServiceQueue
Response queue	jms/OOZWebServiceResponseQueue
Error queue	jms/oozErrorQueue

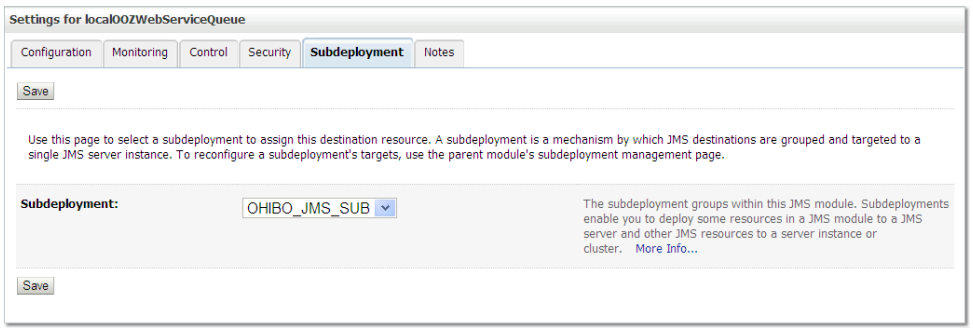
Create another JMS Module Resource but this time of type ‘queue’:



Enter name and JNDI name. The JNDI name of each queue must correspond with an entry in the list above.

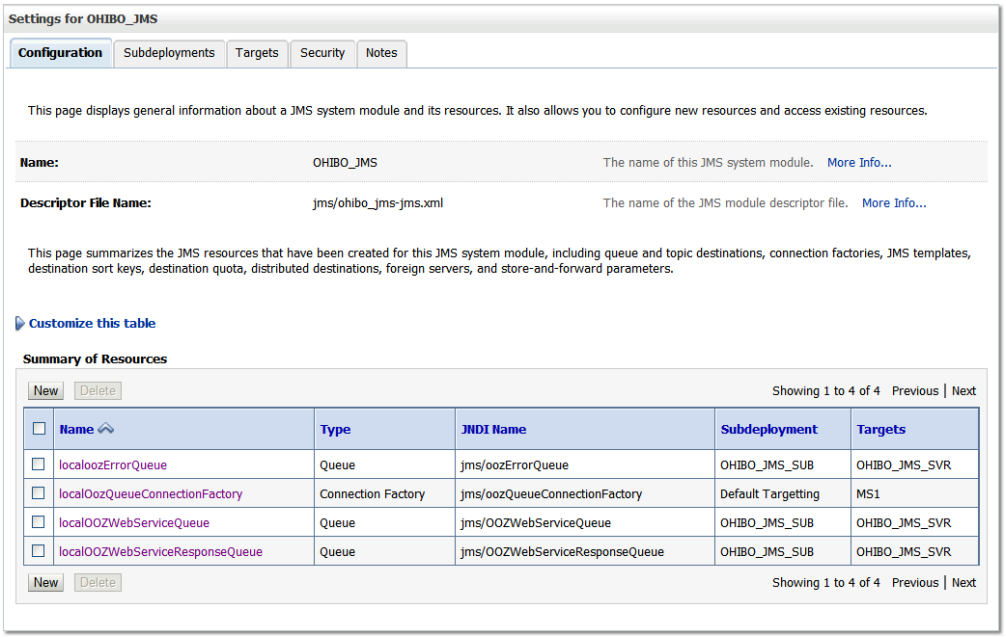


Finally, link the queue to the subdeployment:



The process for each queue is identical. The procedure below was used to create one of the queues, repeat this process for the remaining queues.

Finally, check that all queues are created:

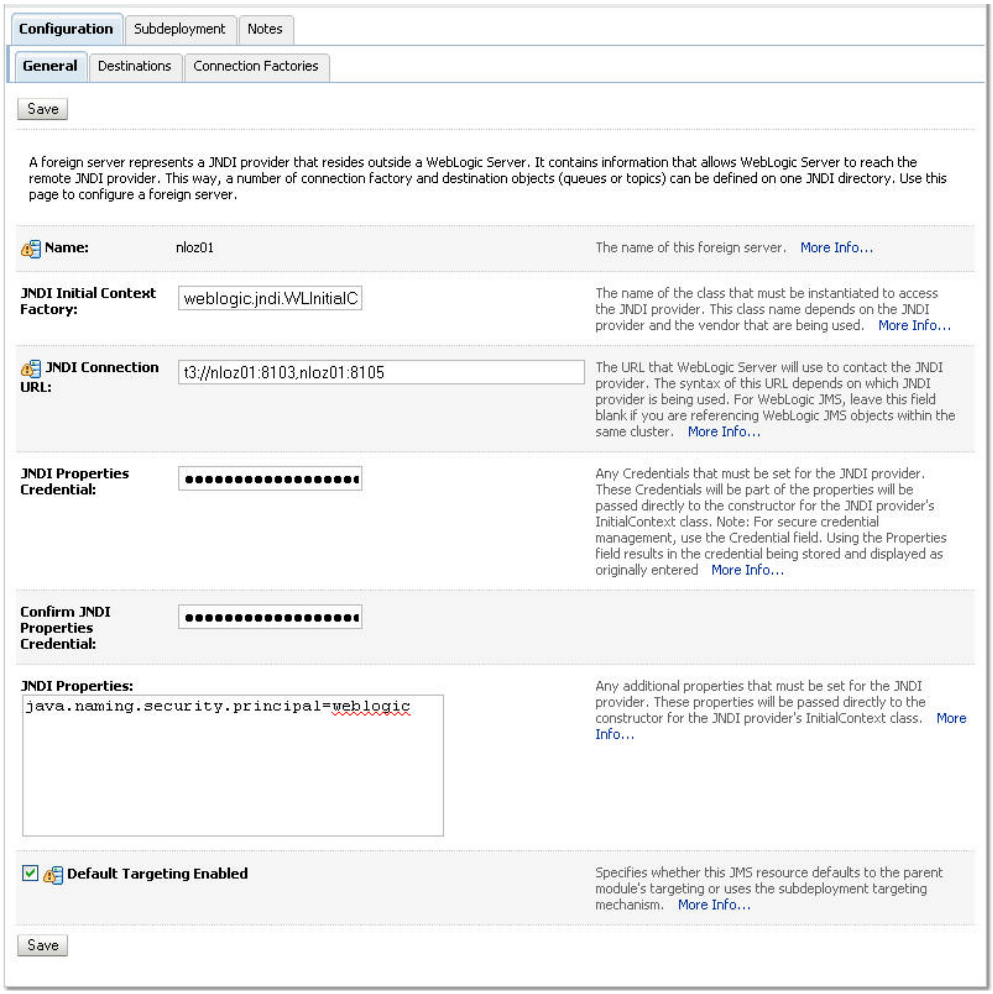


5.7. Create foreign server (only when using OHI Self Service)

If you deploy OHI Connect to BackOffice for OHI Self Service you need to configure a Foreign Server. A Foreign Server represents a JNDI provider that is outside local WebLogic Server. It contains information that allows a local WebLogic Server instance to reach a remote JNDI provider.

Create a Foreign Server with the following property values:

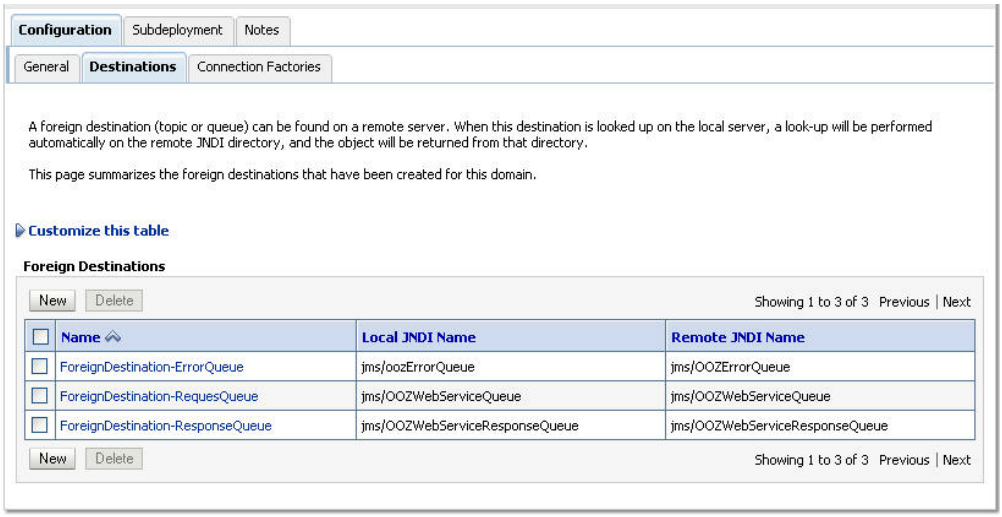
Name	Value
JNDI Initial Context Factory	**weblogic.jndi.WLInitialContextFactory** This is the default.
JNDI Connection URL	**t3://<host>:<port>,<host>:<port>** The URL that WebLogic Server will use to contact the JNDI provider. <host>: DNS name or IP address of the server that hosts the OHI Self Service. <port>: the port number from which you want to access the OHI Self Service server instance. e.g.: t3://nloz01:8103,nloz01:8105
JNDI Properties Credential	The credentials that must be set for the JNDI provider. Specify the secure password for the domain where OHI Self Service runs. These credentials must be specified along with the domain username in the JNDI Properties field.
JNDI Properties	**java.naming.security.principal=<*weblogic account*>** <*weblogic account*> is a secure username for the domain where the OHI Self Service runs, e.g. weblogic



After creating a foreign server, you need to define a foreign connection factory and foreign destinations.

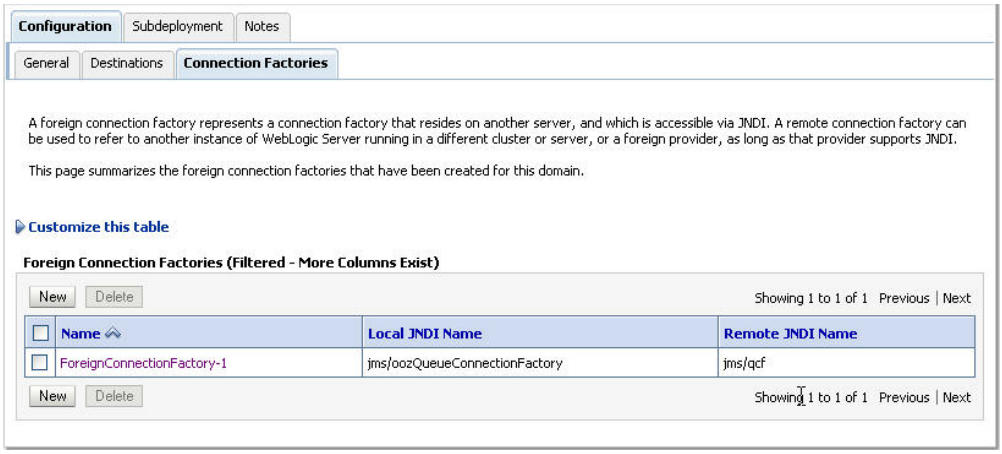
A Foreign Destination represents a queue that can be found on the remote server. Create foreign destinations with the following Local JNDI Names (replace OOZ by C2B if you use the English version!):

Local JNDI Name	Remote JNDI Name
jms/OOZWebServiceQueue	The JNDI name of the request queue at the OHI Self Service server.
jms/OOZWebServiceResponseQueue	The JNDI name of the response queue at the OHI Self Service server.
jms/oozErrorQueue	The JNDI name of the error queue at the OHI Self Service server.



You also need to define a foreign connection factory. Create a foreign connection factory with the following Local JNDI Name:

Local JNDI Name	Remote JNDI Name
jms/oozQueueConnectionFactory	The JNDI name of the connection factory in the remote JNDI provider, in this case the WebLogic instance where OHI Self Service running.



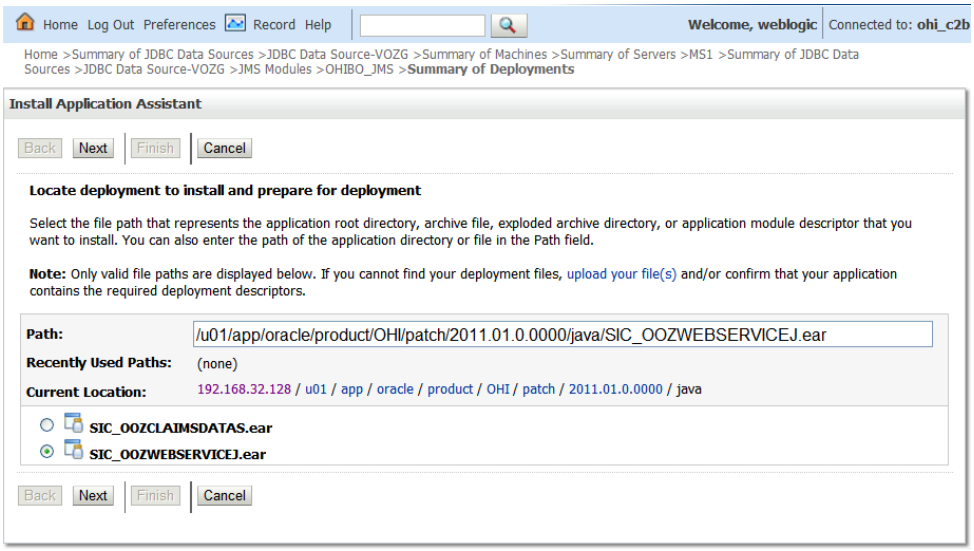
5.8. Deploy the application EAR file

Ensure that the managed server is running and that the JMS Server is healthy.

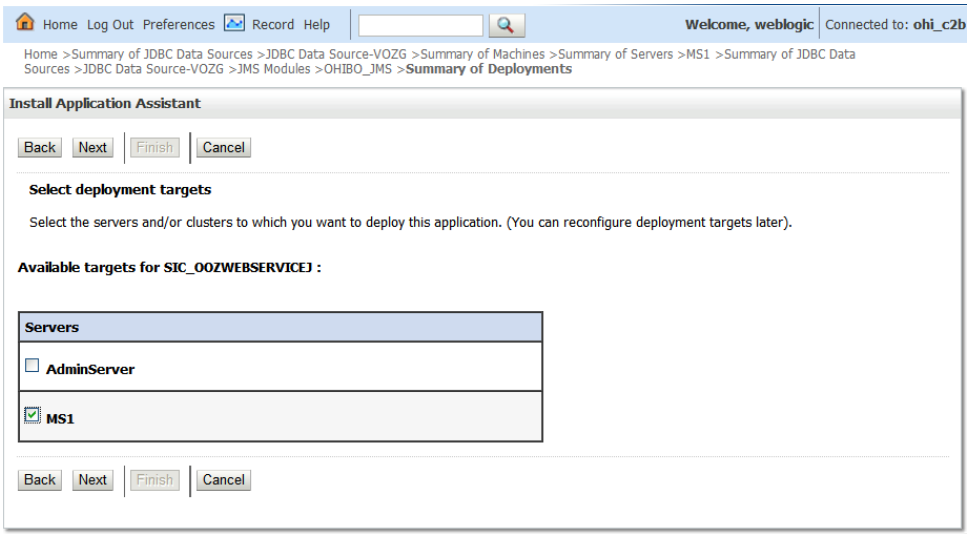
Now you can deploy the EAR file. Weblogic assumes that the EAR file can be found through the file system of the Admin Server host.

In our example, we will deploy the SIC_OOZWEBSERVICEJ.EAR through the GUI.

Go to 'deployments' and select the EAR file to be deployed:



Install the EAR file as an application and target the EAR file to a managed server:



Do not change the 'optional settings'.

Finally start the newly deployed EAR file (start the server process).

If you receive a warning during start up, check the following:

- Is the JMS Server OK?
- Do you have a subdeployment targeted at the JMS Server?
- Have you used the correct JNDI names?
- Are the queues linked to the subdeployment?
- Has the connection factory been configured for default targeting?

6. Administration

6.1. Logging

The Connect to Back Office web services are set up for logging through log4j. The use of log4j is a widespread open source solution. More information can be found on

- http://logging.apache.org/log4j
- http://en.wikipedia.org/wiki/Log4j

To set up logging, the following steps must be performed:

- Set up directory structure
- Create log4j configuration file
- Configure WebLogic to use log4j

6.1.1. Set up directory structure

It may be convenient to retain the $OZG_BASE/$OZG_ADMIN structure to support the Connect to Back Office Web Services:

- $OZG_BASE/java: used to locate EAR files
- $OZG_ADMIN: log4j.properties, output files for web services logging.

In the remainder of this chapter we will assume that

- $OZG_ADMIN will be used for locating log configuration and log output files.
- $OZG_ADMIN is located at /u01/app/oracle/product/OHI/admin
- $OZG_ADMIN does not include symbolic links

Beware that in an environment with multiple OHI Back Office installations sharing the same $OZG_ADMIN folder it may be convenient to use $OZG_BASE instead of $OZG_ADMIN for the log4j configuration and output files.

6.1.2. Create log4j configuration file

Below is an example of a configuration file.

```
#
# Console appender
#
log4j.appender.C=org.apache.log4j.ConsoleAppender
log4j.appender.C.target=System.out
log4j.appender.C.layout=org.apache.log4j.PatternLayout
log4j.appender.C.layout.ConversionPattern=%-5p %d [%t] %c: %m%n
#
# File appender; this should be a RollingFileAppender to prevent it
# from becoming to large because only in that case MaxFileSize etc.
# will be interpreted.
```

```
#
log4j.appender.F=org.apache.log4j.RollingFileAppender
log4j.appender.F.File=/u01/app/oracle/product/OHI/admin/SIC_OOZ_Webservices.log
log4j.appender.F.MaxFileSize=50000KB
log4j.appender.F.MaxBackupIndex=1
log4j.appender.F.layout=org.apache.log4j.PatternLayout
log4j.appender.F.layout.ConversionPattern=%-5p %d [%t] %c: %m%n
#
# Log settings:
#
log4j.rootCategory=INFO,F
log4j.logger.com.oracle.ohi.c2b.sicoozwebservicej=INFO,F
```

Note that for each web service you need to specify a logger entry like:

```
log4j.logger.com.oracle.ohi.c2b.sicoozwebservicej=INFO,F
```

In the above example, the logger name is the package associated with the web service. In this case *com.oracle.ohi.c2b.sicoozwebservicej* was used for the SIC_OOZ_WEBSERVICEJ web service

Finally, ensure that the file permissions of log4j.properties restrict access to authorised users only. In our test set up, we applied 'chmod 640' on log4j.properties.

6.1.3. Configure WebLogic to use log4j

Start the WebLogic administration console.

- Select the managed server for which you want to set up logging.
- Select Logging/Advanced
- Change the Logging Implementation from JDK to Log4j
- Select Server Start/Arguments and add
  -Dlog4j.configuration=file:/u01/app/oracle/product/OHI/admin/log4j.properties
- Copy $WL_HOME/server/lib/wllog4.jar to $DOMAIN/lib ($WL_HOME/../user_projects/domains/<domain>/lib
- Download log4j.jar (for example from http://logging.apache.org/log4j/) to $DOMAIN/lib
- Stop and start the managed server
- Check the managed server log to verify that log4j is being used.

Verify the output and log file which where created during startup of the managed server. Warnings like this:

```
log4j: WARN no appenders could be found for logger XYZ
```

indicate that log4j.properties is not properly configured. Possible errors are:

- Log4j.properties cannot be located. Verify that the location specified in log4j.configuration matches with the location of the actual file.
- The logger for the service XYZ has not been configured.

Once log4j.properties is configured, these messages will go away and you will see messages in the designated log file.

6.2. Start WLS Node manager

. ozg_init.env $OZG_ORATAB_FRS11G2

cd $WL_HOME/server/bin

. setWLSEnv.sh

./startNodeManager.sh

6.3. Start WLS Configuration Wizard

. `ozg_init.env $OZG_ORATAB_FRS11G2`

Start `$WL_HOME/common/bin/config.sh`

You might run into the errors below:

Could not reserve enough space for object heap

Could not create the Java virtual machine.

In that case run the same command after issuing the following command:

```
export _JAVA_OPTIONS=-XX:MaxPermSize=512m
```

7. Appendix A - Installation of SIC_OOZWEBSERVICES

It is possible to use a synchronous version of the SIC_OOZWEBSERVICEJ web service.

The EAR file for the synchronous version is SIC_OOZWEBSERVICES.ear

The installation does NOT require setting up JMS queues.

Installation:

- Ensure you have completed the installation of the WebLogic domain as described in Ch 3.
- Deploy the SIC_OOZWEBSERVICES.ear, target to a managed server and start the web service

8. Appendix B - Functional Testing

8.1. Test SIC_OOZWEBSERVICEJ

It is possible to functionally test the SOAP/JMS interface with

Steps:

- Install HermesJMS
- Configure a session with the WebLogic host (config file needed)
- Select the jms/OOZWebServiceQueue
- Select Messages>Send Text Message and send an XML file with a request for SIC_OOZWEBSERVICEJ to jms/OOZWebServiceQueue
- Verify that the web service is responding by looking into the jms/OOZWebServiceResponseQueue and jms/oozErrorQueue.

Beware for the instructions above: replace OOZ by C2B if you use the English version!

8.1.1. Install HermesJMS

HermesJMS is a free tool to interact with JMS providers. It allows you to play back prepared SOAP messages to functionally test the SOAP/JMS interface.

For more information and to download the tool go to
http://www.hermesjms.com/confluence/display/HJMS/Home

Note that prior to installing HermesJMS you must create an environment variable JAVA_HOME which points to a JVM (version 1.5 or higher).

8.1.2. Create wlfullclient.jar for interacting with Weblogic

You will need client library files for testing the web service. The easiest solution is to create the client libraries on the web logic server.

- Go to $WL_HOME/server/lib on the weblogic server.
- Issue the following command to create a full client JAR file for interacting with WebLogic:

  java –jar wljarbuilder.jar

- Copy wlfullclient.jar to the computer running HermesJMS.

8.1.3. Configure a provider in HermesJMS

Start HermesJMS and open the preferences dialog.

Select the ‘Providers’ tab and add a classpath group named ‘weblogic’.

Add the wlfullclient.jar file to the library.

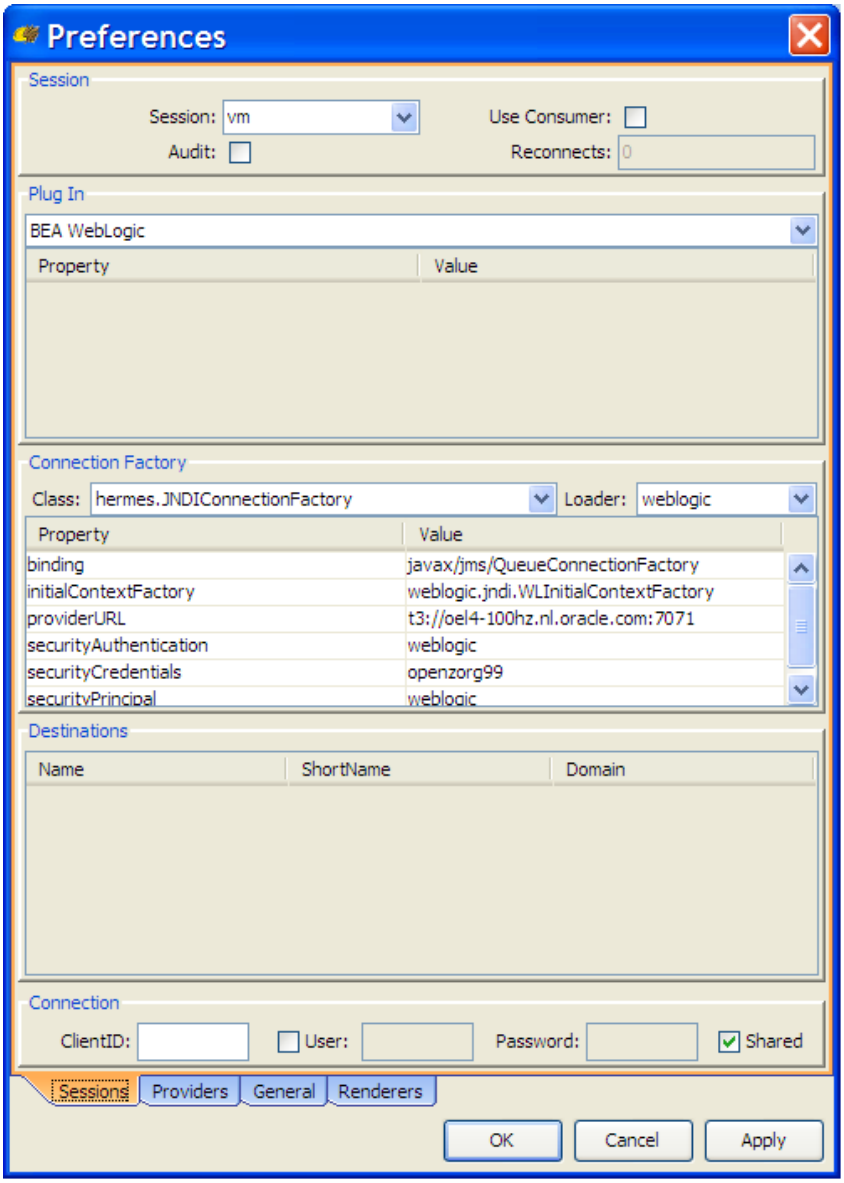
8.1.4. Configure a context in HermesJMS

Open the 'Sessions' tab in the preferences dialog.

- Enter a session name, eg. 'vm' or whatever has your fancy.
- Select the 'weblogic' loader.
- Select the BEA WebLogic plugin
- Select the hermes.JNDIConnectionFactory class
- Set the properties as indicated below:

Name	Value
providerURL	T3://<hostname><port> Hostname: managed server host Port number: managed server port number
initialContextFactory	weblogic.jndi.WLInitialContextFactory
Binding	javax/jms/QueueConnectionFactory
securityAuthentication	Weblogic
securityPrincipal	<administrator>, eg. weblogic
securityCredentials	<administrator_password>, eg. openzorg99

At this point the screen should look like this:



8.1.5. Discover the JMS queues for SIC_OOZWEBSERVICEJ

Right-click on the session and select 'Discover'.

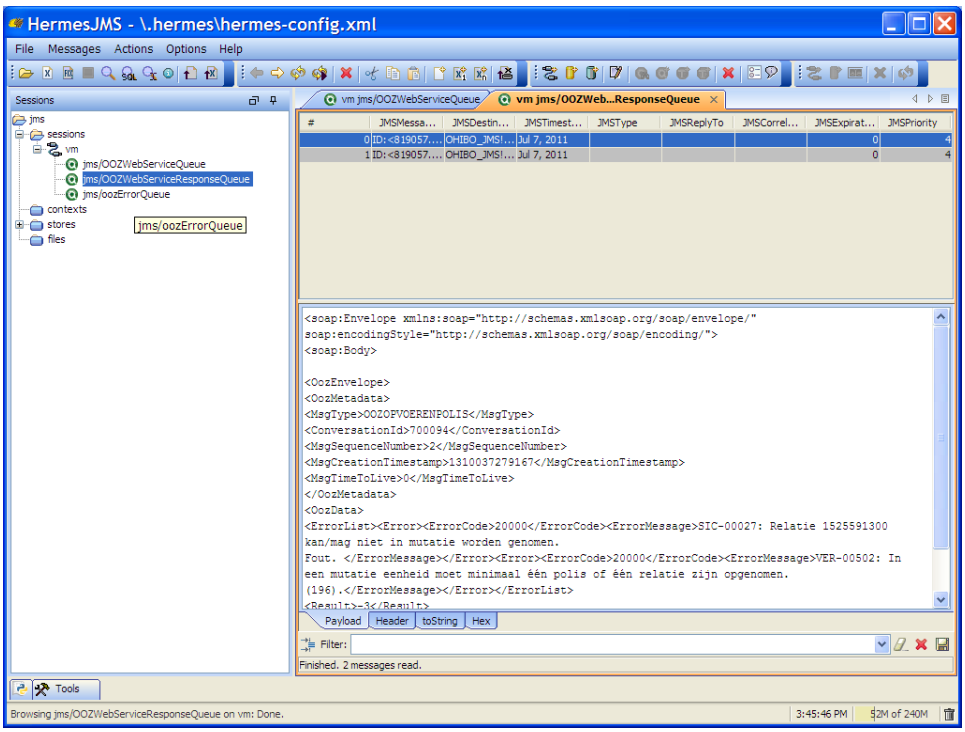
At this stage you will see the 3 queues which were set up for SIC_OOZWEBSERVICEJ.

8.1.6. Send test message to jms/OOZWebserviceQueue (replace OOZ by C2B if you use the English version!)

- Create an XML file containing a test message. You may use the sample message for this.
- Choose 'Send Text Message' in HermesJMS and select the file containing the test message. Once you click the 'Send' button, the message is sent to the SOAP/JMS service.
- Now double-click on the OOZWebServiceResponseQueue.
- Refresh the contents of this queue, until a message arrives from the web service.

- If no message arrives you may need to check the oozErrorQueue.

The screendump below shows a functional error message returned by SIC_OOZWEBSERVICEJ after processing the sample message:



8.2. Test SIC_OOZWEBSERVICES

You can functionally test the SOAP/HTTP variant of the SIC_OOZWEBSERVICEJ web service by taking the OozEnvelope part of an existing SOAP/JMS message and inserting it in a SOAP/HTTP request.

The whole process consists of the following steps:

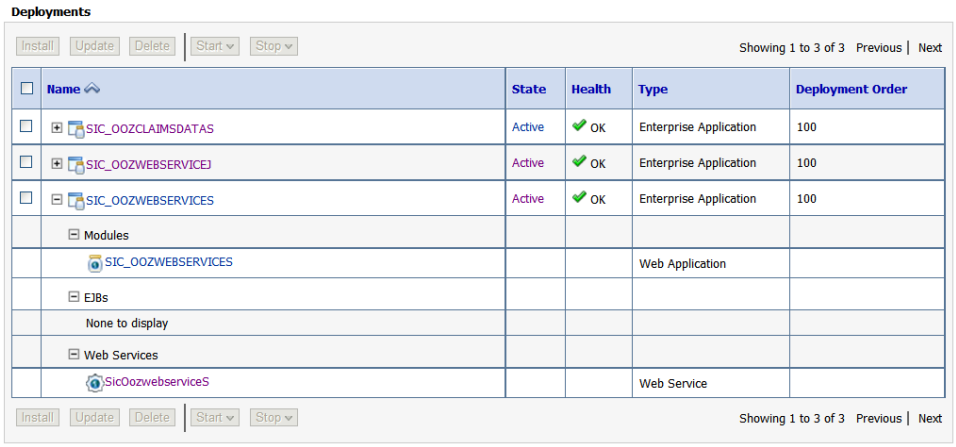
- Generate WSDL using WebLogic administration console
- Install SoapUI tool
- Create SoapUI project
- Create SOAP request
- Run SOAP request

8.2.1. Generate WSDL using WebLogic administration console

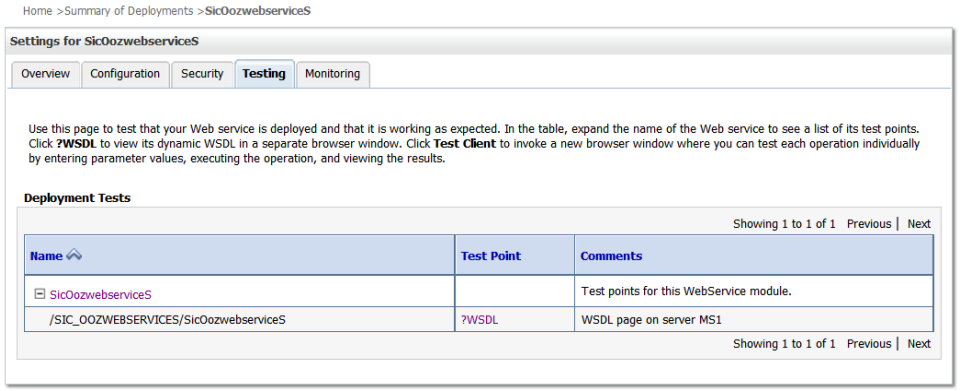
Start the WebLogic administration console.

Select the SIC_OOZWEBSERVICES web service in the deployments page.

Select the web service as shown below:



Click on the SicOozwebserviceS web service and choose the 'Testing' tab. Within this tab, expand SicOozwebserviceS like below:



Click on WSDL to generate the WSDL file.

Save as SicOozWebserviceS.wsdl.

8.2.2. Install SoapUI tool

soapUI is an open source tool for functional testing. You can download it from www.soapui.org.

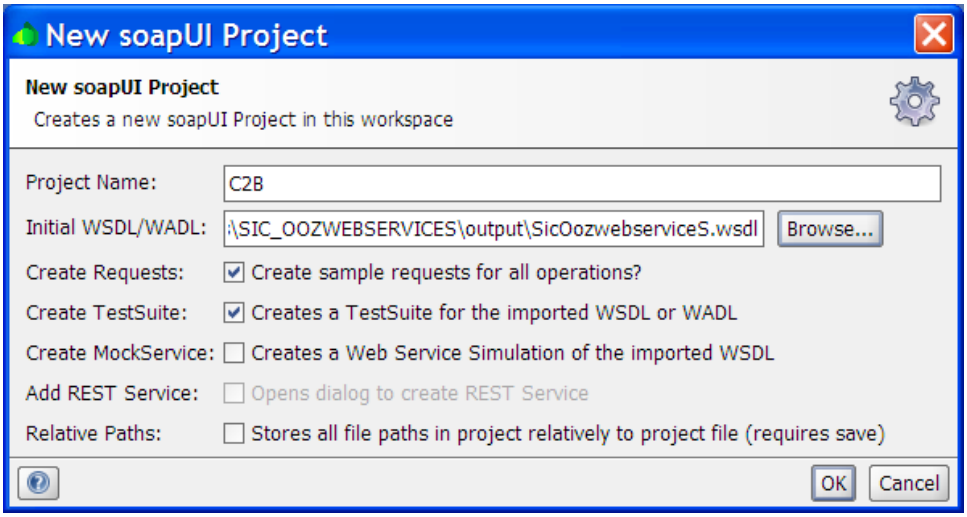
We have used version 3.5.1.

8.2.3. Create soapUI project

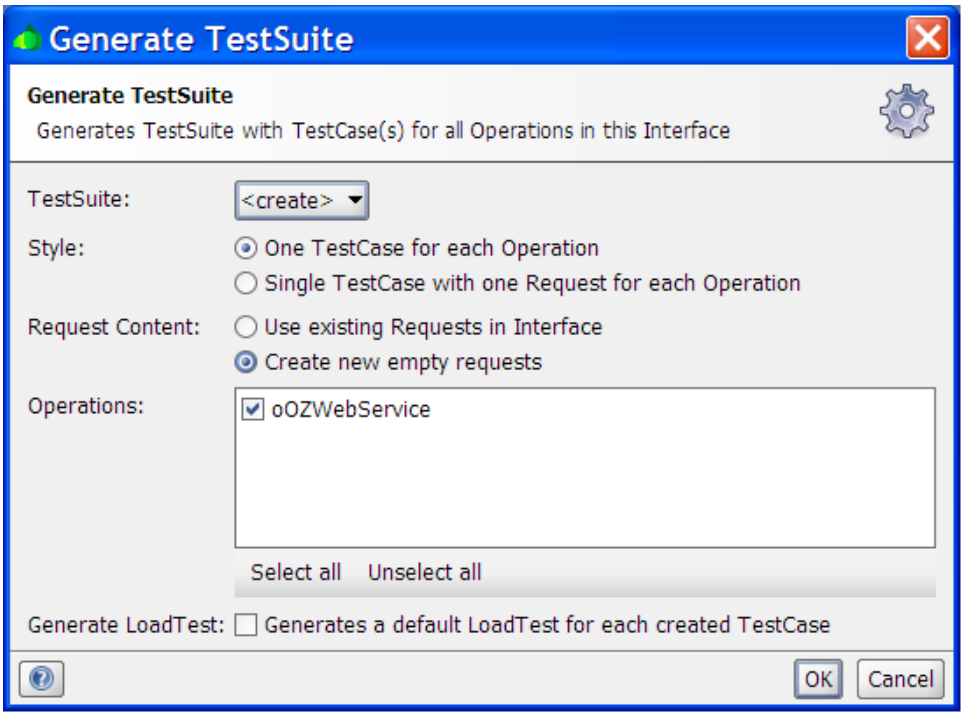
Start the soapUI tool, ignore timeout warnings.

Perform the following steps:

- Create a new project.
- Enter a project name (eg. C2B)
- Include the previously created WSDL file using 'Browse'
- Ensure that 'create request' and 'create testsuite' are checked.
- Click OK to create the soapUI project.



For the generation of the test suite, ensure to create empty requests:



Click OK to generate the project and test suite.

8.2.4. Create SOAP request

Use a text editor to view a SOAP/JMS message suitable for testing the SOAP/JMS variant of SIC_OOZWEBSERVICE. An example is given in the remainder of this chapter.

Select the contents of <OozEnvelope>, including the open and close tag for <OozEnvelope>:

<OozEnvelope>*Contents*</OozEnvelope>

Now wrap the selection in a CDATA fragment:

<![CDATA[<OozEnvelope>*Contents*</OozEnvelope>**]]>**

Open the C2B project in soapUI and right-click on Request 1 to start the request editor.

Replace the ? in <arg0>?</arg0> with the CDATA fragment. The request will now look like this:

The request (in the left window of the request editor) will look like this:

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:sic="http://sicoozwebservices.c2b.ohi.oracle.com/">
   <soapenv:Header/>
   <soapenv:Body>
      <sic:oOZWebService>
         <!--Optional:-->
         <arg0>
<![CDATA[<OozEnvelope>Contents</OozEnvelope>]]>
      </arg0>
      </sic:oOZWebService>
   </soapenv:Body>
</soapenv:Envelope>
```

8.2.5. Run SOAP request

Click the PLAY button in the top left of the request editor window to submit the request. Once the response is received, it will be shown in the right window of the request editor.

The response is a SOAP message which may contain a :

1. Confirmation from Oracle Health Back Office.
   Action: no action required.

2. Functional error message from Oracle Health Back Office
   Action: no action required.

3. Technical error message from Oracle Health Back Office
   Action: verify that the web version is compatible with the version of Oracle Health Back Office.

4. Technical error message indicating that the web service could not be reached
   Action: ensure that the web service is up and running at the host and port number which were specified in the WSDL.

5. Technical error message regarding the contents of the request.
   Action: edit the message until it is accepted as a valid request.

8.3. Sample message OOZWebService

The definition of the OOZWebService message payload itself that can be exchanged is documented in an XSD file.

The XSD files for the OHI Connect to Back Office web services can be found on the application server when the application server OHI software is installed.

In folder $OZG_BASE/xml the files SIC_OOZWEBSERVICE.xsd and SIC_OOZWEBSERVICE_EN.xsd define the structure of the messages for the Dutch respectively English version of the OOZWebService service.

8.3.1. Dutch sample message

Below a sample Dutch message is given.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
 soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<soap:Body>
<OozEnvelope>
  <OozMetadata>
    <MsgType>OOZOPVOERENPOLIS</MsgType>
    <ConversationId>700094</ConversationId><!--wijzig-->
    <MsgSequenceNumber>1</MsgSequenceNumber>
    <MsgCreationTimestamp>21129975</MsgCreationTimestamp>
  </OozMetadata>
   <OozData>
     <WebService>
       <OOZOpvoerenPolis>
            <BerichtKenmerken>
               <DatumIngang>2011-04-01</DatumIngang>
               <Merk>PBSTD</Merk>
               <Functionaris>RSULING</Functionaris>
               <Brondocument>7094</Brondocument><!--wijzig-->
               <Mutatiebron>AANM</Mutatiebron>
               <IndGereedMelden>N</IndGereedMelden>
               <InMutatieToegestaan>N</InMutatieToegestaan>
               <IndMutatieWooneenheid>N</IndMutatieWooneenheid>
               <IndAanpassenRelAdres>J</IndAanpassenRelAdres>
            </BerichtKenmerken>
            <Relatie>
               <Relid>
                  <RelVolgNr>1</RelVolgNr>
                  <BSNNr>700000161</BSNNr><!--wijzig-->
                  <GebDatum>1994-05-14</GebDatum>
               </Relid>
               <VestAdres>
                  <Adres>
                     <Land>NL</Land>
                     <PCNr>1112</PCNr>
                     <PCLetter>XH</PCLetter>
                     <HuisNrPostb>4</HuisNrPostb>
                     <HuisNrToev/>
                  </Adres>
               </VestAdres>
               <Naam>Volumetest1</Naam>
               <Vrltrs>T</Vrltrs>
               <Geslacht>1</Geslacht>
               <GebDatum>1994-05-14</GebDatum>
               <BSNNr>700000161</BSNNr><!--wijzig-->
            </Relatie>
            <Polis>
               <VerzNmr>
                  <RelID>
                     <RelVolgNr>1</RelVolgNr>
                  </RelID>
               </VerzNmr>
               <FinInfo>
                  <OntvWijzePrem>2</OntvWijzePrem>
                  <OntvWijzeDecl>2</OntvWijzeDecl>
```

```
                  <IncFreq>1</IncFreq>
               </FinInfo>
               <OvereenkDatum>2011-04-01</OvereenkDatum>
               <Contract>
                <CollNrExt>274</CollNrExt>
               </Contract>
            </Polis>
            <Dekkingen>
               <Verzekerden>
                  <Relid>
                     <RelVolgNr>1</RelVolgNr>
                  </Relid>
                  <DatumIngang>2011-04-01</DatumIngang>
                  <DatumAanm>2011-03-02</DatumAanm>
               </Verzekerden>
               <MPEId>181</MPEId>
            </Dekkingen>
            <RedenToetr>005</RedenToetr>
         </OOZOpvoerenPolis>
         <Transactie>OOZOPVOERENPOLIS</Transactie>
      </WebService>
   </OozData>
</OozEnvelope>
</soap:Body></soap:Envelope>
```

8.3.2. English sample message

Below a sample English message is given.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
 soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<soap:Body>
<OozEnvelope>
  <OozMetadata>
    <MsgType>C2BADDPOLICY</MsgType>
    <ConversationId>0-1611433041</ConversationId>
    <MsgSequenceNumber>1</MsgSequenceNumber>
    <MsgCreationTimestamp>1157707483046</MsgCreationTimestamp>
  </OozMetadata>
  <OozData>
    <WebService>
      <C2BAddPolicy>
        <MessageProperties>
          <StartDate>2012-01-01</StartDate>
          <Brand>PBSTD</Brand>
          <Officer>MSMALLEN</Officer>
          <IndApplyMod>N</IndApplyMod>
          <IndModHousehold>N</IndModHousehold>
        </MessageProperties>
        <Party>
          <PartyId>
            <PartySeqNo>1</PartySeqNo>
            <BSNNo>104949612</BSNNo>
            <DateOfBirth>1970-11-11</DateOfBirth>
          </PartyId>
          <Residence>
            <Address>
              <PCNo>2571</PCNo>
```

```
              <PCLetter>WG</PCLetter>
              <HouseNoPOBox>168</HouseNoPOBox>
            </Address>
          </Residence>
          <CodeType>
            <TypeCodeType>19</TypeCodeType>
            <CodeType>104949612</CodeType>
            <StartDate>2008-01-01</StartDate>
          </CodeType>
          <Name>M-1914 Verzekeringnemer 1</Name>
          <BSNNo>104949612</BSNNo>
          <Initials>B</Initials>
          <NameOrder>1</NameOrder>
          <Gender>1</Gender>
          <DoB>1970-11-11+01:00 </DoB>
          <MaritalStatus>2</MaritalStatus>
          <Origin>ZRG</Origin>
        </Party>
        <Policy>
          <PolicyHolder>
            <PartyId>
              <PartySeqNo>1</PartySeqNo>
            </PartyId>
          </PolicyHolder>
          <FinInfo>
            <PremiumAccount>
              <Account>
                <AccountNo>366718428</AccountNo>
              </Account>
            </PremiumAccount>
            <PremiumReceiptMethod>1</PremiumReceiptMethod>
            <ClaimsReceiptMethod>1</ClaimsReceiptMethod>
            <ColFreq>1</ColFreq>
          </FinInfo>
          <StartDate>2012-01-01</StartDate>
        </Policy>
        <Coverage>
          <Member>
            <PartyId>
              <PartySeqNo>1</PartySeqNo>
            </PartyId>
          </Member>
          <MPRStartDate>2006-01-01</MPRStartDate>
          <Package>PBSTD</Package>
          <PremStruc>STDPC</PremStruc>
          <CovStruc>STDDC</CovStruc>
          <YDStruc>STDER</YDStruc>
          <YDStep>ER0</YDStep>
          <ContrCare>PBNAT</ContrCare>
        </Coverage>
        <EntryReason>007</EntryReason>
        <PreviousInsurer>0701</PreviousInsurer>
      </C2BAddPolicy>
      <Transaction>C2BADDPOLICY</Transaction>
    </WebService>
  </OozData>
</OozEnvelope>
</soap:Body></soap:Envelope>
```