ORACLE
INSURANCE

Oracle Health Insurance Back Office

HTTP Service Layer (HSL)
Installation & Configuration Manual

Version 1.25

Part number: F37348-01

January 14, 2021

CHANGE HISTORY

Release	Version	Changes
10.16.2.2.0	1.0	• Creation
10.16.2.2.0	1.1	• Revision
10.17.1.0.0	1.2	• Changed grant instructions
10.17.2.0.0	1.3	• Documented hsl.<app>.developermode and hsl.developermode • Added reference to Doc[2] (Back Office HTTP Service Layer User Manual)
10.17.2.1.0	1.4	• Extended set of relevant properties.
10.17.2.2.0	1.5	• Minor revision of 'Creating a HSL database account' • Revised 'Security Configuration' • Removed 'Restricting access with custom roles' from Security Aspects • Renamed 'Security Aspects' to 'Additional Security Aspects' and revised contents. • Added 'Deployment Validation' • Added 'Appendix C – Testing with SoapUI' • Added 'Appendix D – Generating a WADL file' • Extended set of relevant properties for HSL_CLA.war deployment.
10.17.2.3.0	1.6	• Added paragraph 'Examining the Log File' • Added JDK version specific information regarding JSSE configuration.
10.18.1.0.0	1.7	• Added Appendix E – Authentication and Authorization • Added Appendix F – HSL_AUN and HSL_AUZ Services • Revised 2.1 including diagram • Revised introduction and document title
10.18.1.2.0	1.8	• Added Appendix G – PSL services • Use setUserOverrides.sh instead of modifying startManagedWebLogic.sh and Server Start arguments. Support for WLS 12.2.1.3.
10.18.1.3.0	1.9	• Added warning about patch **28278427**
10.18.1.3.0	1.10	• Revised installation of PSL services
10.18.1.4.0	1.11	• Updated 'Additional Security Aspects' Mentioned use of Basic Authentication for 'admin' requests. • Updated 'setting user context' in Appendix E • Added HTTP codes to 'Troublehooting'
10.18.1.4.0	1.12	• Updated HTTP codes overview in 'Troubleshooting' • Revised Appendix G – PSL services
10.18.2.0.0	1.13	• Corrected typo: hsl.auz, not hsl.aur. • Rewrote explanation of setUserOverrides.sh
10.18.2.0.0	1.14	• Services Components: added HSL_AUN and HSL_AUZ • WLS Preparation: fixed typos • Additional Security Aspects: removed remarks about patch 28278427 (already covered by certification matrix)
10.18.2.1.0	1.15	• Added example properties for new PSL service with shortname zvp • Added description of allowedorigins property to protect against cross site scripting attack
10.18.2.2.0	1.16	• Added additional deployment notes for HSL_C2B
10.18.2.3.0	1.17	• Added Appendix for HSL_JUP • Introduction of properties file templates
10.19.1.0.0	1.18	• No changes. Republished with different part nr.
10.19.1.4.0	1.19	• Added properties for logdirectory and logfilesuffix • The operation of developermode changed • Change in properties template is described • Updated description of data source creation
10.19.2.0.0	1.20	• Added description of deployment check verifyInterfaceVersion.
10.20.1.0.0	1.21	• No changes, republished.
10.20.3.0.0	1.22	• Adapted for impact of DB 19c and FMW12.2.1.4 certification
10.20.4.0.0	1.23	• Added new recommended Initial Capacity configuration option for data sources in Creating a Data Source paragraph
10.20.7.0.0	1.24	• Introduced HSLBOWS.ear and adapted deployment instructions and references to individual web service deployments. • Introduced Appendix I as an example of limiting access to web applications through the use of a security policy.
10.21.1.0.0	1.25	• No changes, republished with new part number.

RELATED DOCUMENTS

A reference in the text (**doc[x]**) is a reference to another document about a subject that is related to this document.
Below is a list of related documents:

Doc[1] Object Authorisation within OHI Back Office (CTA 13533)

Doc[2] Back Office HTTP Service Layer User Manual (CDO 15195)

Doc[3] OHI Back Office JET Application Installation & Configuration Manual (CTA13686)

Contents

1 Introduction

The OHI Back Office HTTP Service Layer is an optional component to provide so-called Use Case services.

Use Case services constitute a group of specific operations aiming to support use cases that are common for Dutch healthcare payers. Examples of typical use cases: requesting a new policy, adding an insured member, changing insured products, changing payment method etc.

OHI BO Use Case Services are implemented through the HTTP Service Layer (HSL).

The services in this service layer are based on RESTful Services technology which has the following advantages for current web application frameworks (like AngularJS and Oracle JET):

- accessible through HTTP (for example through Javascript)
- Supports (Javascript friendly) JSON as input and output formats
- standardized interface language through using HTTP verbs (GET, POST, PUT, PATCH, DELETE)
- standardized set of exceptions through HTTP error codes

This HTTP Service Layer is intended to ease integration in a Service Oriented environment.

This document describes the generic technical details regarding the HTTP Service Layer, how to install and update it and how to change configuration settings.

1.1 Licenses

Customers are required to have the appropriate license for using the HTTP Service Layer. Customers who have acquired a Connect to Back Office (C2B) license or an OHI SOAP ServiceLayer (SVL) license are currently permitted to install and use the web service component of the HTTP Service Layer. This is valid until further notice.

The corresponding PL/SQL services may not be used when no Connect to Back Office license, SOAP Service Layer license or HTTP Service Layer license has been obtained.

For further information please consult your OHI sales representative.

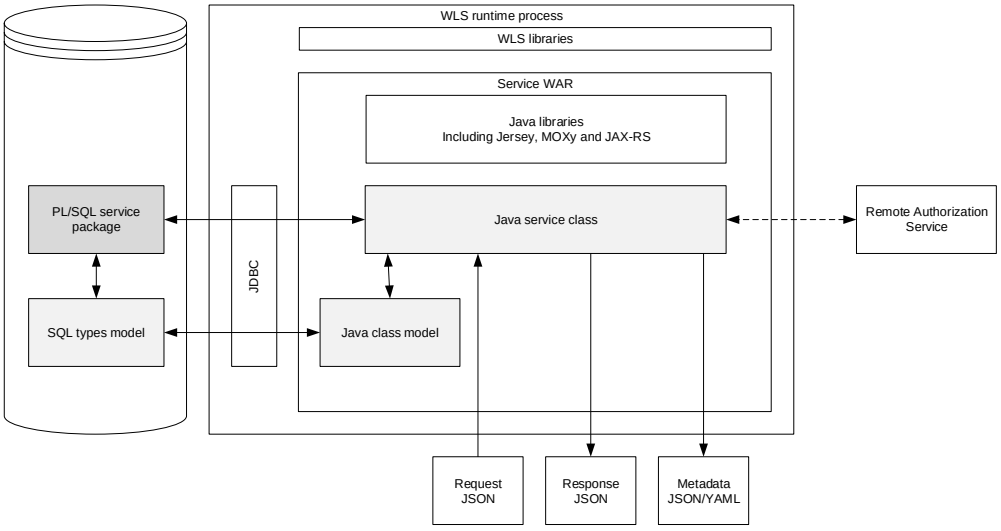
Attention: OHI Back Office release 10.20.3.0.x and 10.20.4.0.x are certified against Fusion Middleware 12.2.1.3.0 and 12.2.1.4.0. This document will assume an installation of Fusion Middleware 12.2.1.4.0. For installation of Fusion Middleware 12.2.1.3.0, see version 1.21 of this document, as delivered with OHI 10.20.1.0.0.

2 Architectural overview

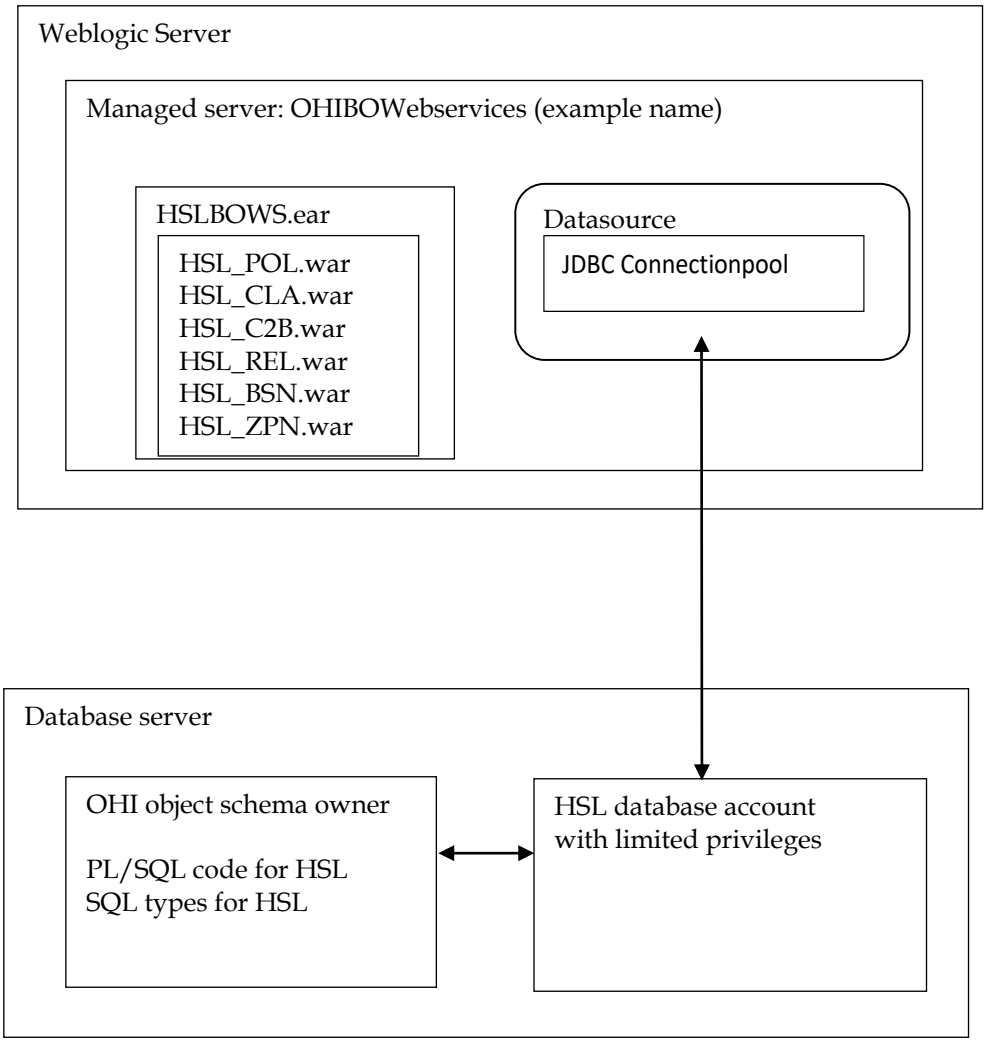
This chapter gives a high level architectural overview of the current HTTP Service Layer implementation.

2.1 Services components

The functionality of each service is implemented through a PL/SQL service package.
The service interface is provided through a Java layer.
Jersey, JAX-RS and MOXy are used to serialize and deserialize JSON objects and for input validation.
JDBC is used to map Java objects to SQL objects and vice versa.
The PL/SQL service package performs the required operations, using operation parameters and inbound objects to communicate with the OHI Back Office database.



The high level schema below shows how the services are deployed. It also shows the database connection to OHI which uses a database account with restricted access to execute the HSL service implementation in PL/SQL.



3 Installation of HSL services

This chapter describes the steps to (re)install the HSL services.

This chapter contains the following parts to separate the various work areas:

- Sizing/load aspects
- Database installation
- WLS preparation
- Security configuration
- (Re)deployment of the HSL application
- Additional Security aspects

3.1 Terminology

Note the following use of terminology:

- HSL stands for HTTP Service Layer. The underlying technology is based on RESTful service technology.
- A HSL service has one or more service operations.
- Each HSL service resides in its own HSL application.
- The HSL applications are packaged together as an EAR file, which is deployed to the WebLogic application server.

3.2 Sizing/load aspects

From the "Introduction" and the "Architectural overview" chapters it should be clear that the HSL services are implemented through PL/SQL in the database.

The Java layer providing the REST interface handles request and response messages. It validates an incoming request, calls the PL/SQL service implementation to perform the required operation and transforms the result into a response message.

This choice means that the larger part of the processing is carried out on the database server and only a small part is handled on the application server.
Since the architecture for HSL is similar to the SVL services, the distribution of loads on the application and database server is expected to be comparable.
Based on the SVL services it may be assumed that for heavy processing only 1 CPU thread will be busy processing HSL service requests if 10 CPU threads are needed for the database processing for these requests.

Based on SVL experience, most of the simpler service operations on a well-sized and well-performing production environment should not take more than 0.1 up to 0.5 second in total elapsed time when measured on the WebLogic Server. Of this elapsed time most of the time should be spent by the database server handling the call, as mentioned before.

More complicated calls and service calls that return large data sets may take more time but usually should not exceed response times of more than a few seconds. As an example, if it would be offered, a typical premium calculation call should be executed

within a second and a large set of claim lines (several hundreds) should usually be returned within 5 to 10 seconds.

3.2.1 Deployment choices

The overall load on the OHI application resulting from HSL service calls is customer specific and may change over time.

HSL services are likely to be used by customer-facing applications. Although it may technically be possible to deploy HSL services to the application server running the Forms GUI for internal users, you should be aware of the peak loads from HSL services during commercial campaigns. These loads may well exceed your normal capacity. You should devise your own strategy to cope with these extra loads. This strategy may include using separate application servers for internet users, using a separate database with cached data for information requests, throttling inbound requests, etc.

If you choose to install HSL services on the application server for the Forms GUI it is advisable to actively monitor the respective loads of Forms processes, SVL processes and HSL processes. This allows you to pick up trends to help you refine your infrastructure strategy.

Especially if you have multiple applications using the same HSL services, it may help to use a service bus to create a 'separation of concerns'. The service bus allows you to map the HSL interface specification to a customer-specific interface which means less maintenance on the client applications when deploying a new version of a HSL service. As long as the mapping on the service bus is synchronized with the HSL service interface, the code client applications can remain the same.

Stringent requirements for high availability and failover are also reasons to consider a service bus as a go-between.

3.3 Database installation

All database components of the HTTP Service Layer are owned by the OHI Back Office schema and are installed through the OHI Back Office release installation procedure.

To use the database components of the HTTP service layer, one or more database accounts must be created with HTTP Service Layer access privileges.

Before creating the account(s), check if you are licensed to use the HTTP Service Layer.

Please check if you have a database object (package) HSL_UTIL_PCK in the OHI Back Office schema. If not, something went wrong regarding the installation of the HTTP Service Layer code. If this is incorrect please contact the OHI Support department.

If the package is present in your database you can continue with the database part of the installation.

3.3.1 Creating a HSL database account

The OHI Back Office schema owns the PL/SQL code to implement the HTTP Service Layer but may not be used to execute the services.

The use of a separate database account to access the HTTP Service Layer components reduces the risk of accessing unauthorized OHI data and makes that account accountable for HSL actions. The HSL account(s) need a minimum of object privileges to the HSL database objects.

One or more HSL accounts can be created:

- Default HSL account for use with WebLogic application server
  This account is configured in the HSL properties file as the default account for HSL service requests.

- Optional additional HSL accounts for use with WebLogic application server. These may be configured in the HSL properties file for one or more specific services.

- Additional HSL account for use with bespoke PL/SQL code development by the customer. Please follow the directions in Doc[1].

The following steps are needed to setup a HTTP Service Layer database account:

1. Create a schema owner, for example HSL_USER. Determine the password policy, temporary tablespace, etc. according to your company standards but beware there is no interactive login which might show expiration messages for the password due to the enforced password policy.

2. Grant create session system privilege to this account.

3. Grant the HTTP Service Layer object privileges: logon as the OHI Back Office schema owner, enable server output, and run

```
alg_security_pck.HSL_grants
(pi_owner   => '<ohibo_owner>'
,pi_grantee => '<hsl_user_account>'
)
```

   Example:

```
execute
alg_security_pck.HSL_grants
(pi_owner   => 'OZG_OWNER'
,pi_grantee => 'HSL_USER');
```

   IMPORTANT: This command needs be run only once. While installing subsequent OHI BO updates, the privileges of the HSL user accounts are automatically updated.
   However, if you run into ORA-01403 errors during an execution your first check should be to run this command in sqlplus, enabling server output before running, and see whether missing grant privileges were granted.

3.4 WLS Preparation

When the database account has been created and granted successfully, a WebLogic Server environment (software home) must be prepared for deploying the HSL application.

We expect that you are familiar with the WebLogic concepts like 'Domain', ' Managed Server', 'Cluster', etc.

These are your options:

- Use the same WebLogic environment which is used for servicing the OHI Back Office user interface and batches. In this case you should create a new WebLogic domain (with a new Admin Server) for the HSL applications to prevent interference with the GUI application.

- Deploy the HSL applications in a separate WebLogic environment (possibly on a separate server). This allows you to separately upgrade or patch the different WebLogic environments, or implement a workload distribution.

  Deploy HSL applications to multiple environments for better scalability. Be sure to deploy each HSL application only once in a Managed Server or a cluster of Managed Servers.

- For testing purposes you may want to have multiple versions within the same domain. In that case you should have a separate Managed Server for each deployment.

Some remarks about installing in a separate WebLogic environment:

- The OHI Back Office GUI application (Forms) installation requires a WebLogic Server "Infrastructure" installation. That means the domain created for Forms needs to have its own database schemas with OPSS and Audit database tables (created by RCU). For the HTTP Service Layer domain these schemas are not required provided you do not select more components during the domain configuration than described.

- When installing in a separate WebLogic Server environment, use a different Installer: use the "Generic" installer instead of the "FMW Infrastructure" installer. When installing in a separate WebLogic environment make sure the correct components are installed when creating the Domain. You need at least:

  ```
  o Weblogic Advanced Web Services for JAX-WS Extension -
    12.2.1.4.0 [oracle_common]

  o Weblogic JAX-WS SOAP/JMS Extension                  -
    12.2.1.4.0 [oracle_common]
  ```

  When you have not installed these components your web services will respond with 'There are error messages.' All info in the functionalFaultType will contain question marks (???).

The instructions in the following paragraphs cover the setup of a new domain including the setting up of Managed Servers, a machine definition, data sources, etc.

This will support the following scenarios:

- ✓ Creating a separate domain with a single Managed Server
- ✓ Creating a separate domain with a cluster of 2 Managed Servers
- ✓ Adding a Managed Server to an existing domain

Using setUserOverrides.sh

Traditionally, Server Start arguments for the WebLogic Managed Server had to be added in multiple locations (depending on the way the Managed Servers were started):

- Via the Admin console: in the tab "Server Start" in the field "Arguments" for Admin Servers and Managed Servers
- In the script $DOMAIN_HOME/bin/startManagedWebLogic.sh for Managed Servers
- In script $DOMAIN_HOME/bin/startWebLogic.sh for Admin Servers

WebLogic 12.1.2 introduced a better way to pass Server Start arguments to the WebLogic servers. See document “How To Customize Env Parameters Via 'setUserOverrides.sh' File (In WLS 12.1.2.0.0 ~ 12.2.1.3.0) (Doc ID 2138183.1)” on My Oracle Support for details.

By using setUserOverrides.sh the Server Start arguments only need to be maintained in one place. This document will assume you use this new method and it will only give instructions for setting the Server Start arguments using this method. The file is located here:

$DOMAIN_HOME/bin/setUserOverrides.sh

3.4.1 Requirements

The following requirements/limitations must be taken into account:

- ✓ A certified WebLogic Server version including JAX-WS (SOAP/JMS) extensions. The HSL services must be deployed on a single Managed Server or a cluster of Managed Servers (the ‘target’).
- ✓ The HSL services may not be deployed on a Managed Server which is also used for hosting the OHI GUI application (Forms). The Managed Server may not belong to a cluster used for deploying the GUI application.
- ✓ One deployment can only service one single OHI Back Office environment (it connects to a specific connection pool which accesses a specific OHI Back Office ‘instance’).

If the HSL application must be deployed more than once (for servicing different OHI Back Office environments) each deployment should be on its own Managed Server or Cluster.

HSL can be deployed on the same Managed Servers as SVL.

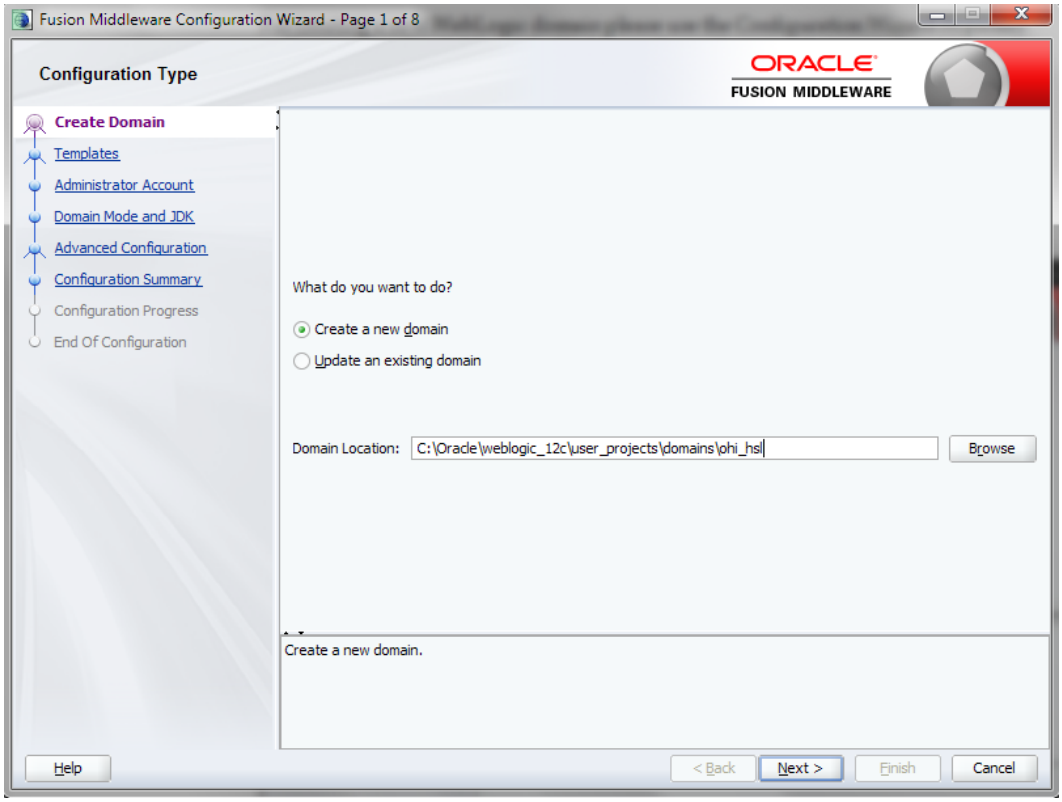
3.4.2 Creating a domain

Before creating a Domain, be sure to understand the difference between a “FMW Infrastructure” and a “Generic” WebLogic installation, and the consequences. Make sure the environment variable DOMAIN_HOME is not set.

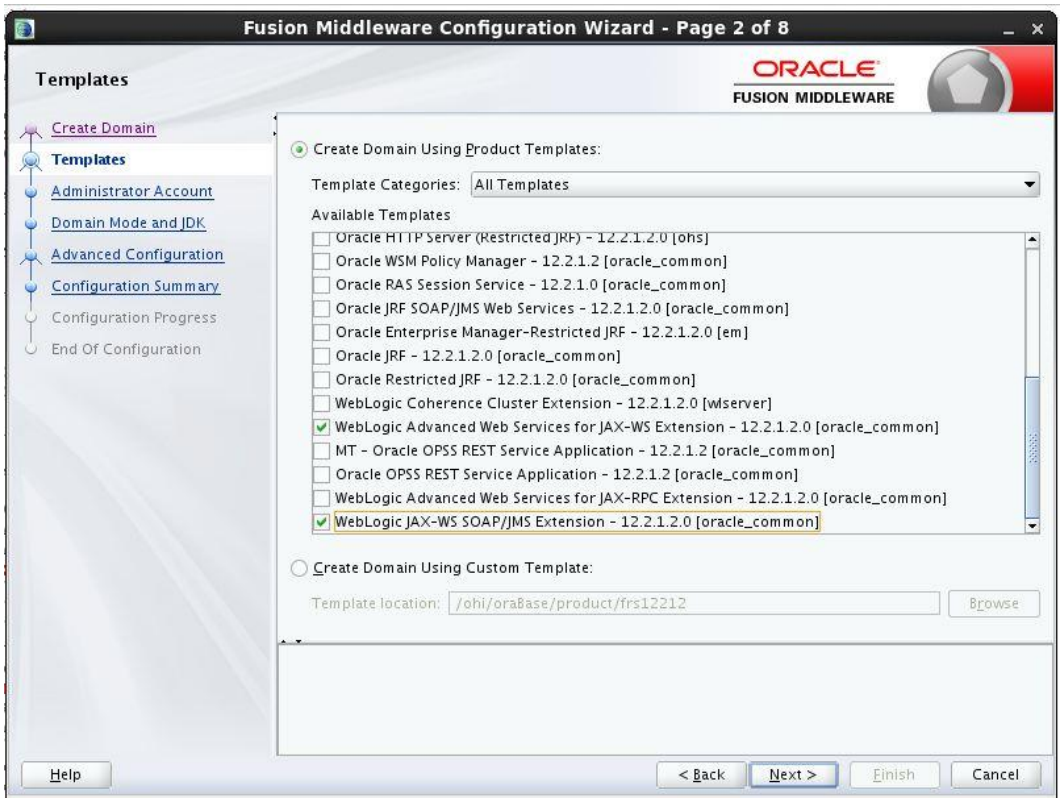
If you create the new Weblogic Domain from the same software home as the Forms Domain, you have to choose the same “Domain Mode” (Development or Production), to avoid errors during startup of the new Managed Server(s).

For creating a new WebLogic domain please use the Configuration Wizard (typically in the common/bin folder of the WebLogic Server home, so for example `$MW_HOME/oracle_common/common/bin/config.sh`)

Specify the domain location. This is inside the Weblogic Home by default, but you can specify a location outside the WebLogic Home. The last part of the location will be the Domain Name.



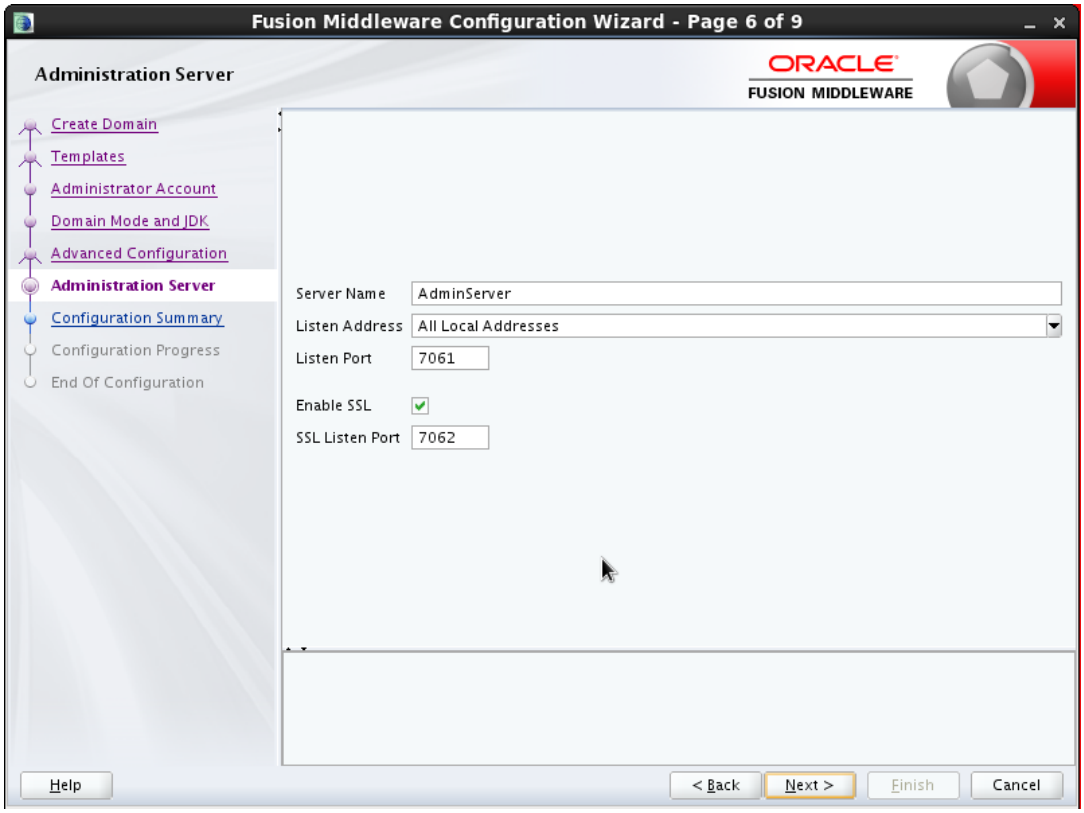
When creating a new domain select at least the options as shown below.



In the next screens, specify the *username* and *password* for the domain administrator account. When prompted for developer or production mode choose *production mode* and pick a JDK.

In this documentation we choose to configure only the Administration Server using the wizard. The Administration Server can be used as the starting point for additional configuration options you may want to choose later:

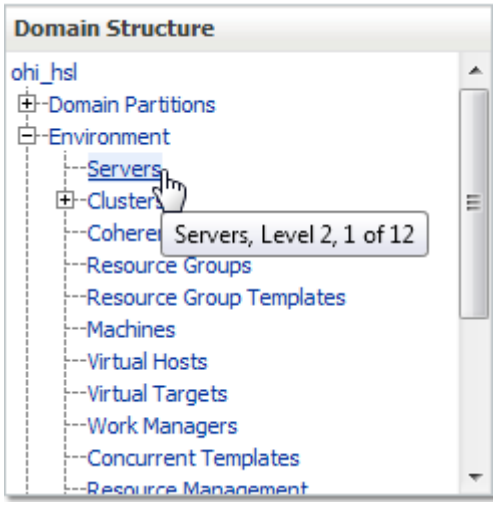
For the Administration Server a free port number must be specified. Enable SSL to support secure connections. An example using non default ports is shown below.



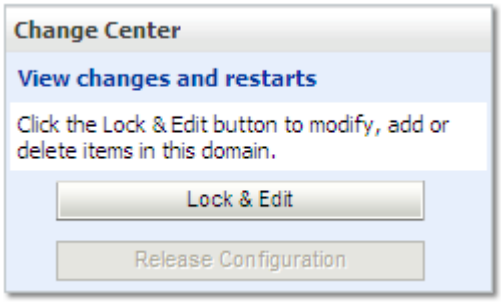
3.4.3 Creating Managed Server(s)

Start the Administration Server (of the existing or newly created domain) using the startWebLogic.sh script (this is present in the root folder of the domain folder, which you created through the Configuration Wizard).

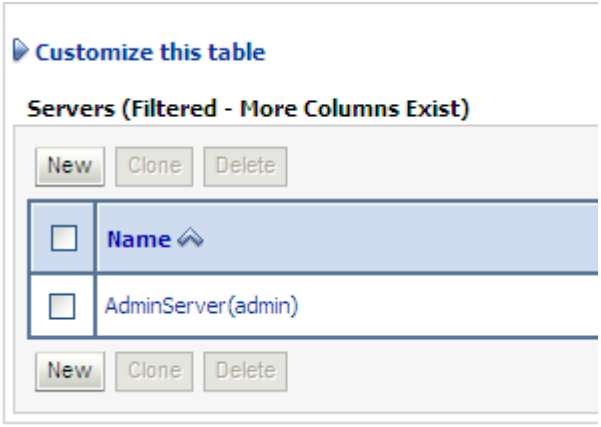
After it has started, logon to the console and choose the Servers option in the left panel:



In the Change Center choose Lock & Edit to get into editing mode.



This enables the New option in the 'Summary of Servers' overview:



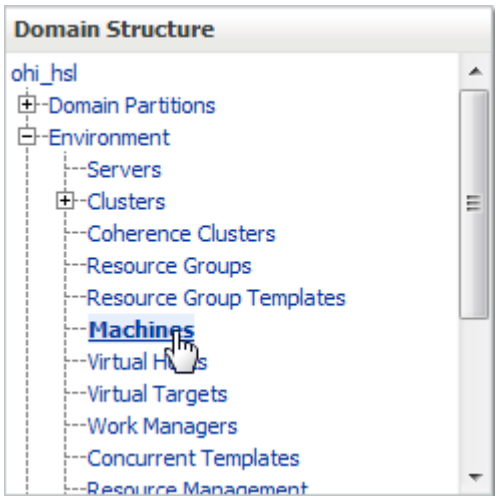
You need to provide a name and listening port for the Managed Server. For easy reference you may want to include the domain name in the name of the Managed Server, for example 'ms_ohi_hsl'.

At this point you should decide whether or not to make the Managed Server part of a Cluster.

If no Cluster exists you can create one; if there is an existing Cluster you can make the Managed Server a member of the Cluster.

3.4.4 Creating a machine definition

It is recommended to create a machine definition to make it easier to start up Managed Servers:



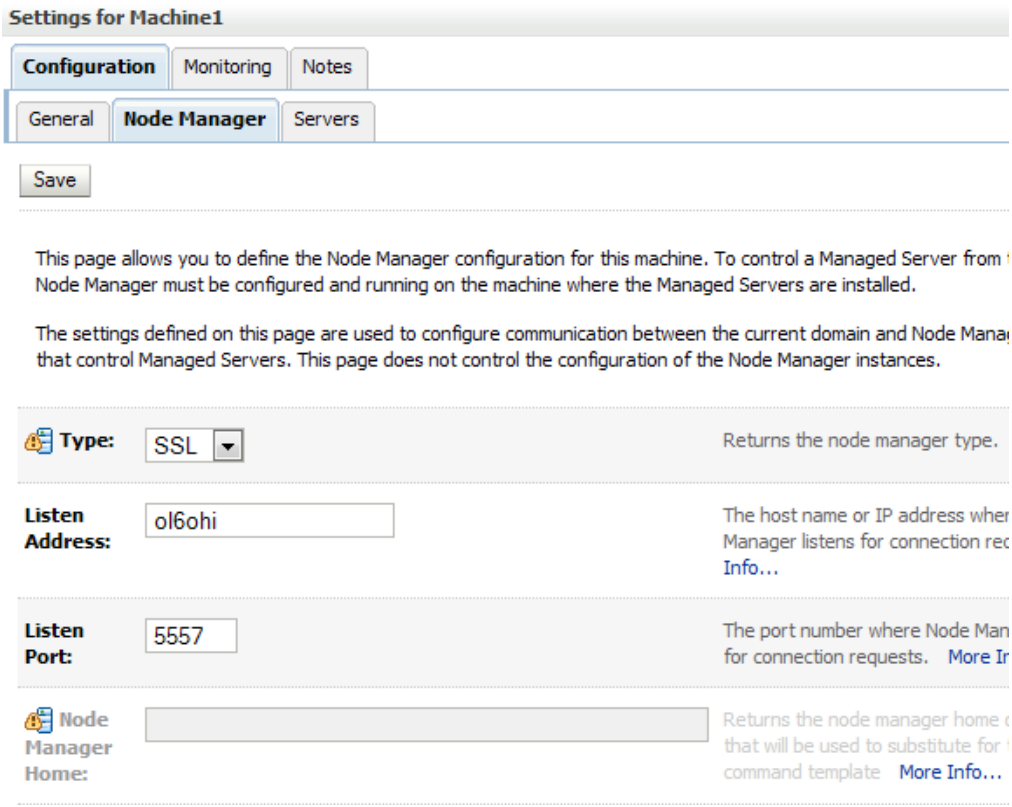
You can now assign Managed Servers to the new machine definition. In the example below Managed Server ms_ohi_hsl is assigned to Machine1.

☐	ms_ohi_hsl	Configured		Machine1

If you start a Node Manager you can use the console to start the Managed Servers.

You need to associate the machine with the Node Manager so that the Node Manager can start the Managed Server within the domain of the machine definition.

Do this in the Node Manager tab for the machine definition like in the example below:



Make sure the listen address is the actual listen address that is used by the Node Manager. This is passed as first parameter to the `$WL_HOME/server/bin/startNodeManager.sh` shell script. The correct value can be found as ListenAddress in the file nodemanager.properties.

This address can be changed in the file nodemanager.properties which is located in the <domain home>/nodemanager folder. This is necessary when you have a node manager per domain.

You need to create a boot.properties file for the new Managed Server for the domain in the domain home Managed Server ../data/nodemanager.

This is done automatically when you start the Managed Server in the console (after you have started the AdminServer for the domain).

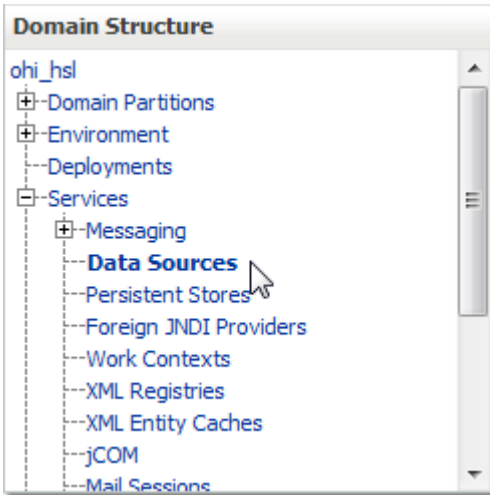
When you are running in Development Mode, a boot.properties file is automatically created for the AdminServer.

Because you are running in Production Mode, you need to create the file yourself, in the $DOMAIN_HOME/servers/AdminServer/security folder. This file is used when the AdminServer is started by the script startWebLogic.sh. If the file is not present, the script prompts for the username/password. The same goes for the Managed Servers when you start them through a script.

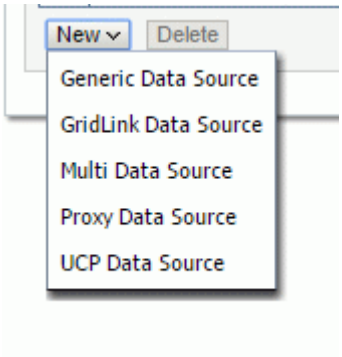
3.4.5 Creating a data source

The HSL application needs a data source to connect with the OHI Back Office database.

To create a data source, navigate in the Domain Structure panel on the left to the data sources option. Choose 'Lock & Edit' so you are able to create a new data source.



Create a new 'Generic data source' in case of a 'regular' database instance:



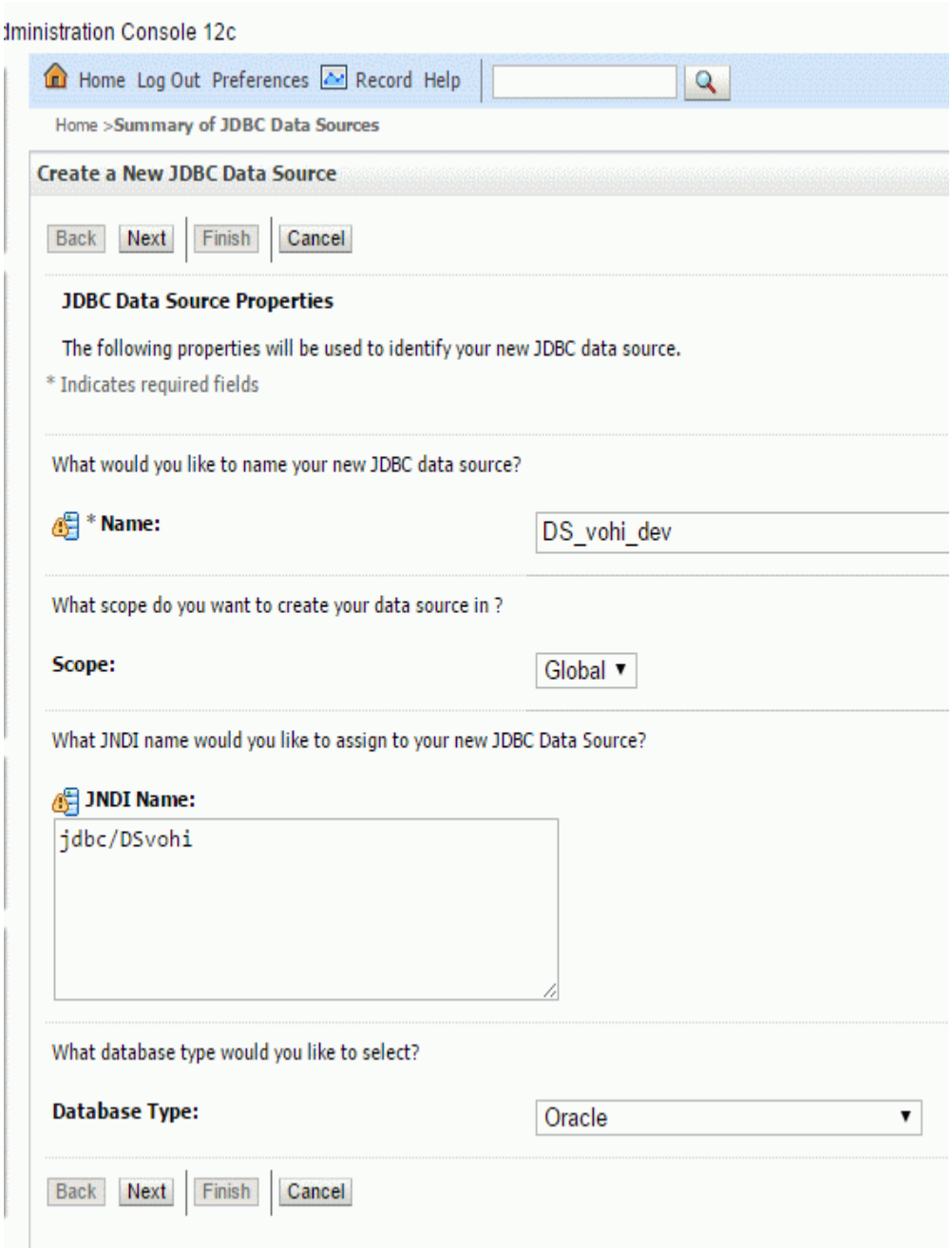
When the data source connects to a RAC data source it is more useful to choose for 'GridLink Data Source' as this can respond to instance state changes.

Next, choose a name for the data source to reflect its purpose. For example, you may want to reference the database name: DS_OHI_prd.

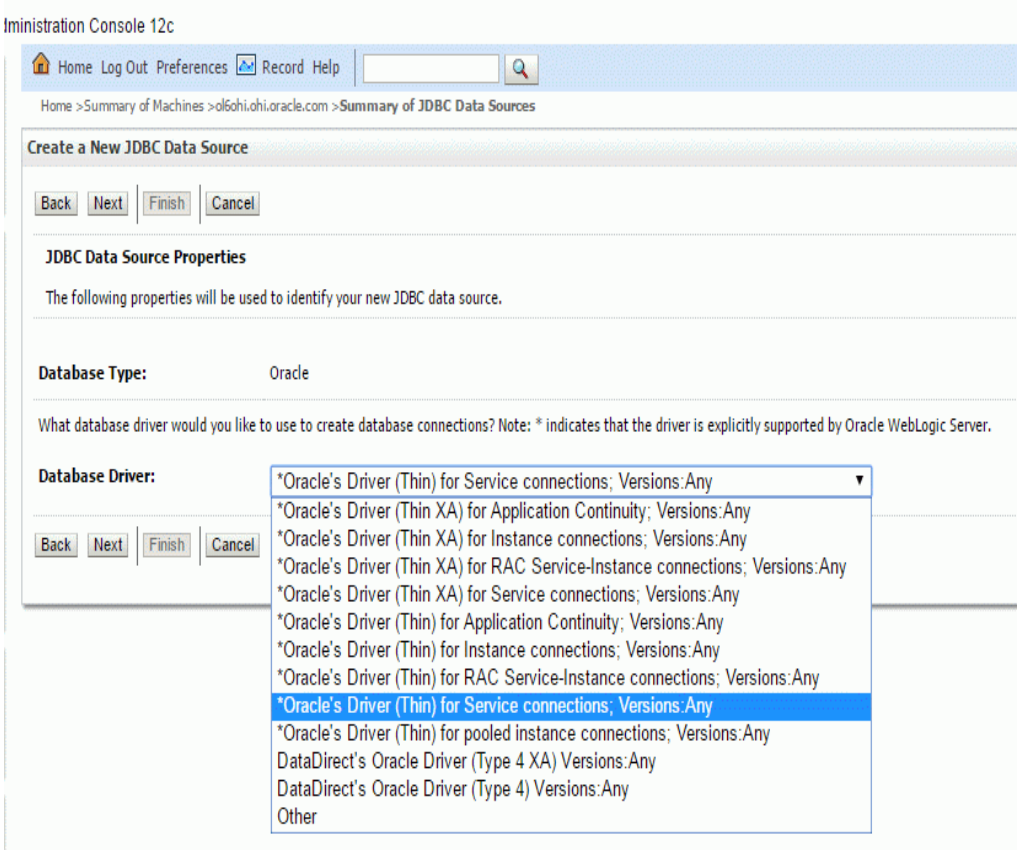
Next specify a JNDI name. The JNDI name will be used in the properties file for starting the HSL application.

Specify 'Oracle' as the database type (for GridLink this is a predefined value).

An example:



Next you need to specify a database driver. For regular database instances, use "Oracle's Driver (Thin) for Service connections; Versions: Any".



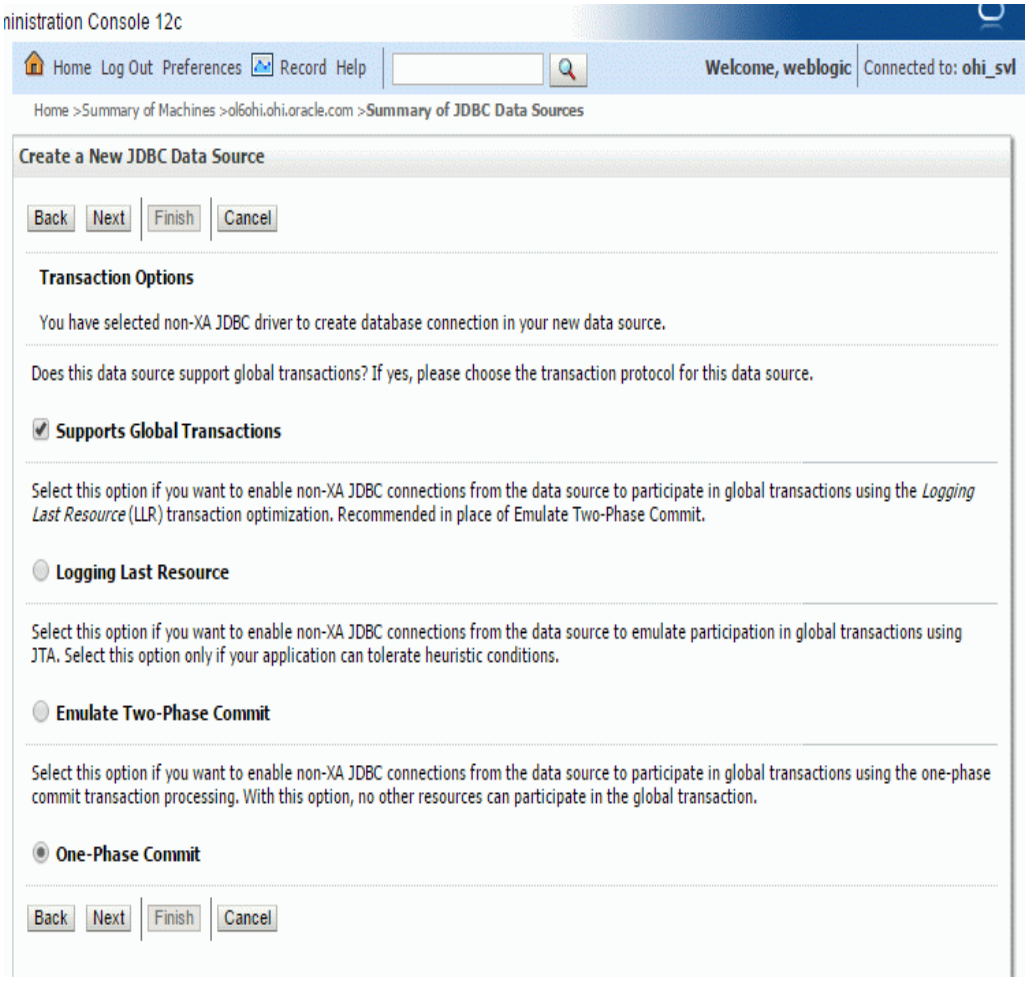
If you are using RAC (or considering to use RAC) choose the thin RAC driver if you created a regular data source (we do not advise this anymore but in previous versions of this manual this was the usual option). Do not use the XA driver.

When you have chosen to create a GridLink data source use “Oracle’s Driver (Thin) for GridLink Connections, Versions: Any”.

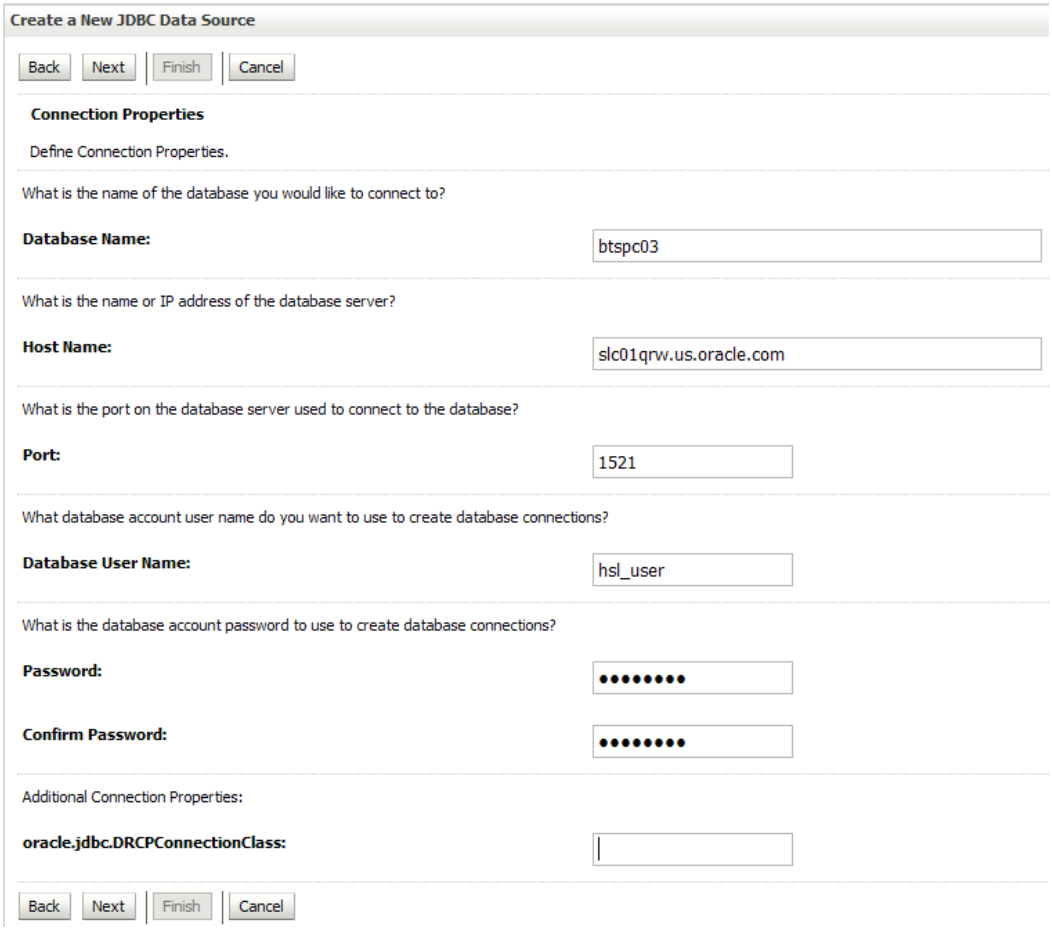
Choose the following Transaction Options:

- ‘Supports Global Transactions’;
- ‘One-Phase Commit’ (this is why you don’t need the XA driver)

Example:

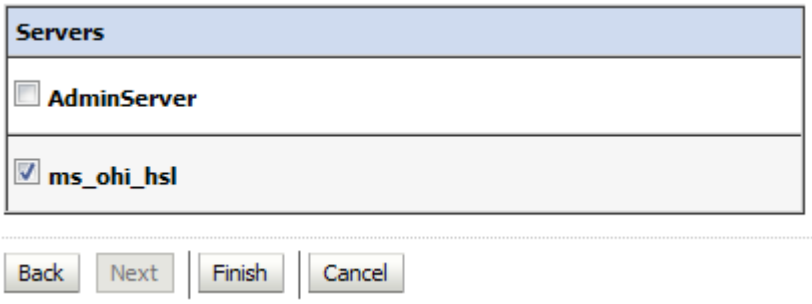


Next specify the connection details like the example on the page below. Be sure to use values which are valid for your environment.



On the next page the result of your answers will be shown. You can test the connection with the data shown (the table name is not relevant).

When you navigate to the next page you can select the targets where the data source should be deployed to. In the example below only the Managed Server shown will be used for deploying the data source to.



Press Activate Changes to conclude your configuration.

At this point, go back to your data source and re-open the connection pool tab.

Initial Capacity setting

Consider setting the 'Initial Capacity' to 0. During the setup of new connections a health check, if you configured this (for more information see the OHI Release Installation manual), claims a shared lock that might stall (patch) sessions and vice versa.

Setting this option to zero implies no connections are set up after the connection pool is initialized but only on demand. Press Lock & Edit and set the option to 0 and press Save.

Wrap Data Types setting

Navigate to the 'Advanced' part.

Ensure that the option 'Wrap Data Types' is unchecked. This setting is needed for passing CLOB objects to and from the database and when activated slows down execution. Uncheck this option and press Save and Activate the change.

Example:

3.5 Security Configuration

All HSL applications are preconfigured to use basic authentication and SSL encryption.

The following steps are needed to set up minimal security for the HSL application:

- Set up security realm
- Setup Weblogic user for accessing HSL application
- Enable SSL
- Configure key store
- Configure logging level
- Configure user lockout

Before you can install HSL applications, you need to decide on the security model you want to use. The options are:

1. Use the predefined default user restuser and the predefined policy. During deployment, choose the security model indicated with 'DD Only'.

2. Use the default policy and create your own roles to restrict access to the web services. During deployment, choose the security model indicated with 'Custom Roles'

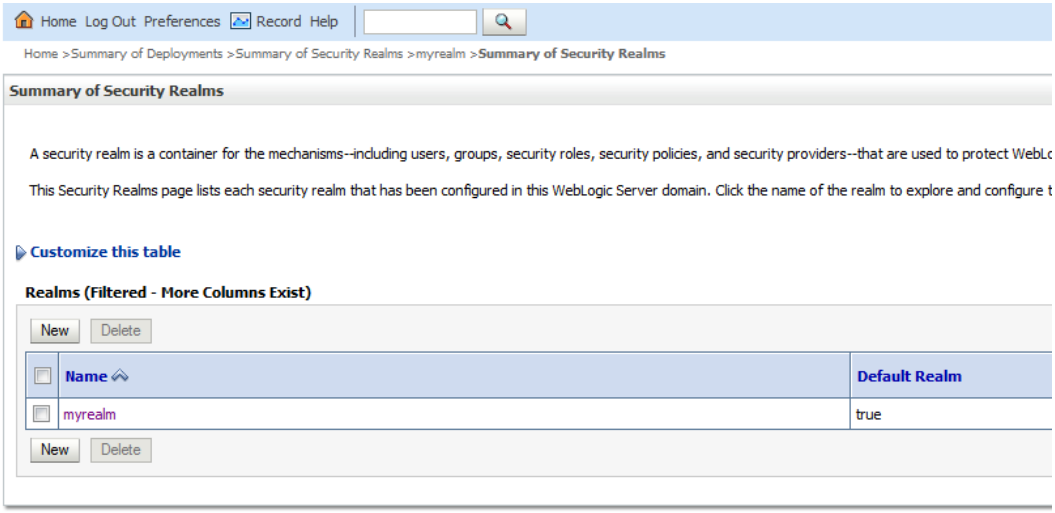
3. Use a custom security model to overrule the default of each web service. See paragraph 3.7 "*Additional Security Aspects*". During deployment, choose the security model indicated with 'Custom Roles and Policies'.

If you want to use OAUTH 2.0 token authentication and validation (as an alternative to Basic Authentication) you need to choose 'Custom Roles and Policies'. See *Appendix E – Authentication and Authorization* in this document for details.

3.5.1 Set up security realm

Create a security realm if this has not already been done (normally realm 'myrealm' will already be present).

The security realm ‘myrealm’ as shown below will be used to configure the security at application level.



If there are no other security realms, this will be the default security realm.

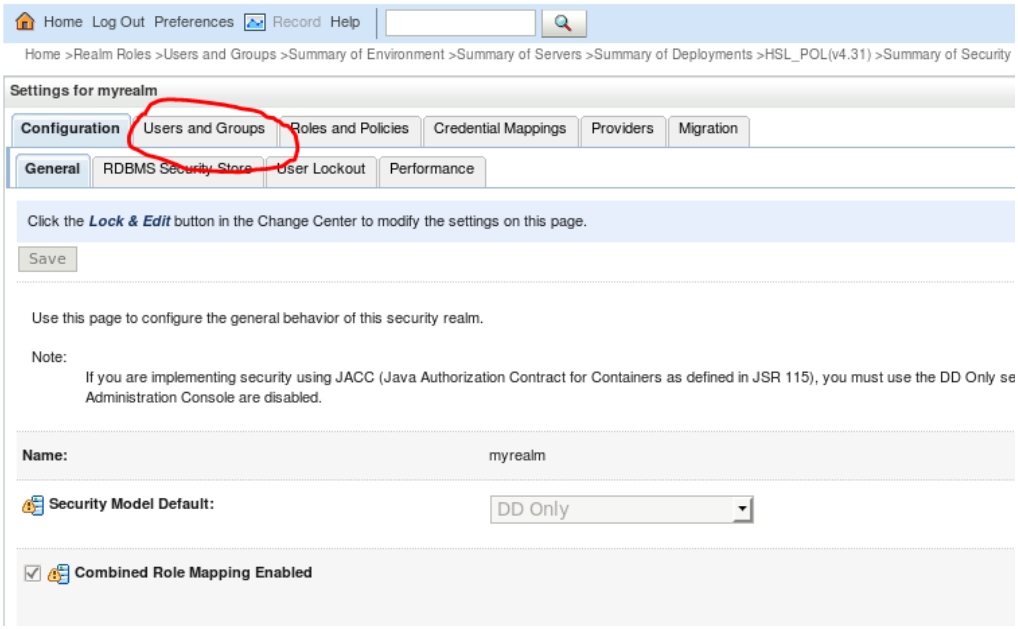
3.5.2 Setup Weblogic user for accessing HSL application

Each operation of an HSL application requires basic authentication. This means that each call must be made as an authenticated Weblogic user.

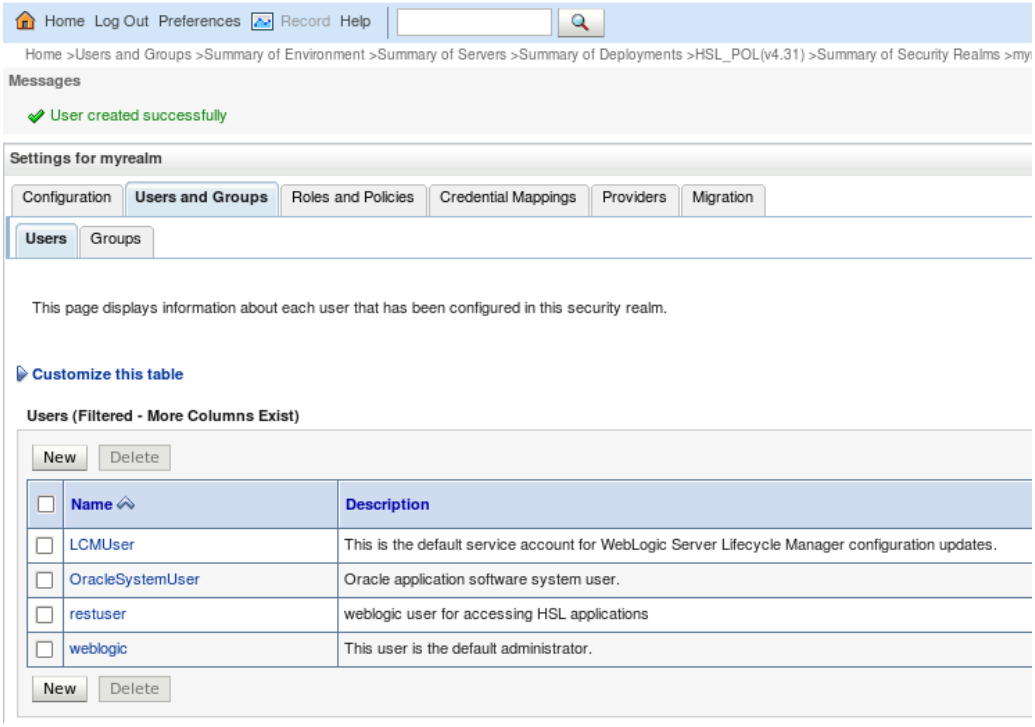
The name of the default Weblogic user to access the HSL application is defined in a preconfigured deployment descriptor in the WAR file. This default name is ‘restuser’.

So if you deploy the HSL application with the default security model (DD Only) you will need to create the Weblogic user ‘restuser’ and authenticate as ‘restuser’ when invoking the HSL application.

To set up the Weblogic user ‘restuser’, select ‘Users and Groups’ for your security realm:



Add ‘restuser’ to the pool of Weblogic users:

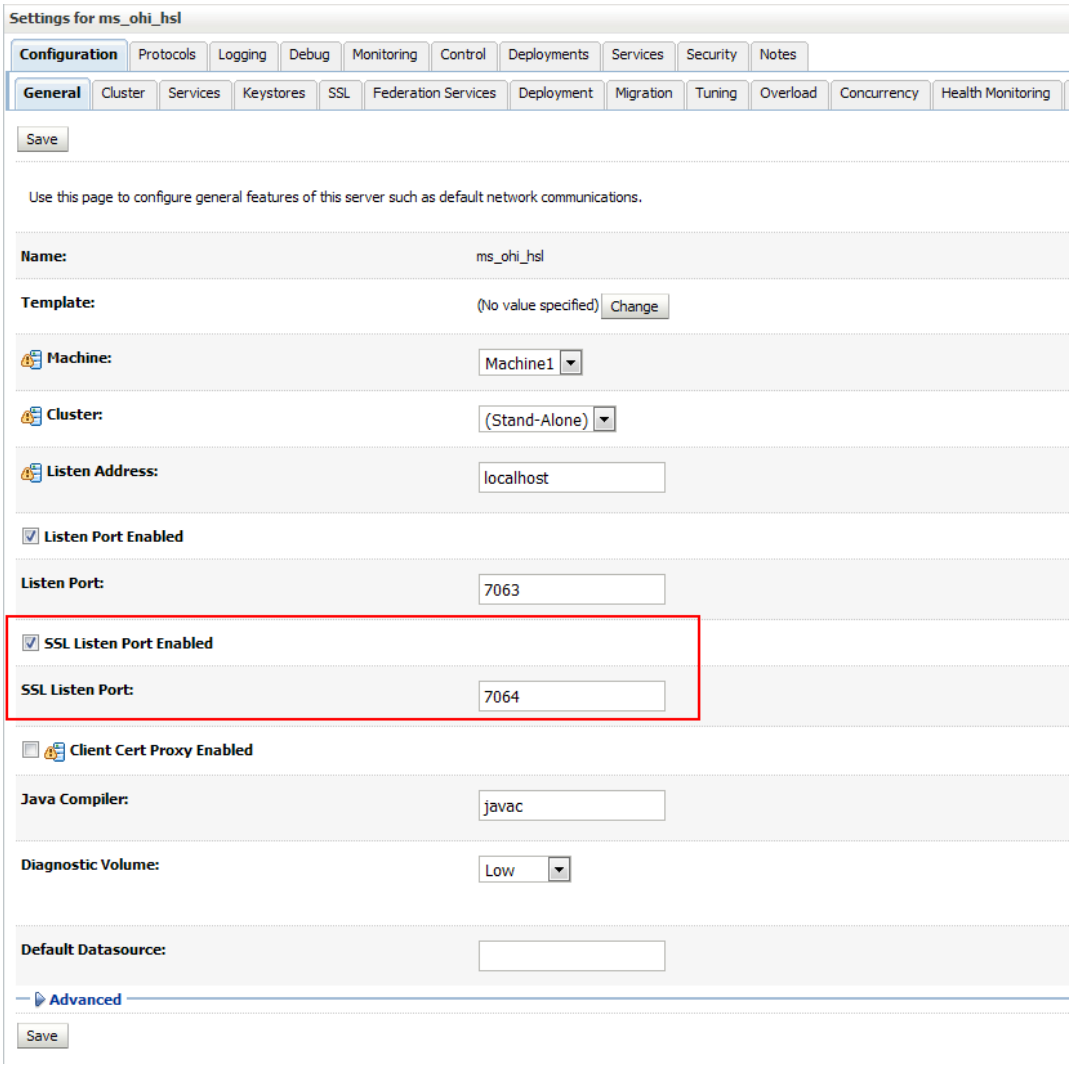


Note that if you do not want to authenticate an HSL application using the predefined weblogic user 'restuser', you can choose to deploy using 'Custom Roles and Policies'. See 'Additional Security Aspects' below.

3.5.3 Enable SSL

The HSL services are preconfigured to use a default policy which uses SSL. Therefore you need to enable SSL for every Managed Server to which you deploy the HSL application.

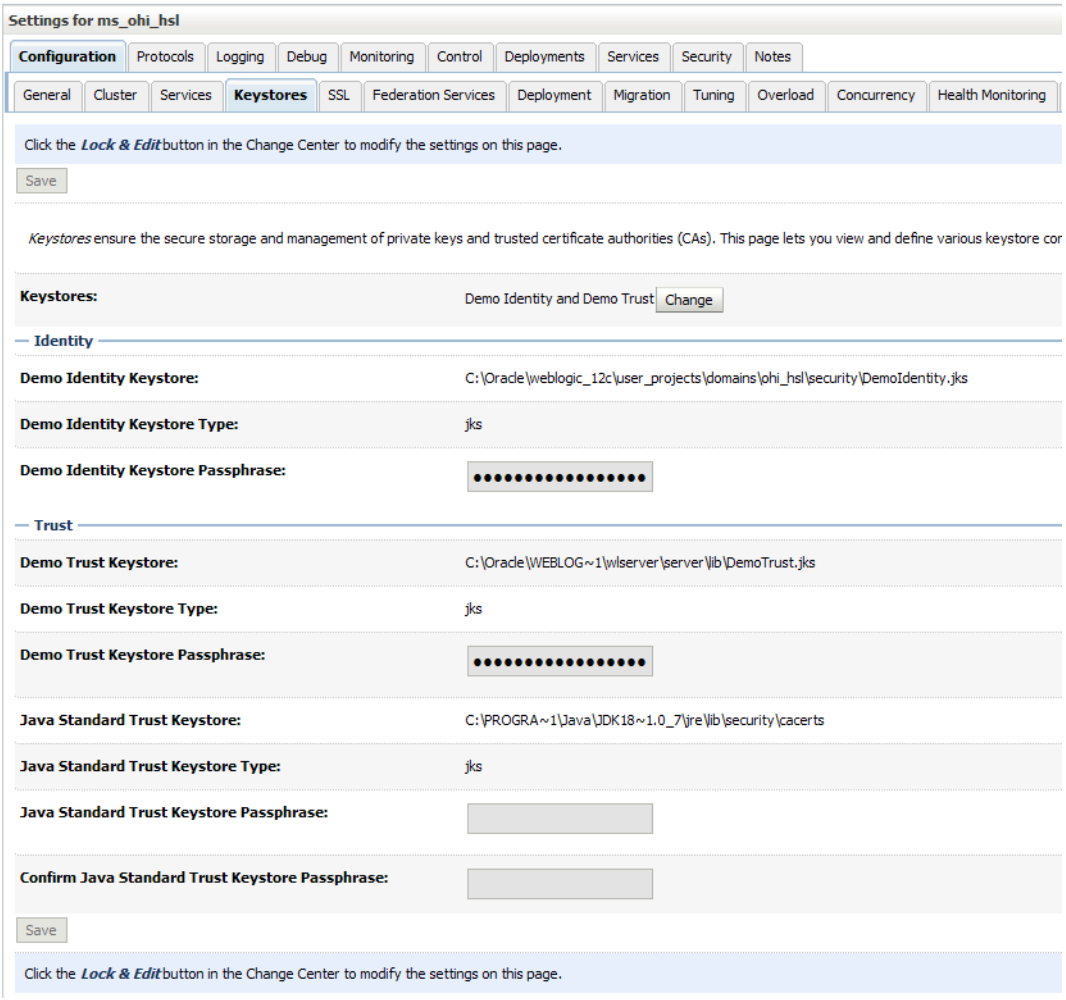
Go to the Managed Server configuration and enable SSL in the 'Configuration > General' tab:



In previous versions op WebLogic, you nneded to configure Java Secure Socket Extension (JSSE) if you JDK version was below JDK 1.8.0_162. Because the miminum JDK version for WebLogic 12.2.1.4.0 is JDK 1.8.0_221, no action is needed any more.

3.5.4 Setting up a key store

For testing purposes you may want to use the built-in keystore as shown below in the 'Configuration > Keystores' tab for the Managed Server:



Note that in a production environment it is not safe to use the demo keystore.

For more information about configuring keystores please read the WebLogic documentation. As a starter you can use this address: Configuring Keystores

It contains references to pages which describe in more detail how to obtain private keys, digital certificates, etc.

You should take action and not rely on the demo keystore!

3.5.5 Configure Managed Server logging level

The standard logging level for a Managed Server regarding security issues is intentionally non-informative to discourage fraudulent users.

A typical security-related error message looks like:

Got 'Unknown exception, internal system processing error.'

If you are trying to setup the HSL application to work with SSL and basic authentication in a non-production environment you can configure verbose logging with the following Server Start argument for the Managed Server:

```
-Dweblogic.wsee.security.debug=true
```

Create a new file $DOMAIN_HOME/bin/setUserOverrides.sh and add the following text:

```
#!/bin/bash
echo Adding Settings from UserOverrides.sh

# global settings (for all servers)
# this will decrease start up times
JAVA_OPTIONS="-Djava.security.egd="file:/dev/./urandom" ${JAVA_OPTIONS}"
export JAVA_OPTIONS
CONFIG_JVM_ARGS="-Djava.security.egd=file:/dev/./urandom ${CONFIG_JVM_ARGS}"
export CONFIG_JVM_ARGS

# specify additional java command line options for the Admin Server
#if [ "${SERVER_NAME}" = "${AS_NAME}" ]
#then
#
#fi
#export JAVA_OPTIONS

# specify additional java command line options for specific servers
if [ "${SERVER_NAME}" = "ms_ohi_hsl" ]
then
  # add settings for HSL
  # Custom Setting for ms ohi hsl to set debug level for SSL
  JAVA_OPTIONS="-Dweblogic.wsee.security.debug="true" ${JAVA_OPTIONS}"
fi
export JAVA_OPTIONS
```

Replace the server name ms_ohi_hsl with your server name.

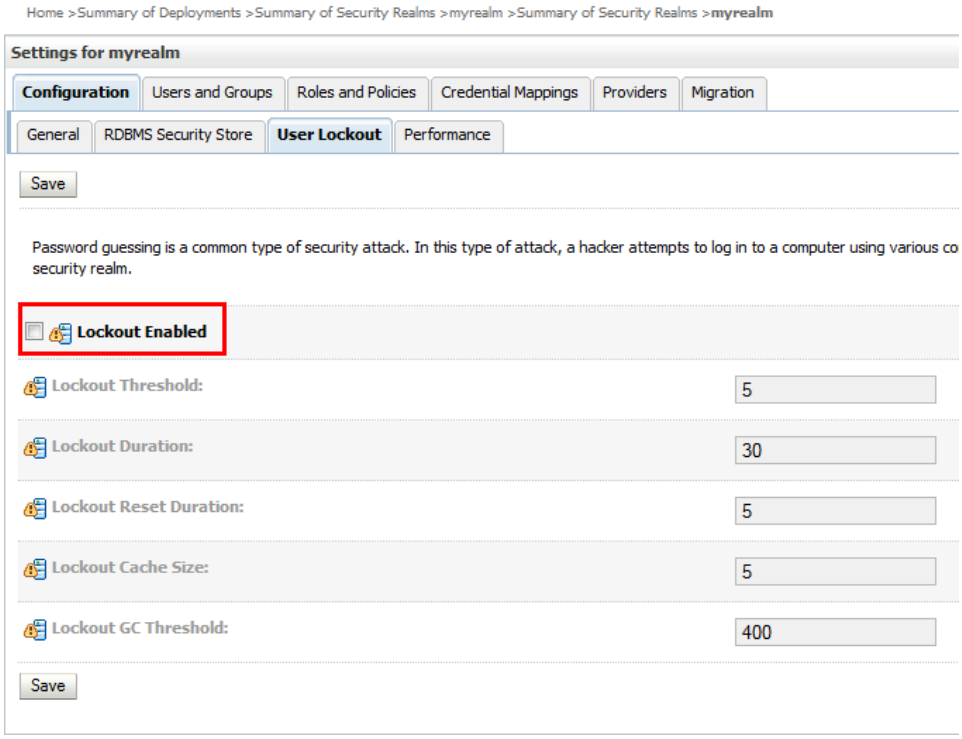
When startup times of your service calls are important and the security of the connection is less important you may consider to specify an alternative for retrieving cryptographically strong random numbers (included above):

```
JAVA_OPTIONS="-Djava.security.egd="file:/dev/./urandom" ${JAVA_OPTIONS}"
```

Restart the Managed Server to get the new verbose messages later on.

3.5.6 Set user lockout

While setting up HSL services for testing you may want to disable user lockout. In a production environment you should enable user lockout to discourage fraudulent use. Navigate to the Security Realm and use the 'Configuration > User Lockout' tab.



3.6 (Re)deployment of the HSL Application

The HSL web applications are packaged in a single archive named 'HSLBOWS.ear'. This EAR file must be deployed to WLS.

The EAR file containing the HSL applications resides in the $OZG_BASE/java directory on the application server containing the OHI Back Office software release.

You can copy this to another location if required.

Ensure that the .ear file is located on the WLS Admin Server host (this is the server running the WLS Administration Console).

Note that you cannot use an older EAR file with a newer OHI Back Office release and vice versa.

The following scenarios are discussed:

- Deploy to a single Managed Server
- Deploy to multiple Managed Servers
- Deploy to a cluster
- Deploy for DTAP (development, test, acceptance, production)

Until OHI Back Office release 10.20.6.0.0 all HSL applications were deployed separately. Before deploying 'HSLBOWS.ear' ensure that none of the HSL_<app>.war files are deployed. If HSL services do exist they need to be deleted in the WLS console. Note that HSL_AUN, HSL_AUZ and HSL_JUP are not part of the 'HSLBOWS.ear' but are deployed via the 'OHIJET.ear'.

3.6.1 Deploy to a single Managed Server

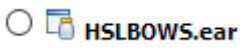
..3.6.1.1 Deploy EAR file

In the Domain Structure pane, select the Deployments branch. This will show the applications that have already been deployed

If you want to shorten this list, use 'Customize this table' to exclude the libraries.

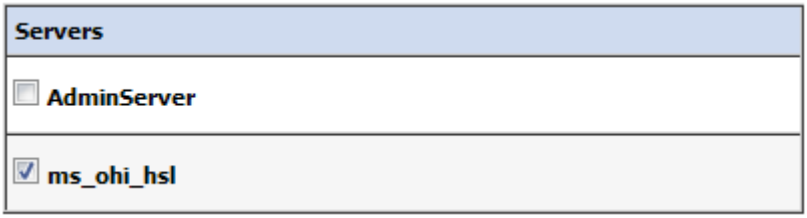
Select 'Lock & Edit' to enter editing mode, this will enable the 'Install' button which you need to use next.

In the new window, locate the .ear file on the WLS server, select one and press 'Next':

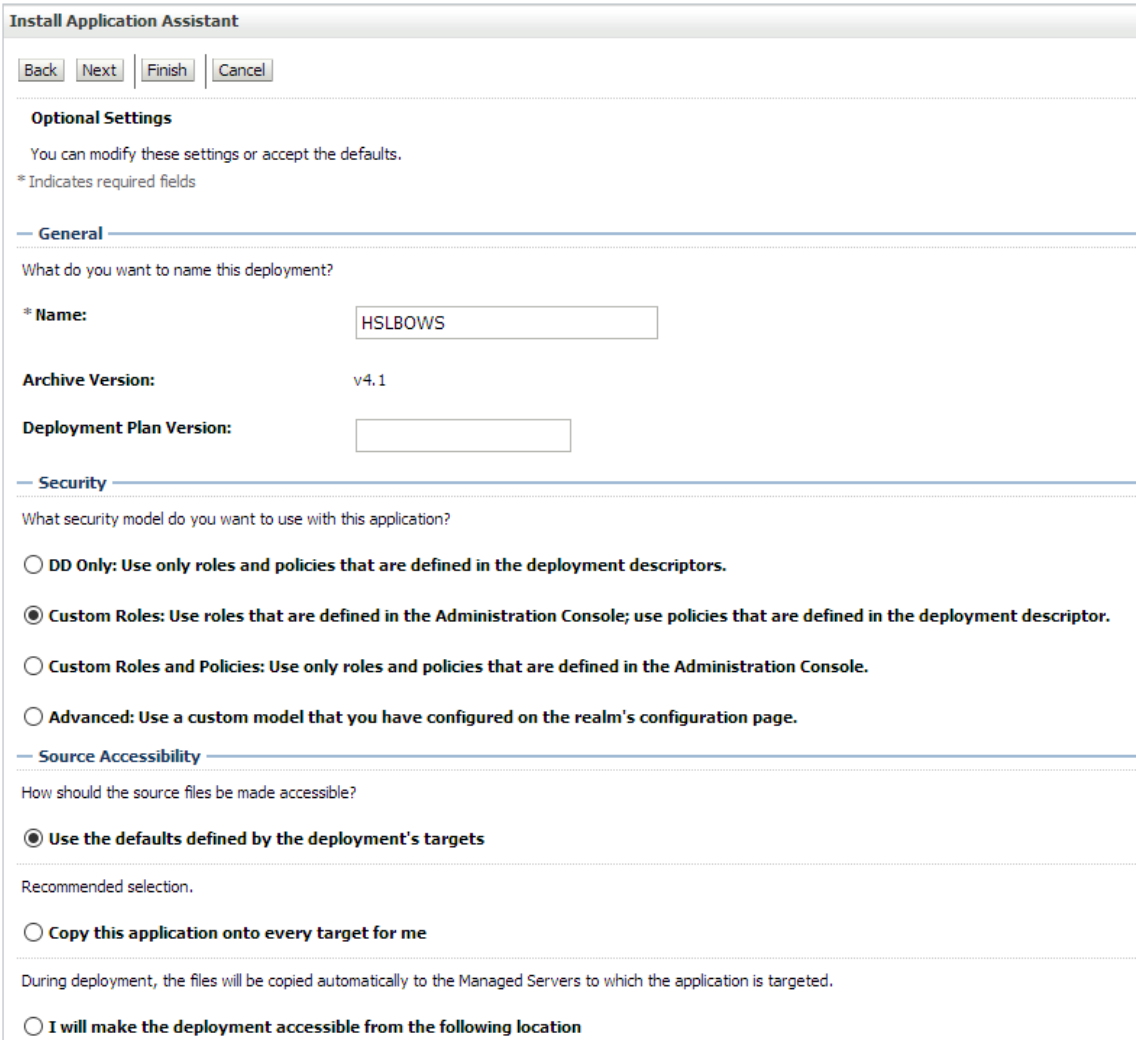


Note: See Appendix E to decide if you need to install HSL_AUN, HSL_AUZ and HSL_JUP too.

Select 'Install this deployment as an application', press 'Next' and select the target(s) for deployment. In the example below only Managed Server ms_ohi_hsl is chosen.



Press 'Next' and decide about a deployment name and security model. At this moment the version of the .war file is also shown (can contain up to 4 digits like any application source).



Select 'Custom Roles' if you want to use the default policy and create your own roles to restrict access to the web services. Select 'Custom Roles and Policies' if you want to overrule the default of each web service.

Regarding source accessibility, select 'Copy this application….' if you want to remove the EAR file from its current location.

Finish the configuration.

Beware that – in Production mode - you need to Activate your changes in order to enable the web services. At that moment the deployment will show status 'Prepared'.

By selecting the deployment in the Control tab and pressing Start → Servicing all requests the State will change to 'Active' (assuming your Managed Server is in 'Running' state, the hsl.properties file has been specified and can be found).

..3.6.1.2 *Specifiy configuration file*

Before using the web services, implement the following actions as described below. These actions have to be executed only once. There is no need to repeat them when you update a deployment or delete and install it again.

Add a Server Start argument by adding a line to the file $DOMAIN_HOME/bin/setUserOverrides.sh you created earlier. Add the line to the part for the HSL server, as indicated below:

```
# specify additional java command line options for specific servers
if [ "${SERVER_NAME}" = "ms_ohi_hsl" ]
then
  # add settings for HSL
  # Custom Setting for ms_ohi_hsl to set debug level for SSL
  JAVA_OPTIONS="-Dweblogic.wsee.security.debug="true" ${JAVA_OPTIONS}"
  # Set location for HSL properties file
  JAVA_OPTIONS="-Dhsl.properties="/u01/app/oracle/product/OHI/vohi/hsl.properties" ${JAVA_OPTIONS}"
fi
export JAVA_OPTIONS
```

- Make sure to keep the parts with ${JAVA_OPTIONS} on the same line

This example uses a properties file with the name hsl.properties which is located in the $OZG_BASE folder of your OHI Back Office application server environment, but you can specify any name and location.

The contents of this file are discussed in a Chapter 5 "*Configuration files for HSL services*").

When completed, (re)start the Managed Server. This can be done from the WebLogic Admin console, or from the command line with the following commands;

```
cd $DOMAIN_HOME/bin
./startManagedWebLogic.sh ms_ohi_hsl http://localhost:7061
```

The example above contains the Managed Server's name as first parameter and the listen address of the Admin Server of the domain as second parameter

Check in the <ManagedServer>.out file in the logs folder of your Managed Server whether the command line contains the arguments as specified above.

If the file specified by hsl.properties cannot be read , messages as below will show up:

```
ERROR: logfile could not be set because of: null
```

3.6.2 Deploy to multiple Managed Servers

You may deploy the application to more than one target.

Example: if you choose to target the application to Managed Servers MS1 and MS2, the application will be available on separate end points. The URLs of these end points will only differ in port number.

If you choose this scenario, be aware that each Managed Server should have different Server Start argument values (hsl.properties).

3.6.3 Deploy to cluster

You may deploy the application on all the Managed Servers of a cluster. This may be needed for better scalability. Be aware to use some form of load balancing to allow the use of a single end point.

The best way to implement this type of deployment depends on your specific situation.

If you are planning a load balanced environment with multiple Managed Servers in a cluster it is vital that the configuration of every Managed Server is aligned with the others.

3.6.4 Deploy for multiple environments (DTAP)

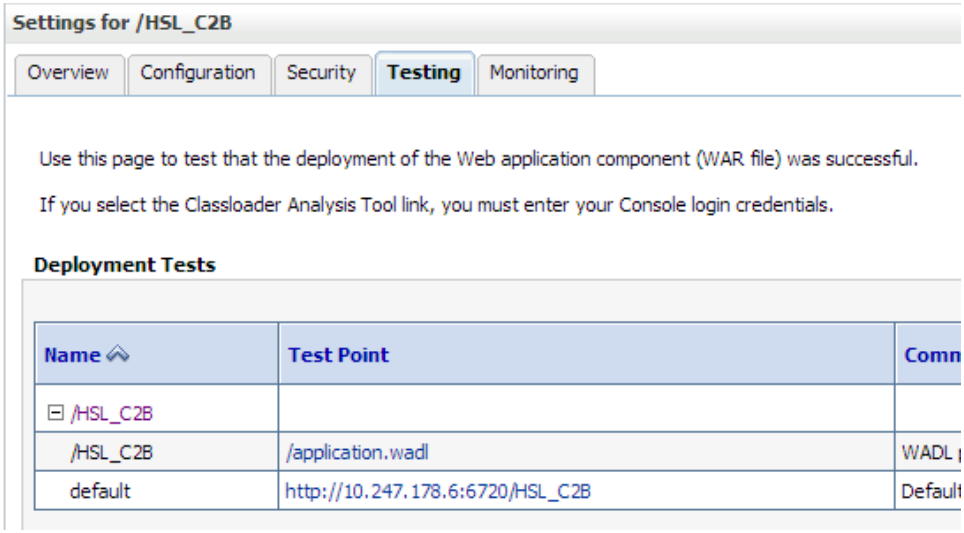
If you use several OHI-related environments to support the various DTAP (Develop-Test-Accept-Production) stages you may want to have different versions of the HSL application running at the same time.

To implement this you need to:

- Create at least one Managed Server for each of the DTAP stages.
- Create a data source for each OHI Back Office database and deploy that data source only to the corresponding Managed Server(s).
- Create an hsl.properties file for each Managed Server.
- Configure each Managed Server to start up with the appropriate hsl.properties.
- Deploy the appropriate version of the HSL application to its corresponding Managed Server(s) and give it a unique deployment name to identify its deployment.

3.6.5 Validate deployment

Be aware that the URLs displayed in the Admin Console cannot be used to test or validate the deployment.



Also note that, even with the correct URLs, you cannot use a browser to test, because the request needs to send a Request Header "Accept:application/json".

You may get no response, or a reply like this:

```
<exceptionResponse>
<internalStatus>Not Acceptable</internalStatus>
<message>Wrong value for Accept</message>
</exceptionResponse>
```

Instead, use curl, as described in chapter 4 "*Deployment validation*" or SoapUI, as described in Appendix C "*Testing with SoapUI*".

3.7 Additional Security Aspects

Since HSL services provide an additional channel to access OHI Back Office data, you must prevent unauthorized use of the HSL applications.

Please consult the 'Oracle Health Insurance Security Aspects' guide for more information about OHI Back Office security aspects.

In order to prevent the exposure of sensitive data or unauthorized changes to the OHI Back Office data, access of the HSL applications should be limited to trusted systems and interfaces. Otherwise people in your organization might be tempted to try to misuse the functionality provided by the HSL services.

By default, HSL service operations require basic authentication as a minimal policy to reduce the risk of unauthorized access and network sniffing. Basic authentication requires HTTPS communication and providing a Weblogic username/password with each call.

The preconfigured deployment descriptor in the HSL applications requires authentication by a Weblogic user 'restuser'. See the instructions for creating this user in 'Security Configuration' above.

If you deploy using 'Custom Roles and Policies' you may:

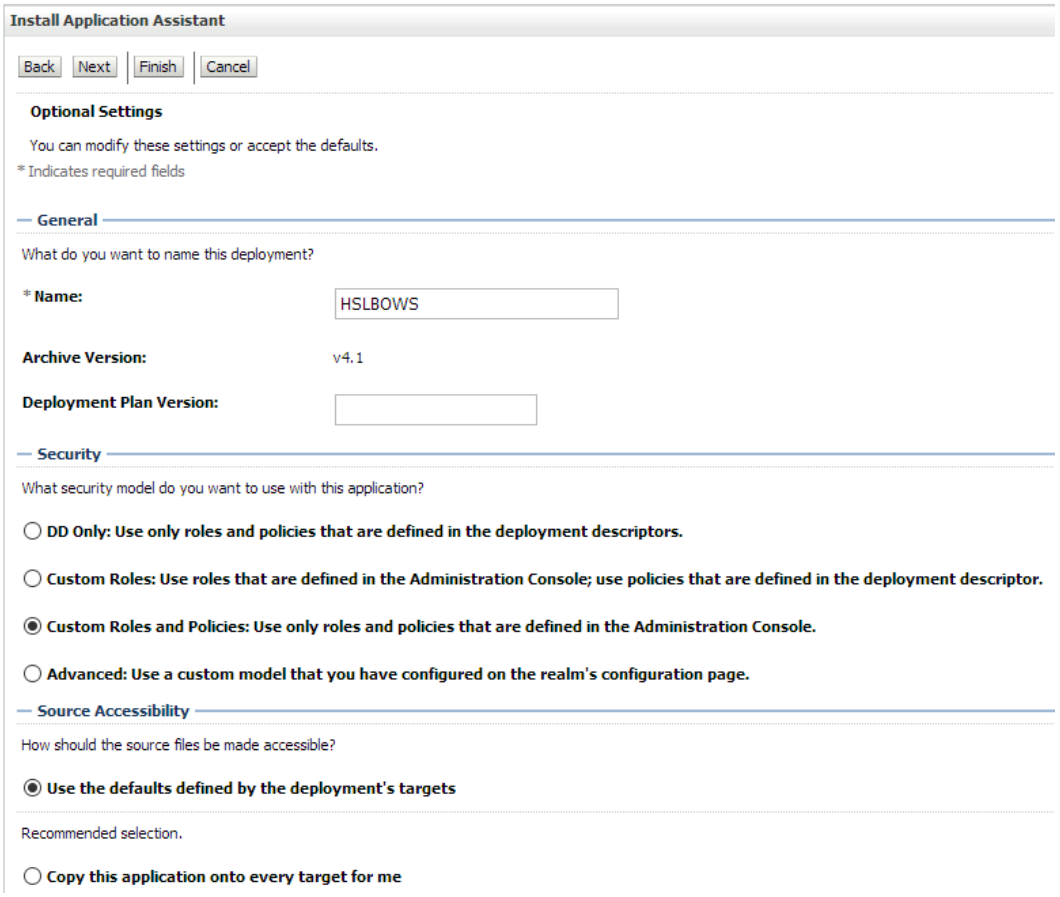
- Deploy the HSL applications for use with any weblogic user
- Use a custom security policy for a deployed web application

Note that all 'admin' requests require Basic Authentication and SSL (/swagger, /swagger.json, /swagger.yaml, /dbinfo and /templates)

Finally, note that it is your responsibility as an administrator to secure the HSL services within your organization.

3.7.1 Deploying HSL Application for use with any weblogic user

If you want to deploy a HSL application for use with another weblogic user than the default 'restuser', you should deploy with the security model 'Custom Roles and Policies':



You can now use any weblogic user to access the HSL application.

3.7.2 Using a custom security policy for a deployed application

The Weblogic console allows the administrator to specify a custom security policy for HSL applications deployed using 'Custom Roles and Policies'.
For example, a custom security policy can be used:

- to limit access to a specified list of named Weblogic users; or
- to limit access to a group of Weblogic users; or
- to limit access to Weblogic users with a specific role; or
- to limit access to a specific web module; or
- to limit access to a specific web service operation inside a module (see Appendix I for an example)
- or a combination of the above.

3.7.3 Cross Site scripting protection

As a measure to prevent against potential cross site scripting attacks you can limit the callers of the REST services by specifying a set of trusted origins. For this the property 'allowedorigins' can specify one or more originating (trusted) server URL's (property files are discussed later on in this document).

An example:

```
hsl.allowedorigins=https://server.domain.com:7430, https://localhost:8000
```

4 Deployment validation

Especially when deploying a HSL application for the first time, it makes sense to validate that the HSL application is in working order.

Before you begin, check in the WLS Admin Console that the deployment status of the HSL application is active.

The following validation tests can be performed by the administrator:

- Template Listing
- getDatabaseInfo operation
- Get Online Swagger definition

Apart from the template listing, each of the validations requires a JDBC connection between the HSL application and the OHI BO database, so you are not only testing the deployment itself but also the integration between the HSL application and the OHI database.

If a validation test fails see the paragraph '*Troubleshooting*' below to find and resolve the problem.

The validation tests described below assume that you test with 'curl'.

4.1 Testing with Curl

An operation of the HSL application can be invoked with many HTTP client tools. One of these tools is curl, which is present on any Linux/Unix server.
Assuming that you have terminal access to the Linux server running WLS, curl is a good tool to run the deployment validation tests.

Use '`curl --version`' to check the curl version. Ensure that you are running curl 7.35.0 or higher as that supports the required SSL implementation.

A typical invocation of a HSL operation using curl would look like this

```
curl -D - -X <verb> -k -H Accept:application/json --user <user>
<url>
```

Explanation of the used options and placeholders:

- `-D -`
  Dump response headers to stdout
- `-X <verb>`
  add HTTP verb (GET/PUT/POST/PATCH/DELETE)
- `-k`
  Allow curl to run HTTP requests without checking SSL certificates.
- `-H Accept:application/json`
  Add request header to require a response in application/json format.
  This is required for every HSL operation.
- `--user <user>`
  The username of the WLS user used for Basic Authentication.

The <user> must refer to an existing WLS user.
Note that curl will prompt for a password if it is not given at the command line.

- <url>
  The path to the HSL operation.

The url format is https://server:port/application/path where

server	**This must be one of the managed servers listed in WLS console as an active target for the HSL application.**
port	The SSL port of the managed server running the HSL application. Every HSL operation requires SSL and Basic Authentication.
application	This is the name of the HSL application as listed on the WLS deployment page. For example, HSL_POL, HSL_REL, HSL_CLA or HSL_C2B.
path	The path to this operation. Each operation is uniquely identified by a <path> + <verb> combination. Path examples: '/dbinfo' or 'templates' or 'api/swagger.json'

In the following example, a template listing is requested from the HSL_POL application on the local WLS host running a managed server at SSL port 7094:

```
curl -D - -k --user restuser -H Accept:application/json
https://localhost:7094/HSL_POL/templates
```

4.2 Template Listing

This operation lists the templates which were used to generate the Java code for the HSL application. The listing itself is irrelevant, but since the operation does not require a JDBC connection with the OHI BO database, it is the simplest of all deployment tests. If it fails the remaining deployment tests will also fail.

The template listing is invoked through

```
https://server:port/application/templates
```

In the following example we retrieve the template listing through curl for the HSL_POL application. It is assumed that we run curl locally on the WLS server and that the managed server running the HSL application can be accessed through SSL port 7094.

```
curl -k -H Accept:application/json --user restuser
https://localhost:7094/HSL_POL/templates
```

The output is a JSON object listing template files and template versions used to generate the Java code for the HSL application.

4.3 getDatabaseInfo

This operation provides information about the database connection between the HSL application and the OHI BO database.
If you are familiar with OHI BO's SVL services, note that the getDatabaseInfo

operation is comparable with the 'isAlive' operation implemented in every SVL service.

This operation requires a working database connection and invokes the PL/SQL implementation package specific to the HSL application.

The getDatabaseInfo operation is invoked through

```
https://server:port/application/dbinfo
```

In the following example the getDatabaseInfo of the HSL_POL application running on SSL port 7094 on our local WLS host is invoked through curl.

```
curl -k -H Accept:application/json --user restuser
https://localhost:7094/HSL_POL/dbinfo
```

The output is a JSON object with information about the database connection and the PL/SQL package implementing the HSL application in the OHI BO database:

```
{
   "basePath": "https://localhost:7094/HSL_POL/pol",
   "database": "BDDEV1722",
   "instance": "CDB02",
   ”jndiName": "HSL_BDDEV1722",
   "plsqlPackage": "hsl_pol_sp_pck $Revision: 4.21 $",
   "user": "HSL_USER",
   "userContext": "MANAGER"
}
```

4.4 verifyInterfaceVersion

This operation checks whether the deployed version at the application server and the corresponding objects as installed in the database do match with each other.

The verifyInterfaceVersion operation is invoked through

```
https://server:port/application/api/verifyInterfaceVersion
```

In the example below the operation is called for application HSL_CLA running on SSL port 7410 on the local WLS host:

```
curl -k -H Accept:application/json --user restuser
https://localhost:7410/HSL_CLA/api/verifyInterfaceVersion
```

The output is a JSON object like:

```
{
  "items":
    [{"name":"plsqlSwaggerRevision","value":"4.47.1.1"}
    ,{"name":"warSwaggerRevision","value":"4.47.1.1"}
    ,{"name":"match","value":"true"}
    ]
}
```

It states whether both versions do match (true) or not (false). They should match and if not you should find out what went wrong, is the problem at the database side of the application server deployment side.

4.5 Get Online Swagger definition

Each HSL application has an operation to generate a Swagger definition which documents the operations and the objects of the HSL service.
This documentation is not only useful to client application developers, but can also be used as the basis for code generation.

The Swagger 2.0 standard is supported by many leading software vendors including Oracle. It is documented on http://swagger.io.

The Swagger definition can be retrieved as follows:

- `https:/`**`server:port/application`**`/api/swagger.json`
  Returns the Swagger definition in JSON format

- `https:/`**`server:port/application`**`/api/swagger`
  Returns the Swagger definition in JSON format

- `https:/`**`server:port/application`**`/api/swagger.yaml`
  Returns the Swagger definition in YAML format

In the following example we retrieve the online Swagger definition of the POL service running on localhost at SSL port 7094:

```
curl -k -H Accept:application/json --user restuser
https://localhost:7094/HSL_POL/api/swagger.json
```

The output is a JSON object containing the Swagger definition of the deployed HSL application.

For retrieving the YAML format beware that you specify x-yaml in the -H argument:

```
curl -k -H Accept:application/x-yaml --user restuser
https://localhost:7094/HSL_POL/api/swagger.yaml
```

4.5.1 Saving the Swagger definition to a file

By redirecting the output of the curl command to a file you can use the contents for other purposes like viewing the Swagger definition in an editor.

In the example below we save the online Swagger definition of the POL service running on localhost at SSL port 7094 to a file called 'saved_swagger.json':

```
curl -k --user restuser -H Accept:application/json
https://localhost:7094/HSL_POL/api/swagger.json >
saved_swagger.json
```

4.5.2 Viewing the Swagger definition

The online Swagger editor (http://editor.swagger.io) provides a user friendly overview of the Swagger definition.

In the following example we use curl to retrieve the Swagger definition of the HSL_POL service and save it to a file. Having opened the saved Swagger definition we then copy its contents into the online Swagger Editor.

Assuming that the HSL_POL application is running on localhost at port 7094 the following command can be used to save the Swagger definition to 'saved_swagger.json':

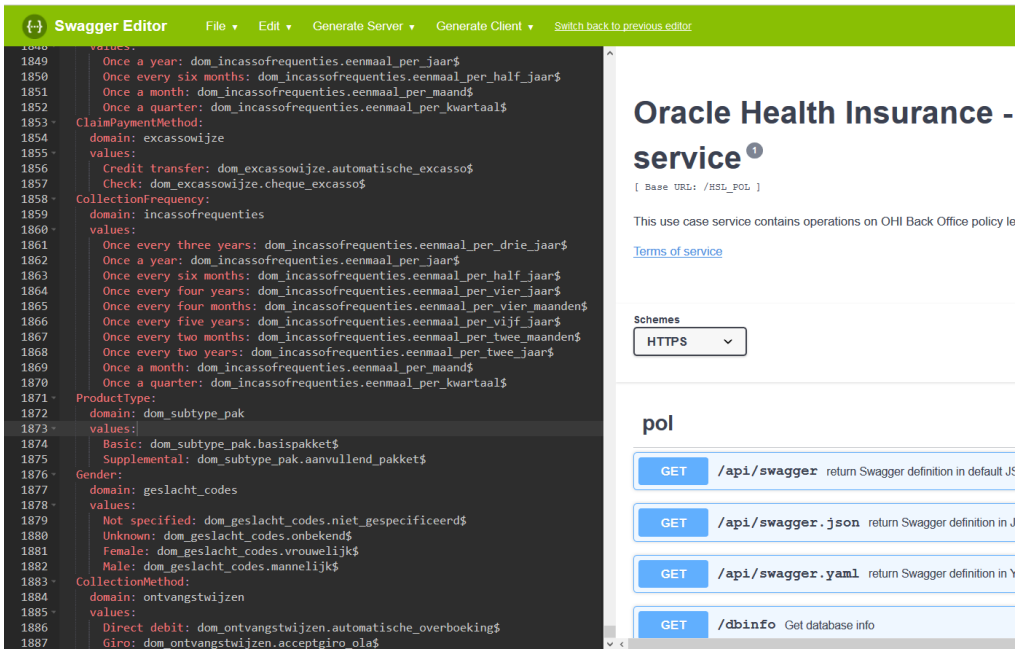
```
curl -k --user restuser -H Accept:application/json
https://localhost:7094/HSL_POL/api/swagger.json >
saved_swagger.json
```

Open 'saved_swagger.json' in a text editor (or browser) and copy the contents of the entire file.

Open the online Swagger editor by browsing http://editor.swagger.io

Use 'File > Clear Editor' to clear the contents of the online editor (if any).
Right-click and paste the contents of the saved Swagger definition into the editor.

The screen will look like this:



You can now navigate through the paths, operations and type definitions of the HSL service. More information about Swagger can be found on http://swagger.io

4.6 Troubleshooting

If the deployment validation fails, first check that the following items have been configured correctly:

- **hsl.properties configuration file**
  This sets the data source for your HSL application.

- **hsl.properties Server Start argument**
  This parameter tells the HSL application where to find the hsl.properties file. If the hsl.properties parameter refers to a non-existing file, the HSL application cannot be started by WLS and its state will be 'Failed'.

- **Data source configured in hsl.properties configuration file**
  This data source is used to create the JDBC connection between the HSL application and the OHI BO database

- **HSL database account**
  This account in the OHI BO database has access to the PL/SQL components used by the HSL application.

Troubleshooting tips:

- Edit the hsl.properties file and set 'developermode=true' for your HSL application.
- Restart the managed server for your application.
  Error messages will now be included in the output (normally they are suppressed from the output).
- Reproduce the request with curl. Be sure to use the 'dumpheader' option (-D) to dump the response headers.

The table below may help you to pinpoint the problem:

HTTP	Message	Problem	Action
	WLS Console: java.lang.RuntimeException: Property file could not be loaded.	The configuration file could not be loaded when starting the HSL application.	Restart application after ensuring that the file referred to by the 'hsl.properties' Server Start argument exists and is readable.
500	Unable to resolve 'xyz'	The jndiname property for this application does not refer to a valid data source.	Examine hsl.properties and ensure that the application's jndiname points to a valid datasource.
500	ORA-06550:line 1..	The required HSL objects cannot be accessed by the database user related to the data source.	Verify that the data source points to a HSL database account. Verify that the HSL database account has access to the HSL objects (see 'Creating a HSL database account' above).
401	Missing Authentication Scheme	No WLS user credentials supplied for this request	Add WLS user credentials to the request. The preconfigured WLS user is 'restuser'.
401	Unauthorized	Wrong WLS user credentials. Wrong username and/or password.	Add correct WLS user credentials to the request.
422	Functional Error	A syntactically correct request could not be completed due to a functional error or business rule violation.	Adapt the request or the database situation so the cause of the failure is prevented.
412	Precondition Failed	Another user already updated these data.	Refresh data and try again.

5 Configuration Files for HSL services

In the previous chapter a properties file was referenced in the web service application server deployment description. This chapter provides more information about that file.

5.1 Properties file templates

With the OHI Back Office release installation, a properties file template called hsl.properties.template is distributed to the $OZG_BASE/conf/Back-Office directory. Each OHI Back Office release may overwrite this template with an updated version. The hsl.properties.template can be used as an example to create your own hsl.properties file (for example in $OZG_BASE/conf).

Please note that all values are examples. You should consider if these values are appropriate for your installation and replace them with your own values if needed. Values indicated with <<SOME_NAME>> in the templates are placeholders and must be replaced. This notation is intended to make scripted deployment easier.

Also make sure not to set log level to FINE, FINER or FINEST in production mode, use SEVERE or WARNING instead.

The generic properties are described in more detail in the next section. The HSL_AUN, HSL_AUZ and HSL_JUP specific properties are described in more detail in Appendix E & F. Also consult Appendix E & F to determine if you need to add the properties for the HSL_AUN, HSL_AUZ and HSL_JUP services at all.

5.2 Back Office HSL properties file

The location of the Back Office properties file for the HSL services is specified as a Server Start argument for a Managed Server with:

```
-Dhsl.properties=<filename>
```

This file contains properties to configure the various deployed HSL applications:

- Datasource to connect the HSL application to the OHI database
- Default OHI officer on whose behalf a request is executed
- Logging configuration.
  Note that HSL services use Java Util Logging (JUL). You may find more information about the configuration options of JUL on the internet.

In the subsection below, the properties are described in more detail, where "<app>" is a placeholder for the service name, like "rel" or "pol" (see the properties file template for more examples).

5.2.1 hsl.<app>.jndiname / hsl.jndiname

The JNDI name of the data source to connect this HSL application to the OHI database is configured in `hsl.<app>.jndiname`. If not set, this value defaults to the value of the `hsl.jndiname` property.

There is no default value if `hsl.jndiname` is not set, which will result in an error.

Setting `hsl.<app>.jndiname` allows you to use different datasources for different HSL applications. A different datasource may connect to the same database using a different account, or to a different database altogether.

As an example, you may want to use HSL_PRD for the REL service and HSL_RO ('read only') for the POL service to avoid changes to the policies in the production database.

Note that you must use // for each forward slash in the JNDI name.

Example:

```
hsl.jndiname=jdbc//HSL_BDDEV1622
```

5.2.2 hsl.<app>.authorization / hsl.authorization

By default, HSL service operations require basic authentication as a minimal policy to reduce the risk of unauthorized access and network sniffing. Basic authentication requires HTTPS communication and providing a Weblogic username/password with each call.

To enforce basic authentication, the value of `hsl.<app>.authorization` or the default `hsl.authorization` property, should be set to BASIC.

If OAUTH2 is desired as authentication method, the value should be set to TOKEN. More details about the configuration of OAUTH2 can be found in *Appendix E – Authentication and Authorization*.

5.2.3 hsl.<app>.usercontext / hsl.usercontext

The OHI officer (Dutch: functionaris) on whose behalf a request is executed.

Setting `hsl.<app>.usercontext` allows you to set an OHI officer per HSL application. If not set, this value defaults to the value of the `hsl.usercontext` property.

The user context is inserted in the call context which is included in the call to the PL/SQL implementation procedure. Note that the PL/SQL implementation may set a different OHI officer based on the request data.

This user context determines the user identity that is used for logging changes to the data, and which language is used for messages. The value must be the Oracle username of a registered BackOffice user (in Dutch: "Functionaris").

Notes:

- the usercontext must always be set, even if `hsl.<app>.usercontext.control` is set to TOKEN
- the user context from the hsl.properties file may be overwritten in the SQL at the HSL application level. This should be documented in the functional specification(s) which apply to the given HSL application.
- This value does not have to match the technical account (HSL_USER) used for the DataSource. If you do want to use HSL_USER, make sure you register a BackOffice user with that Oracle username.
- Do not use the value "MANAGER". Records created and updated by HSL functionality should be recognizable as such. Using MANAGER will make it impossible to distinguish those records from records created or updated by batch procedures and conversion scripts.

Example:

```
hsl.usercontext=HSL_FUNC_USER
```

5.2.4 hsl.<app>.usercontext.control / hsl.usercontext.control

The property `hsl.<app>.usercontext.control` controls where the usercontext is taken from. If not set, this value defaults to the value of the `hsl.usercontext.control` property. For the HSL applications using BASIC authentication, the value should be set to 'PROPERTY' (meaning `hsl.<app>.usercontext` is used to set the user context).

If OAUTH2 authentication is used, the value TOKEN can also be used. See Appendix E for a description of this value.

5.2.5 hsl.<app>.developermode / hsl.developermode

For security reasons, a response for a failed request contains minimal information so that potential hackers cannot use this information to misuse the HSL services. For functional errors, being either a BAD REQUEST or an OHI business rule violation, the functional error is returned. For all other errors, the error message will be replaced with 'Non-disclosed'. The original error message is written to the log file.

If `hsl.<app>.developermode` is set to 'true', the response for a failed request contains the original error message for all errors. If not set, this value defaults to the value of the `hsl.developermode` property. If `hsl.developermode` is not set, developermode is disabled.

Note that in production mode it is strongly advised to delete the `hsl.developermode` and `hsl.<app>.developermode` property from the `hsl.properties` file.

See **Doc[2]** ('Error Handling') for the differences in error handling between developer mode and non-developer mode.

5.2.6 hsl.<app>. allowedorigins / hsl.allowedorigins

For security reasons, to provide cross site scripting attack protection, you can limit the callers of the REST services by specifying a set of trusted origins. See also the separate paragraph with 'Additional Security Aspects' earlier in this document describing this property.

If `hsl.<app>.allowedorigings` is not set, this value defaults to the value of the `hsl.allowedorigina` property. If `hsl.allowedorigins` is not set, protection is not enabled.

Note that in production mode it is strongly advised to set explicitly trusted origins through this property.

An example:

```
hsl.allowedorigins=https://server.domain.com:7430, https://localhost:8000
```

5.2.7 hsl.<app>.logdirectory, hsl.<app>.logfilesuffix & hsl.<app>.logfile / hsl.logdirectory, hsl.logfilesuffix & hsl.logfile

For the purpose of less maintenance, the properties `hsl.logdirectory` and `hsl.logfilesuffix` can be used instead of the `hsl.<app>.logfile` property.

With `hsl.logdirectory`, a generic logdirectory can be defined. When there is no fully qualified logfile defined in `hsl.<app>.logfile`, the log file for the HSL application will be written to this location.

With `hsl.logfilesuffix` a generic logfile name suffix can be defined. When no `hsl.<app>.logfile` property is defined, it will default to "hsl.<app>.log", but when the suffix is set the logfile will default to "hsl.<app>.<logfilesuffix>.log".

When `hsl.logdirectory` is set, the `hsl.<app>.logfile` property no longer has to be set. The HSL application will be able to determine the log file name and location from `hsl.logdirectory` (and optionally `hsl.logfilesuffix`). This means that you will not have to add properties for new HSL applications.

These properties can also be set at app level: `hsl.<app>.logdirectory` and `hsl.<app>.logfilesuffix`, but that will not help to reduce the configuration effort.

`hsl.<app>.logfile` can only be set at <app> level.

Default values:

- `hsl.<app>.logdirectory`: no default value
- `hsl.<app>.logfilesuffix`: no default value
- `hsl.<app>.logfile`: "hsl.<app>.log"

Note:

- the directory referenced in `hsl.<app>.logdirectory` / `hsl.logdirectory` must exist and must be writable by the OS user running the WLS application server
- The `hsl.<app>.logfile` property can be a filename only or a fully qualified path including a filename.

Examples:

- `hsl.logdirectory=/u01/log`
  `hsl.logfilesuffix=BOPROD_AS1`
  `hsl.<app>.logfile` is not set
  This wil result in logfiles for all HSL applications as follows:
  `/u01/log/hsl.<app>.BOPROD_AS1.log`

  This is the recommended setup.

- `hsl.logdirectory=/u01/log`
  `hsl.logfilesuffix` is not set
  `hsl.<app>.logfile` is not set

  This wil result in logfiles for all HSL applications as follows:
  `/u01/log/hsl.<app>.log`

  NOTE: do not use this setup for multiple OHI Environments and/or multiple Managed Servers (e.g. in a clustered environment) to log to the same directory. Locking issues will occur. Use different (sub) directories for each OHI Environment and Managed Server, or use the logfilesuffix to create unique file names.

- `hsl.logdirectory` is not set
  `hsl.logfilesuffix` is not set
  `hsl.<app>.logfile=/u01/log/hsl.myownname.log`

  This wil result in logfiles per HSL applications as follows:
  `/u01/log/hsl.myownname.log`

HSL applications that have no setting for `hsl.<app>.logfile` will try to create a logfile in the WLS domain directory (which is likely to cause issues): `/<WLS domain directory>/hsl.<app>.log`

This setup is not recommended.

5.2.8 hsl.<app>.loglevel / hsl.loglevel

The severity level that determines which log messages should be written is controlled by `hsl.<app>.loglevel`. If not set, this value defaults to the value of the `hsl.loglevel` property. If the global property is also not set, the default value is SEVERE.

Logging levels: SEVERE, WARNING, INFO, CONFIG, FINE, FINER or FINEST.

The following logging levels are currently used: SEVERE, FINE, FINER and FINEST. The logging levels FINE, FINER and FINEST should only be used for debugging.

Example:

```
hsl.loglevel=SEVERE
```

☞ WARNING: When setting the loglevel to FINE, FINER or FINEST this may lead to extensive log messages being recorded which can slow down the processing of service requests considerably. Response times measured while using such detailed log levels are clearly affected and should not be considered as representative for regular use.

5.2.9 hsl.<app>.log.limit / hsl.log.limit

The maximum size of the log file (in bytes) is controlled by `hsl.<app>.log.limit`. If not set, this value defaults to the value of the `hsl.log.limit` property. If the global property is also not set, the default value is 1000000 (1Mb).

When the size of the log file reaches this limit, the log is rolled over to the next log file.

Note that a value of 0 means 'unlimited'.

Example:

```
hsl.log.limit=5000000
```

5.2.10 hsl.<app>.log.count / hsl.log.count

The number of log files to use in the log file rotation is controlled by `hsl.<app>.log.count`. If not set, this value defaults to the value of the `hsl.log.count` property. If the global property is also not set, the default value is 1.

A value of 1 means that only 1 log file is created and no log rotation takes place. When the log.limit is reached, the log file is overwritten and its previous contents are lost.

Set the log.count to 2 or higher to avoid overwriting the log file once it is full.

Example

```
hsl.log.count=2
```

5.2.11 hsl.<app>.log.append / hsl.log.append

Configure `hsl.<app>.log.append` if logging can be appended to existing log files. If not set, this value defaults to the value of the `hsl.log.append` property. If the global property is also not set, the default value is true.

If false, a new log file will be created when rotating log files.

Example:

```
hsl.log.append=false
```

5.2.12 Activating changes to hsl.properties

To activate changes to hsl.properties you must restart the managed server.

5.2.13 Troubleshooting hsl.properties

Note the following if you have trouble starting up with a new hsl.properties file:

- an empty value for ANY property will block any HSL application from starting up.
  Example:
  `hsl.jndiname=`
- lines starting with '#' are ignored.
- empty lines are ignored
- do not use whitespace characters in property lines (even at the end). Whitespace characters are tabs and spaces. Inserting whitespace characters may result in a malfunction in the operation of HSL services.

5.2.14 Keeping hsl.properties up to date

When new HSL properties are released through (patch) releases of OHI Back Office, the installation instructions will tell you to change the hsl.properties file if required. Also, an updated properties file template will be released, as described in the previous section 'Properties file templates'.

5.3 Examining the Log File

When encountering long-running HSL operations, examining the log file allows you to break down the roundtrip into different components.

Ensure that the log level for the HSL application is set to FINE.

If the log level is set to FINEST, writing log messages alone may require significant time and may account for much of the time spent in the HSL operation.

If you changed the log level you must restart the Managed Server to activate the new log properties.

Next, look up the long-running operation in the log file. The example shows log messages of a fictitious operation:

```
Mar 08, 2018 5:09:39 PM com.oracle.insurance.ohibo.hpo.HpoService
getDossierRegels
FINE: begin getDossierRegels
Mar 08, 2018 5:09:39 PM com.oracle.insurance.ohibo.hpo.HpoService getLanguage
FINE: getLanguage() returns: nl-NL
Mar 08, 2018 5:09:39 PM com.oracle.insurance.ohibo.hpo.HpoService
getDossierRegels
FINE: expand=all
Mar 08, 2018 5:09:39 PM com.oracle.insurance.ohibo.hpo.HpoService
getDossierRegels
FINE: limit=10000
Mar 08, 2018 5:09:39 PM com.oracle.insurance.ohibo.hpo.HpoService
getDossierRegels
FINE: number=35
Mar 08, 2018 5:09:39 PM com.oracle.insurance.ohibo.hpo.HpoService
getDossierRegels
FINE: offset=0
Mar 08, 2018 5:09:39 PM com.oracle.insurance.ohibo.hpo.CCallContext
toJDBCObject
FINE: enter toJDBCObject
Mar 08, 2018 5:09:40 PM com.oracle.insurance.ohibo.hpo.CCallContext
toJDBCObject
FINE: leave toJDBCObject
Mar 08, 2018 5:09:41 PM com.oracle.insurance.ohibo.hpo.HpoService
getDossierRegels
FINE: Before calling PL/SQL operation
Mar 08, 2018 5:09:59 PM com.oracle.insurance.ohibo.hpo.HpoService
getDossierRegels
FINE: After calling PL/SQL operation
Mar 08, 2018 5:10:00 PM com.oracle.insurance.ohibo.exception.ExceptionUtil
handleReturnContext
FINE: start handleReturnContext
Mar 08, 2018 5:10:00 PM com.oracle.insurance.ohibo.exception.ExceptionUtil
handleReturnContext
FINE: end handleReturnContext
Mar 08, 2018 5:10:00 PM com.oracle.insurance.ohibo.hpo.HpoService
getDossierRegels
FINE: Before mapping SQL object to Java object
Mar 08, 2018 5:10:21 PM com.oracle.insurance.ohibo.hpo.HpoService
getDossierRegels
FINE: After mapping SQL object to Java object
Mar 08, 2018 5:10:21 PM com.oracle.insurance.ohibo.hpo.HpoService
getDossierRegels
FINE: http code=200
Mar 08, 2018 5:10:21 PM com.oracle.insurance.ohibo.hpo.HpoService
getDossierRegels
FINE: Before creating response
Mar 08, 2018 5:10:22 PM com.oracle.insurance.ohibo.hpo.HpoService
getDossierRegels
FINE: After creating response
Mar 08, 2018 5:10:22 PM com.oracle.insurance.ohibo.hpo.HpoService
getDossierRegels
FINE: end getDossierRegels
```

From this fragment we may derive the following information:

- Total roundtrip is about 43s (5:09:39 - 5:10:22)
- PL/SQL execution: 18s (5:09:41 - 5:09:59)
- Mapping SQL object to Java object:<1s
- Creating response with JSON string:<1s

5.3.1 Changing the log format

The default format for logging timestamps is not suitable for sub-second operations. Logging timestamps in milliseconds since 01-01-1970 is needed if you want to analyse sub-second operations.

To override the default format, create a configuration file with the following contents:

```
# override default format for timestamps in milliseconds since 01-01-1970.
java.util.logging.SimpleFormatter.format=%1$tQ %2$s%n%4$s: %5$s%6$s%n
```

You now need to activate this configuration for the managed server to which the HSL application is deployed:

- Start WebLogic Console
- Choose Environments > Servers > *managed_server*
- Add `-Djava.util.logging.config.file=`*your_config_file* to the Server Start parameters. . Add a line to the file $DOMAIN_HOME/bin/setUserOverrides.sh you created earlier. Add the line to the part for the HSL server:

```
JAVA_OPTIONS="-Djava.util.logging.config.file=”your_config_file” {JAVA_OPTIONS}"
```

- Restart the managed server.
- Call the HSL operation and check that the subsequent log messages show log messages in milliseconds since 01-01-1970

The output should now look like:

```
1520867075960 com.oracle.insurance.ohibo.hpo.HpoService getDossierRegels
FINE: begin getDossierRegels
1520867075971 com.oracle.insurance.ohibo.hpo.HpoService getLanguage
FINE: getLanguage() returns: nl-NL
1520867075974 com.oracle.insurance.ohibo.hpo.HpoService getDossierRegels
FINE: expand=all
1520867075974 com.oracle.insurance.ohibo.hpo.HpoService getDossierRegels
FINE: limit=10000
1520867075975 com.oracle.insurance.ohibo.hpo.HpoService getDossierRegels
FINE: number=11
1520867075975 com.oracle.insurance.ohibo.hpo.HpoService getDossierRegels
FINE: offset=0
1520867075976 com.oracle.insurance.ohibo.hpo.CCallContext toJDBCObject
FINE: enter toJDBCObject
1520867075977 com.oracle.insurance.ohibo.hpo.CCallContext toJDBCObject
FINE: leave toJDBCObject
1520867075978 com.oracle.insurance.ohibo.hpo.HpoService getDossierRegels
FINE: Before calling PL/SQL operation
1520867092723 com.oracle.insurance.ohibo.hpo.HpoService getDossierRegels
FINE: After calling PL/SQL operation
1520867092728 com.oracle.insurance.ohibo.exception.ExceptionUtil
handleReturnContext
FINE: start handleReturnContext
1520867092729 com.oracle.insurance.ohibo.exception.ExceptionUtil
handleReturnContext
FINE: end handleReturnContext
1520867092730 com.oracle.insurance.ohibo.hpo.HpoService getDossierRegels
FINE: Before mapping SQL object to Java object
1520867093066 com.oracle.insurance.ohibo.hpo.HpoService getDossierRegels
FINE: After mapping SQL object to Java object
1520867093068 com.oracle.insurance.ohibo.hpo.HpoService getDossierRegels
FINE: http code=200
1520867093072 com.oracle.insurance.ohibo.hpo.HpoService getDossierRegels
FINE: Before creating response
1520867093073 com.oracle.insurance.ohibo.hpo.HpoService getDossierRegels
FINE: After creating response
1520867093074 com.oracle.insurance.ohibo.hpo.HpoService getDossierRegels
FINE: end getDossierRegels
```

This log output of a fictitious operation gives us the following information:

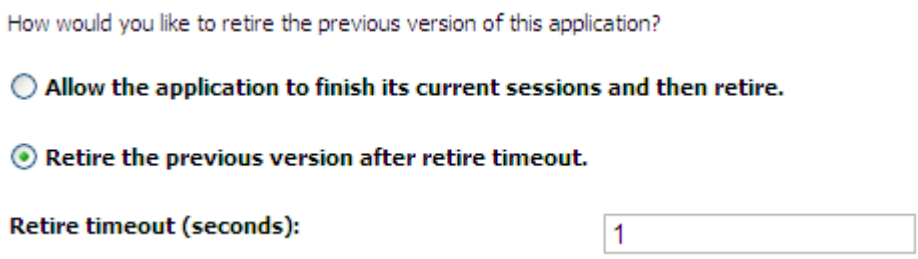
- Total roundtrip is 17114 ms (1520867093074 - 1520867075960)
- PL/SQL execution: 16745 ms (1520867092723 – 1520867075978)
- Mapping SQL object to Java object: 336 ms (1520867093066 - 1520867092730)
- Creating response with JSON string: 1 ms

6 Upgrading HSL services

Future OHI releases may include a new EAR file for HSL services.

To deploy a new version of an existing HSL application, follow the steps below:

- ✓ Check your web service properties file (typically hsl.properties) and implement necessary changes for your release. For information about the contents please see the previous chapter.
- ✓ Logon to the Admin Server console of the domain where the web services are deployed.
- ✓ Navigate to the deployments pane.
- ✓ Choose the 'Lock & Edit' option.
- ✓ If you already have a Retired version of the deployment, mark the check box in front of the retired deployment and delete it.
- ✓ Navigate to the deployment that must be updated and mark the check box in front of it.
- ✓ Press the Update button.
- ✓ Determine whether the same source path still applies (typically a new version is delivered in the $OZG_BASE/java folder of your environment but your organisation may have additional distribution methods implemented). When the correct .ear file is selected press Next.
- ✓ You now have two options for 'retiring' the previous version. Because normally the Back Office application is not available during patching, you can retire the previous version 'immediately', meaning using a timeout of 1 second:



  Press 'Finish' to retire the previous version and continue.

- ✓ Choose 'Activate Changes'.
- ✓ Refresh the screen a few seconds after having activated the changes.
- ✓ Inform the communities which use the web services of the availability and publish the latest URI's to the swagger definitions to them.

It the old deployment cannot be deleted when updating, stop the deployment with the 'Force' option and deploy it again completely (using the 'Install' option for deployments). In some cases (depending on the changes) you may need to repeat the Deployment delete/install when the install results in errors. If the deployment keeps failing, you may have to restart the Managed Server(s) as a last resort.

After this the deployment state of the web services should be Active again (be sure the Managed Server(s) is/are running, otherwise start it/them to get this result).

If not, check whether your OHI database environment and deployed version are correct, meaning that their version levels correspond with each other.

7 Appendix A – Service Information

The following URI provides version information about a running HSL application:

`https://server:port/application/dbinfo`

For example:

https://localhost:7002/HSL_POL/dbinfo

This will return a JSON object like below:

```
{
  "basePath": "https://localhost:7002/HSL_POL/pol",
   "database": "BTSPC19",
   "instance": "CDB19",
   "jndiName": "HSL_BTSPC19",
   "plsqlPackage": "hsl_pol_sp_pck $Revision: 4.39 $",
   "user": "HSL_USER",
   "userContext": "MANAGER"
}
```

Information:

- basePath
  Format: `https://server:port/application/context`
  This is the base URI for all operations in this service.

- database
  The name of the database associated with the current database connection

- instance
  Instance name of the database associated with the current database connection.

- jndiName
  The JNDI name of the database connection (specified in the hsl.properties file)

- plsqlPackage
  The PL/SQL package which implements the operations of the HSL service.
  In this release, the revision number refers to the revision number of the code template used to generate the PL/SQL package. In a future release this will point to the minimum revision number of the compiled PL/SQL package.

- user
  The database account used to log on to the database.

- user context
  The default OHI officer on whose behalf service operations are performed, as specified in the hsl.properties file.

8 Appendix B – Removing a WLS domain

In case you want to restructure your environment or recreate a domain you can remove an existing domain.

In order to do this make sure all servers for the domain are stopped and make sure there is no Node Manager process running which 'guards' this domain.

Next perform the following actions:

- ✓ Completely remove your domain directory including all contents.
- ✓ Remove any reference in start and stop scripts to this domain.
- ✓ Remove, if present, the domain from the <WebLogic home>\oracle_common\common\nodemanager\nodemanager.domains.
- ✓ Remove the domain from the domain-registry.xml file which is located in the Middleware home folder ($MW_HOME).

For more information please use the standard WebLogic documentation.

9 Appendix C – Testing with SoapUI

SoapUI is a tool for testing web services which can be downloaded from http://www.soapui.org.

It is especially useful for functional testing of the HSL application.

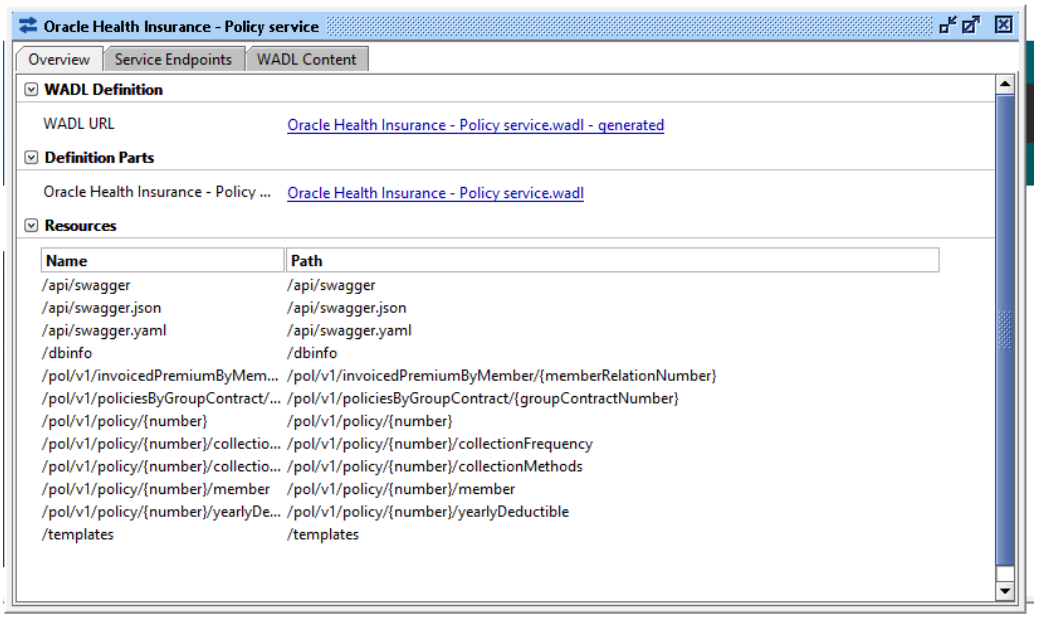
WLS 12c has removed the support for the security protocols SSLv3 and TLS 1.1, because they are now considered insecure.
This means you must access the HSL application with a client that uses TLS 1.2.

Unlike earlier versions, SoapUI 5.3 and 5.4 enable TLS 1.2 by default. The examples below assume SoapUI 5.4 or higher.

9.1 Create REST project and import Swagger definiton

- Follow the instructions in 'Get Online Swagger Definition (curl)' to retrieve the online Swagger definition from the HSL application and save the output to a file (for example 'saved_swagger.json')
- Create a new REST project (empty value for URL)
- Choose 'Project > Import Swagger' and select the saved Swagger definition.

The operations of the HSL application are now discovered:



You may now create requests for the operations provided by this HSL application.

9.2 Create a request

Once you have imported the Swagger definition you may create a request for each of the operations.

In the example below we create a request for the getDatabaseInfo operation:

- Double-click on 'Request 1' of the requested operation (in our case /dbinfo > getDatabaseInfo).

- Set the endpoint for the request to `https://<server>:<port>`
  For example https://127.0.0.1:7094.

- Select 'Headers' and add a HTTP request header with Header value 'Accept' and with Value value 'application/json'.

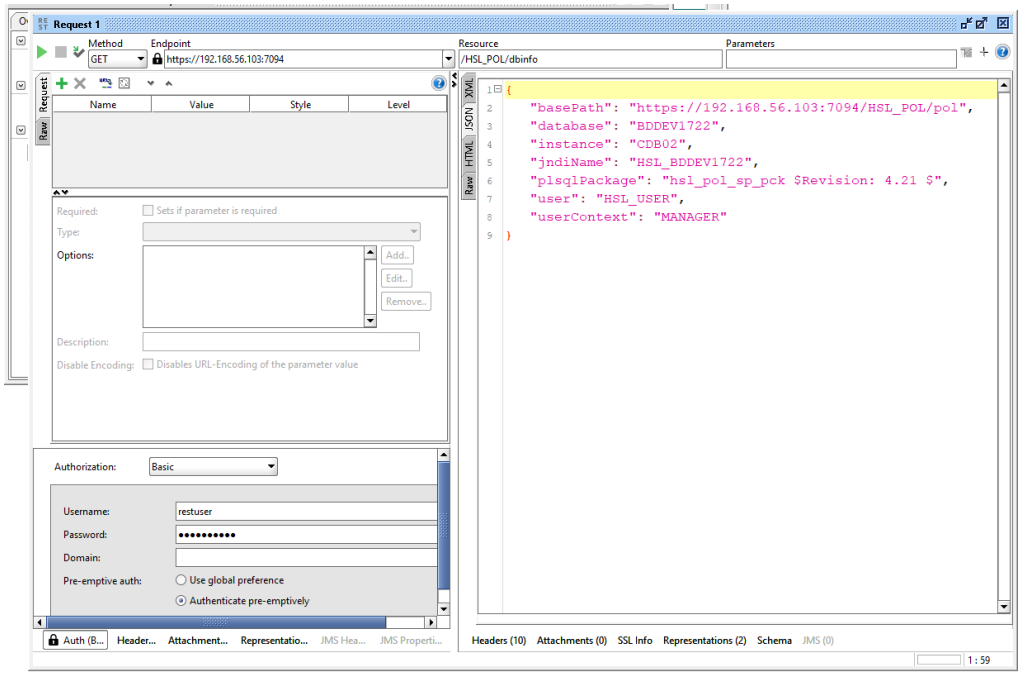
- Add other HTTP request headers as required (not needed for this example)

- Select 'Auth' to add Basic Authentication for the WLS user.
  If you deployed with the 'DD Only' deployment model the WLS user should be 'restuser'.

- Set 'preemptive authentication'.

- Run the request.

The request window should now look like this:



10 Appendix D - Generating a WADL file

A WADL (Web Application Description Language) file may be required by Oracle Service Bus or other middleware to describe your HSL application.

The current HSL applications cannot be used to generate WADL files directly.

However, a WADL file can be easily generated from the online Swagger definition using SoapUI.

This involves the following steps:

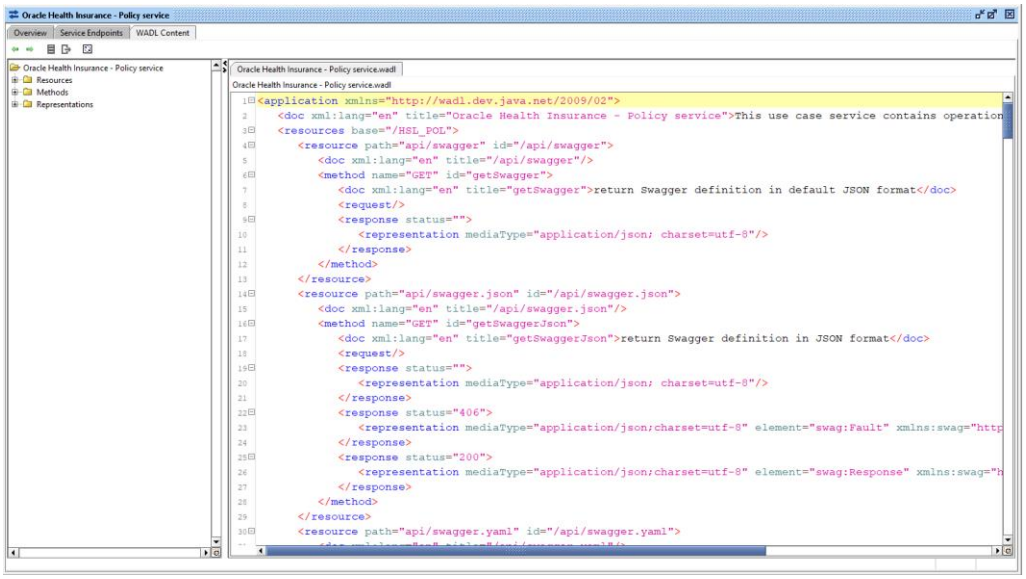
- Create a REST project in SoapUI for your HSL application
- Open the Service Viewer for the REST project
- Export WADL from your REST project

10.1 Create a REST project in SoapUI for your HSL application

Follow the instructions in 'Testing with SoapUI' to set up SoapUI for testing with your HSL application.

10.2 Open the Service Viewer for the REST Project

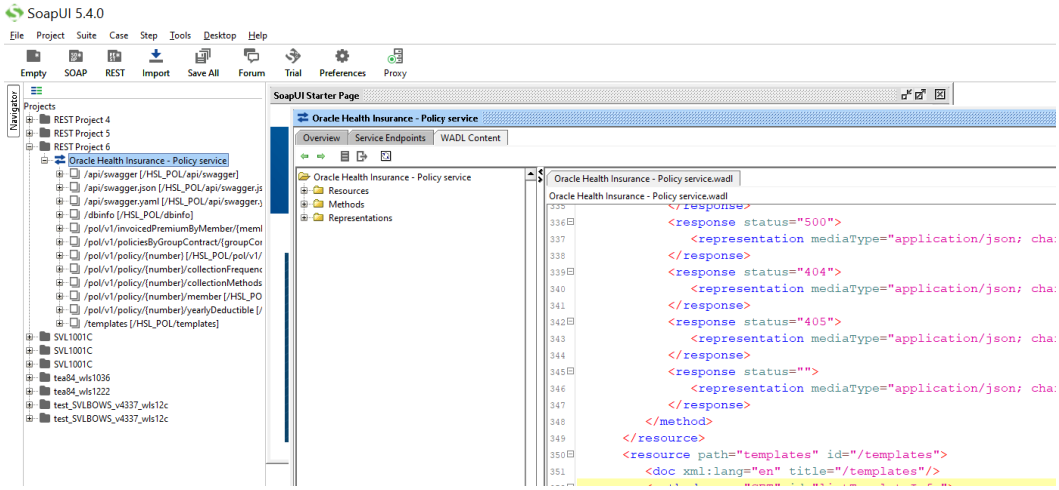
Click on the 'WADL Content' to see the WADL description.
Your screen may look like this:



10.3 Export WADL from your REST project

Save the buffer to a WADL file.

Alternatively you may right-click on the service within the REST project (highlighted in the screen shot below):



And select 'Export WADL' to create the WADL for this application.

11 Appendix E – Authentication and Authorization

In 10.18.1.0.0 the following changes were made to the security of the HSL services (and later on also the PSL services):

- No authentication needed for OPTIONS method
- OAUTH 2.0 token authentication and validation as an alternative to Basic Authentication

11.1 HTTP OPTIONS method

The HTTP OPTIONS method is used by browsers as a pre-flight check to retrieve the allowable methods for a given URL. This check, for which no authentication is needed, is part of the CORS (Cross-Origin Resource Sharing) mechanism.

11.2 OAUTH 2.0 token authentication and validation

If Basic Authentication is used, the 'Authorization' header starts with 'Basic' followed by a base64-encoded username/password combination for a valid Weblogic account. Basic authentication is the default mechanism for HSL services.

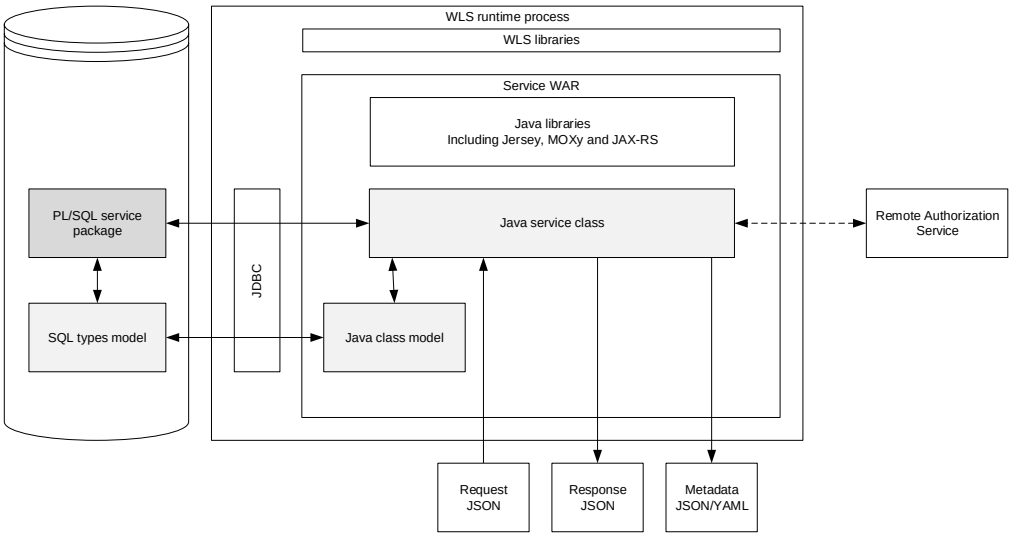
As an alternative for Basic Authentication, support for OAUTH 2.0 has been developed for the HSL and PSL services:

- Authentication based on access-token
- Token validation using an Authorization Service
- Set OHI officer (optional)

For the PSL services OAUTH2 is the default authorization method.

If access-token authorization is used, the 'Authorization' header starts with 'Bearer' followed by an encoded string which must be validated by a remote authorization service.

The token validation using a remote authorization service has been indicated with a dotted line in the application architecture diagram:



11.2.1 WAR File Deployment

In order to use access token validation (OAUTH2), the application must be deployed with 'Custom Roles and Policies'.

11.2.2 Configuration

Authentication and authorization are configured through the hsl.properties and psl.properties file.
To implement the changes in the properties file, the managed server(s) associated with the WAR file must be restarted.

11.2.3 Authorization methods

The `hsl.<app>.authorization` property selects which authorization methods are allowed for a HSL service (and likewise for PSL services in a psl property). This value defaults to the value of `hsl.authorization` (or `psl.authorization`).

If `hsl.authorization` is not set, the default value of 'BASIC' is used to enforce Basic Authentication.

Allowed values:

- BASIC – use 'Basic' HTTP Authorization header with WLS credentials
- TOKEN – use 'Bearer' HTTP Authorization header with JWT token
- Or a combination of these.

11.2.4 Access Token Validation

To support the OAUTH2 access token validation, the HSL or PSL service must be configured to call a remote authorization service, which validates:

- that the caller is a legitimate user of the system
- that the caller is authorized for the requested HSL/PSL service operation (expressed through path+method)

The HSL/PSL service which is called by the client application now dynamically creates the call to the remote authorization service.

The remote authorization should meet the following criteria:

- The authorization service can be invoked using the HTTP protocol.
- POST or GET is used as the default method to invoke remote authorization
- Basic authentication may be used to call the remote authorization service
- JSON is used to format the body parameter

To test this functionality, the HSL_AUN and HSL_AUZ services were developed. See 'Appendix F' for a description.

The following hsl.properties parameters are used to configure the call (for PSL services the same psl properties can be applied):

- `hsl.<app>.tokenvalidation.url` - defaults to the value of `hsl.tokenvalidation.url`

The URL of the authorization service (for testing purposes this can be the URI of the HSL_AUZ service).

- `hsl.<app>.tokenvalidation.method` - defaults to the value of `hsl.tokenvalidation.method`
  Method to access the authorization service operation
  Default value: POST

- `hsl.<app>.tokenvalidation.headerparams` - defaults to the value of `hsl.tokenvalidation.headerparams`
  A template with place holders which is used to add HTTP request headers when calling the remote authorization service. The place holders are described in the next section.

- `hsl.<app>.tokenvalidation.queryparams` - defaults to the value of `hsl.tokenvalidation.queryparameters`
  A template with placeholders which is used to construct a query string. The place holders are described in the next section.

- `hsl.<app>.tokenvalidation.bodyparam` - defaults to the value of `hsl.tokenvalidation.bodyparam`
  A template with placeholders which is used to create a JSON string which is used as a body parameter. The place holders are described in the next section.

- `hsl.<app>.tokenvalidation.authentication` - defaults to the value of `hsl.tokenvalidation.authentication`
  Contents for the Authorization header which is used to authenticate the request to the remote authorization service.
  The value for `hsl.tokenvalidation.authentication` is a base64 encoded username:password string prepended with 'Basic '.
  The username:password combination must refer to a valid Weblogic user which has access to the HSL_AUZ service.
  You may use the linux command base64 to generate the base64 encoded string, e.g. (the -n is needed to return a string without carriage retun character).

  ```
  echo -n wlsuser1:password1 | base64
  ```

Note that the dynamically constructed request may be created from

- `hsl.<app>.tokenvalidation.headerparams`
- `hsl.<app>.tokenvalidation.bodyparm`
- `hsl.<app>.tokenvalidation.queryparams`
- or a combination of these.

11.2.5 Place Holders

The following placeholders are expanded before calling the remote authorization service (in the next paragraphs some example code is shown using these placeholders):

- #method#
  Method (GET,POST,PUT,PATCH,DELETE) of the service operation. The method value is derived from the HTTP request for the service operation.

- #jwt#
  Contents of the access token which must be passed to the remote authorization service.

This is the contents of the 'Authorization' header of the original request after removing the 'Bearer\s' prefix.

- #path#
  Path to the service operation for which the token validation is desired. The path value is derived from the HTTP request for the service operation.

11.2.6 POST Example

```
hsl.tokenvalidation.authentication=Basic cmVzdHVzZXI6b3Blbnpvcmc5OQ==
hsl.tokenvalidation.bodyparam={ "method" : "#method#", "token" : "#jwt#" ,
"resource" : "#path#" }
hsl.tokenvalidation.method=post
hsl.tokenvalidation.url=https://ol6ohi.us.oracle.com:7110/HSL_AUZ/auz/v1/authorizati
on/verify
```

11.2.7 GET example

Fictitious example to configure a verify operation using the GET method:

```
hsl.hba.tokenvalidation.url=https://ol6ohi.ohi.oracle.com:7094/HSL_HBA/hba/v1/author
ization/verify
hsl.hba.tokenvalidation.method=get
hsl.hba.tokenvalidation.queryparams=?id=123&HTTPverb=#method#&token=#jwt#&resource=#
path#
hsl.hba.tokenvalidation.authentication=Basic cmVzdHVzZXI6b3Blbnpvcmc5OQ==
hsl.hba.tokenvalidation.headerparams=hdr1:value1 hdr2:value2
```

11.2.8 Setting user context

Every operation of a REST service must be executed by an OHI officer account (Dutch: functionaris). This is a registered user of the OHI BackOffice application).

The properties below are described for hsl services but also apply to psl services with psl properties.

The default account is set through `hsl.<app>.usercontext` (defaults to `hsl.usercontext`).

If token validation is used, the usercontext is retrieved from the JWT access token.

The following configuration parameters are used:

- `hsl.<app>.usercontext` - defaults to the value of `hsl.usercontext`
  The user context which must be used for executing an operation.
  The value must be a registered OHI officer (in Dutch: 'functionaris').

- `hsl.<app>.usercontext.control` - defaults to the value of `hsl.usercontext.control`
  Allowed values:

  - PROPERTY
    Use the value of `hsl.<app>.usercontext` to set the user context.

  - TOKEN
    Retrieve the user context from the access token.
    Note that `hsl.<app>.usercontext.claim` must be set to indicate which field contains the usercontext.

If `hsl.<app>.usercontext.control` is set to TOKEN, the following configuration parameters control how the usercontext is retrieved:

- `hsl.<app>.usercontext.token.type` - defaults to the value of `hsl.usercontext.token.type`
  Set the type of token.
  Allowed values: JWT

- `hsl.<app>.usercontext.claim` - defaults to the value of `hsl.usercontext.claim`
  Determines which field in the JWT token contains the usercontext. Claim is a standard JSON object of a JTW token. In the example below prn is a reserved claim name for the principal, the subject of the JWT.
  Example: `hsl.<app>.usercontext.claim=prn`

Note that `hsl.<app>.usercontext` must always be set to a valid default value in order to retrieve the Swagger definition.

11.2.9 Overriding User Context with Back Office Parameter

For several HSL services (and not for PSL services!), a Back Office parameter (Dutch: functionaris) has been created to override the user context when for example basic authentication is used.

If the Back Office parameter for setting the user context for a specific service has been set it will overrule the user context as set at the start of the service operation!

If you want to ensure that the user context is set by the `psl.<app>.usercontext` parameter or through the access token, you should remove the value of the 'functionaris' parameter for the given service through the OHI BO application.

12 Appendix F - HSL_AUN and HSL_AUZ Services

Two HSL services were developed for testing the OAUTH2 support of HSL (and PSL), see services: HSL_AUN and HSL_AUZ.

- HSL_AUN
  This service authenticates the username/password account of a database account and, if successful, returns an access token in JWT format.

- HSL_AUZ
  After verifying that the JWT-formatted access token was not compromised, this service should verify that the OHI officer referenced in the token (acquired from HSL_AUN) is authorized for the requested service operation.

The HSL_AUN and HSL_AUZ services may be used by customers to test the OAUTH2 support in the HSL services. A description of OAUTH2 support and the PSL services is found in the previous Appendix.

12.1 Disclaimer

Note that HSL_AUN and HSL_AUZ are only for testing when combined with the HSL services! You can use them as mock-up services to imitate a real OAUTH2 implementation, but should NOT use them to authenticate or authorize requests in a production environment.

In combination with the ZRGOHIJET application and the PSL services they may be used for production purposes.

Also note that at the time of writing, the implementation of the 'postVerify' operation in HSL_AUZ is incomplete.

12.2 HSL_AUN Authentication Service

This service authenticates the username/password account of a database account and, if successful, returns an access token in JWT format.

The HSL_AUN service has a single operation 'postToken' to

- log in to the (OHI Back Office) database using the username and password passed through the 'Credentials' resource.

- Issue a access token in JWT format
  The 'claims' attribute will contain a list of modules for which the principal (OHI officer) is authorized.
  Note: this list is currently empty!

12.3 HSL_AUZ Authorization Service

After verifying that the JWT-formatted access token was not compromised, this service should verify that the OHI officer referenced in the token (acquired from HSL_AUN) is authorized for the requested service operation.

This service has a single operation 'postVerify' to verify that:

- the access token has not expired.

- the access token has not been tampered with

The postVerify operation should also verify that the requested service operation matches with an item in the claims list in the access token. This functionality has not yet been implemented.

12.4 Use of JWT

JWT (JSON Web Token) is emerging as a standard format for access tokens. A JWT is a base64-encoded string consisting of three parts separated by '.' characters:

- header
  Contains encryption method and token type (JWT)

- payload
  Contains principal and claims (priveleges)
  The principal for HSL_AUN and HSL_AUZ is the 'OHI officer' (aka 'functionaris').

- signature
  Checksum based on encrypted header + payload

12.4.1 Payload

The payload may contain the following attributes:

- exp - expirydate in number of seconds since 01-01-1970
  When the token is issued this is the system date + one year.

- iss - token issuer
  Hardcoded: www.oracle.com

- prn - the username of the OHI officer making the request.
  Example: HSL_FUNC_USER

- name - The name of the OHI officer.
  Example: "HSL Web Services"

- claims – a list of modules which can be started by the OHI officer.

12.4.2 Token Verification

The JWT signature contains the encrypted concatenation of the header and payload when the token was issued by HSL_AUN.

When verifying the access token, HSL_AUZ.postVerify recalculates the signature using the header and payload. For this it uses the same algorithm as HSL_AUN.postToken. If the new signature is different from the original signature, the token verification will fail.

12.5 HSL Properties

In addition to the generic HSL properties (see 'Back Office HSL properties file' above), also set the HSL properties in hsl.properties as described in this section before using the HSL_AUN and HSL_AUZ service.

12.5.1 Signature Encryption

The encryption algorithm used by HSL_AUN and HSL_AUZ is driven by the HSL property `hsl.tokenvalidation.rotor=`*`your_secret_key`*

Keep the value of `hsl.tokenvalidation.rotor` secret and limit the access to thje properties file at the OS level! Although HSL_AUN and HSL_AUZ are not currently in production mode for HSL, this may change in the future. And for ZRGOHIJECT and the PSL services production usage is supported.

12.5.2 Authorization

The default value for `hsl.<app>.authorization` is BASIC. This is also the case for the HSL_AUZ service, as any weblogic user can call HSL_AUZ to verify token for token bearer.

The value for HSL_AUN is different: `hsl.aun.authorization=NONE`
Rationale: the postToken operation (as described in the next section) uses the username and password supplied in the Credentials parameter to log into the OHI Back Office database. The operation fails if the credentials are incorrect.

12.6 Deployment

The procedure to deploying the HSL services is described in section '(Re)deployment of the HSL Application'.

Deploy HSL_AUN with 'Custom Roles and Policies'.

Deploy HSL_AUZ with 'Custom Roles and Policies' if you want to fine-tune access through WLS. Deploy with 'Default Descriptors' if any weblogic user may call this service.

12.7 Testing

You may test whether the HSL_AUN and HSL_AUZ services are working by using curl as described earlier.

An example call for testing the HSL_AUN service:

```
  curl -k -i \

  -H Content-Type: application/json \

  -H Accept: application/json \

  -XPOST
https://<backend_url>/HSL_AUN/aun/v1/authentication/token \

  -d "{\"username\": \"cmVuZQ==\",\"password\":
\"d2VsY29tZQ==\",\"grant_type\": \"password\"}"
```

In the example above the username and password string are again base64 encoded (see earlier in this document how to obtain the base64 encoding).

An access token will be returned. This access token is used for accessing a regular service. In the ZRGOHIJET installation manual an example is given of how to pass on such a token to a curl call to test a service.

13 Appendix G – HSL_JUP service

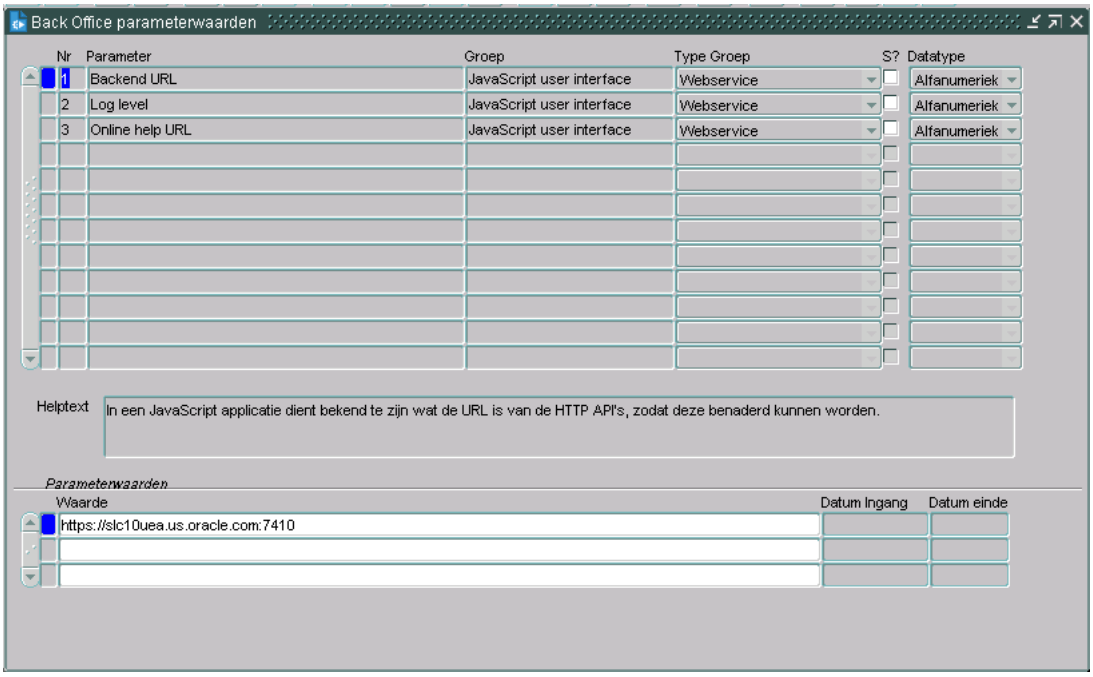
The ZRGOHIJET application (as described in **Doc[3]**) calls HSL_AUN and HSL_AUZ for authentication and authorization of its users. It uses the PSL services (also described in **Doc[3]**) as a back end to access OHI Back Office data.

Before it can do so, it needs to locate a Base URL where these services (all services that are used, so HSL_AUZ, HSL_AUN and the PSL services) can be found.
This is where HSL_JUP comes in. This service, running in the same managed server as ZRGOHIJET itself, will connect to the OHI Back Office database and retrieve the OHI Back Office parameters of group 'Javascript user interface' to retrieve the backend url and log level.

13.1 Back Office parameters

To set the Back Office parameters:

- Open the Forms GUI and select 'Systeem > Beheer > Algemeen > Back Office parameter waarden'.
- Select group 'Javascript user interface' and engage Execute Query.
- Set 'Backend URL' to the Base URL of the HSL_AUN, HSL_AUZ and the PSL services that are used (currently they need to be deployed on the same environment).
  The format is `https://`***`server:port`***
  Example: `https://localhost:8888`
- Optionally, set the log level and Online help URL.



13.2 HSL Properties

In addition to the usual HSL properties (see 'Back Office HSL properties file' above), the HSL property `hsl.jup.authorization` in hsl.properties should be set to a value that differs from the value for the other HSL services. The value must be set to `NONE`. The reason is that anyone must be able to call HSL_JUP to retrieve the OHI BO params for the Javascript UI.

The data source (`hsl.jup.jndiname`) is used to retrieve Back Office Parameters for the ZRGOHIJET Application. The 'Backend Url' parameter should point to the base URL for the HSL_AUN, HSL_AUZ and the PSL services.

13.3 Deployment

The procedure for deploying the HSL_JUP service is described in section '(Re)deployment of the HSL Application'.

Deploy HSL_JUP with 'Custom Roles and Policies'.

14 Appendix H – HSL C2B services – deployment points of attention

Starting with OHI Back Office release 10.17.2.2.0, a set of web service operations has been made available in the HTTP Service Layer in order to replace the 'old' Connect to Back Office (C2B) services that modified policies and relations. These SOAP envelope based services were first delivered in 2006 and have been used for a long time by many OHI customers to send all kinds of policy and relation modification requests to OHI Back Office.

However, as years went by, these services were kept up to date from a functional perspective, but received no technical updates. From a technical and security perspective, they could no longer be supported, because they were based on by now obsolete Java libraries and did not support any form of security.

In 2015 it was announced the C2B services would become obsolete and be replaced by more use case driven HSL services. This intention was frustrated because customers could not agree on a uniform set of commonly accepted service definitions to replace the C2B services. It also became clear that OHI customers could not easily say goodbye to the functionally well known behaviour of the old C2B services. The impact of implementing a new set of functionally different services was perceived as difficult and as a change that would have a large impact on the business.

So, as more strict security requirements demanded phasing out of the old C2B services, time ran out to develop completely new C2B use case services. This resulted in a compromise, where OHI Development would implement a new set of C2B business operations according to the technology used for the existing, modern HTTP Service Layer services, but their functional behaviour would be kept identical, as much as possible.

As a result the C2B 'Use Case services' clearly differ from the other Use Case services that were developed from scratch. The C2B Use Case services should be seen as 'classic C2B operations' being implemented with more modern technology but with a minimum of effort. No new functionality has been added and no new requirements were set, to keep the implementation costs low. It was deemed better to invest later on in new functionality.

This clearly influences the way the C2B Use Case services should be deployed and used. This paragraph focuses on these aspects. Please bear in mind that the old C2B services descend from an OHI era in which it was agreed documentation was not part of the contractual obligations and usage knowledge was transferred as part of consulting activities.

14.1 HSL_C2B deployment aspects

The 'old' C2B operations were made available in 2 different versions:

- A synchronous deployment
- An asynchronous deployment

The asynchronous deployment one required a hard-coded JMS queue and contained a MessageDriven Bean that dequeued the message and called the service. This bean could only be deployed to one WebLogic Managed Server and as such was not scalable In that way it simply processed the offered messages in First in First out order. This had a fortunate side-effect: 2 changes on the same policy were always processed in the correct order provided they were put on the queue in the correct order.

In the new C2B implementation the asynchronous deployment is no longer supported by code in the application. Modern queueing and service implementations offer this functionality out of the box and as such only a synchronous implementation can be deployed. Queues with delayed asynchronous processing should be implemented through standard middleware functionality and put in front of the synchronous implementation

This has deployment implications:

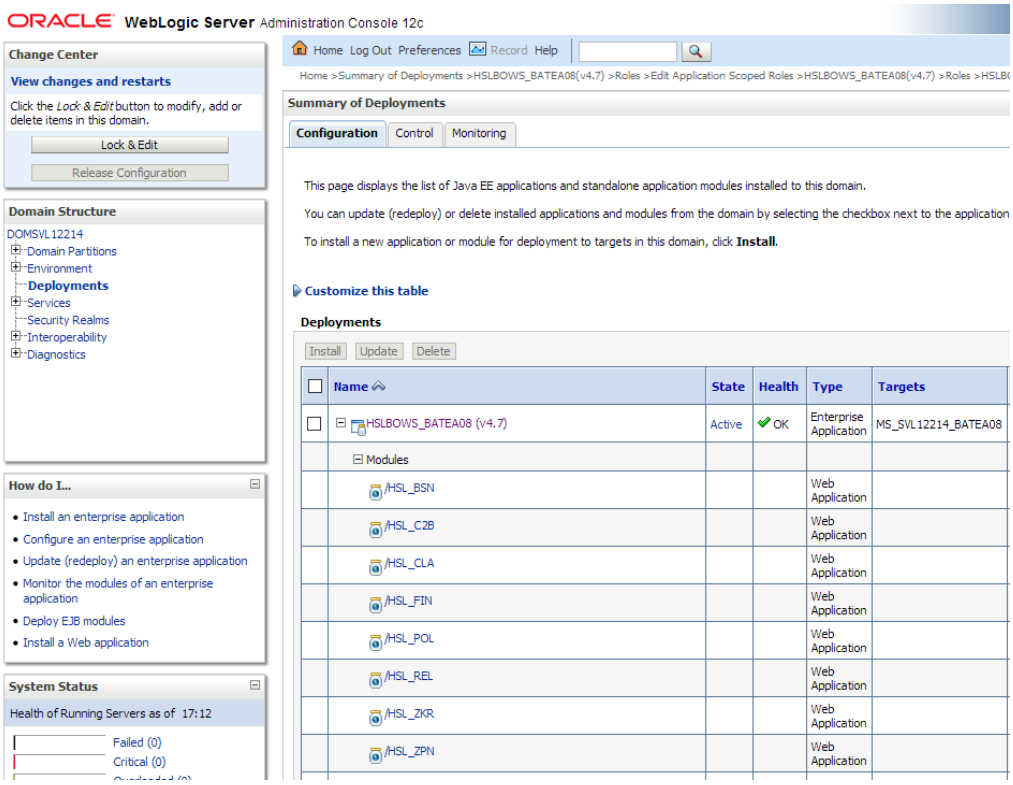
- Scalability is still relatively poor as it is expected messages are processed in FIFO order and it is assumed that changes on the same policy are offered and processed non-concurrently. This means the number of processing threads must be kept to a minimum, preferably one or only a few, provided that dequeueing and processing takes care of the order of processing. A C2B operation for adding a new policy and possibly ending an existing one (a typical functionality from the old C2B SOAP Services) is a relatively resource consuming operation, in which up to a several hundred standard and custom 'policy checks' and other business rules are fired and during which a full copy of the policy to be approved may be created, changed and dropped. This may lead to a response time of several seconds even on modern infrastructure. This response time may increase when synchronous policy processing calls also lead to synchronous callouts towards external providers, like 'GBA' and 'Vecozo'.

- Large amounts of stored messages that may have piled up during a maintenance downtime of OHI Back Office should not be processed through a burst of many parallel threads but must be digested in an ordered manner of (at most) a few parallel processes.

Appendix I – Limiting access to a specific web application inside a module

When HSL applications are deployed using 'Custom Roles and Policies' additional options are available to limit the access to a web application or operation. Weblogic can be configured to limit access through the use of a security policy. To give an idea as to what configuration might involve, an example is shown below.

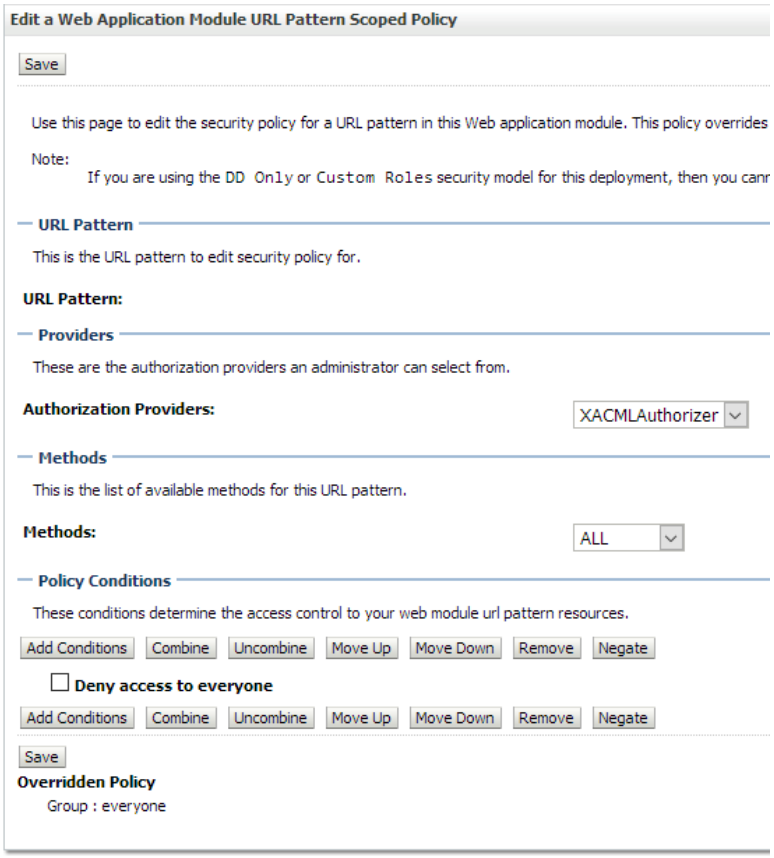
First, expand the HSLBOWS ear-file, revealing the individual HSL applications.



Now to fully disable the HSL_REL service, enter its configuration page by clicking on the link. Click on the tab 'Security' in the resulting page. Finally enter the 'Policies' sub-tab.

As we want to disable access to this specific application in its entirety, adding a policy with an URL pattern of '*' is enough. After the creation of the policy, rules and conditions need to be defined that will be enforced through the policy. Enter the policy's configuration page through clicking on the link named after the URL pattern of the newly created policy.

A number of options are available on the following page to narrow, limit or deny access to a web application. Add a condition and choose the predicate 'Deny access to everyone' from the predicate list. The added condition becomes active immediately after saving the policy at which point all access to the HSL_REL web application is denied.



Note that the created policies survive an update of the ear-file but not a complete reinstall (deletion and subsequent installation of the ear-file). This allows for setting up policies as a one time configuration step and having all future updates adhere to the same (strict) set of policies.