OWF Configuration Guide

DOD GOSS

Exported on Sep 21, 2018

Table of Contents

1 Introduction

1.1 Objectives

This guide covers topics relevant to installing and configuring OWF. It explains extending and customizing the OWF interface and applying themes.

1.2 Document Scope

This guide is intended for OWF developers who wish to configure or customize an OWF instance. For the purpose of this document, a developer is understood to be someone who is comfortable unpacking and packing **WAR** files; editing JavaScript (**JS**), Cascading Style Sheets (**CSS**) and customized configuration files; and configuring OWF.
In this document, the term **Store and Marketplace** are used interchangeably.

1.3 Related Documents

Document	Purpose
User's Guide	Understanding the **OWF user interface**; adding, deleting, modifying **app components** and using **intents**; accessing and using the **Store**; creating, deleting, adding, switching, modifying **app pages**; using **applications**; defining accessibility features such as **high-contrast themes**
Administrator's Guide	Understanding **administrative tools**: adding, deleting, and editing app components, users, groups, applications; **creating default content** for users, groups and group dashboards
Developer's Guide	**Creating app components and descriptor files; integrating app components into OWF**; app component upgrade instructions; walkthroughs for creating app components; adding intents, descriptor URLs, preference API to app components; logging and launching API
Configuration Guide	Overview of **basic architecture and security**; OWF **installation** instructions; instructions for modifying default settings; **database** set up and logging guidance; **framework and theme** customization instructions; OWF **upgrade instructions**; directions for adding and deleting **help** content
Quick Start Guide	Walkthrough of **basic OWF functions** such as using applications; instructions for **setting up a local instance of OWF**, unpacking the OWF Bundle and installing **security certificates**; Truststore/Keystor

2 Overview

2.1 Purpose

OWF is a set of tools, generally delivered in the OWF Bundle. When deployed, OWF is used for organizing and displaying Web applications (application components) in a single browser window known as an OZONE Application.

2.2 Dependencies

The OWF Bundle is shipped with Tomcat v8.5.23 and requires Java 8 or higher. If running OWF with a Web server other than Tomcat, see that Web server's documentation for requirements.

2.3 Components

2.3.1 Ozone Widget Framework (OWF) Web Application

owf.war – This file contains the components which make up OWF. Whether a user logs in and accesses the framework, or an administrator logs in to modify preferences, the **owf.war** is the application that launches those pages to the browser.

2.3.2 Pluggable Security

OWF allows an administrator to customize the type of security that will be implemented for user authentication and authorization. Included within OWF's **/ozone-security** directory are **XML** files that provide examples of optional security configurations. **They are intended as examples and should in no way be used in a production environment.** However, they can be used as the basis for creating a custom security plug-in.
Along with the security-related **XML** files, there is also a **ZIP** file which contains the source and configuration files for the pluggable security modules. Additionally, an Apache **ANT** build script is included. The build script allows for a rebuild of the OWF security **JAR** file for customization.
Also, **ozone-security\ozone-security-project\src\main\resources\conf\sample-log4j.xml** contains a subset of the **OWF-log4j.xml** file (found in the**tomcat\webapps\owf\WEB-INF\classes** directory) that pertains to security settings. It demonstrates how to enable logging for OWF Security.
Note: Refer to section 5: OWF Security for more details about OWF Security.

2.3.2.1 Default Authentication

OWFsecurityContext.xml - Contains the default security implementation and uses a PKI certificate for client authentication. If a PKI certificate is not provided for authentication, the HTTP-BASIC authentication method will be used, prompting the user for a username and password.
NOTE: As of version 7.5.0 of OWF, the default configuration has changed. In prior versions, the default was the configuration which is now stored in OWFsecurityContext_x509_CAS.xml.

2.3.2.2 X509-Only Security

OWFsecurityContext_cert_only.xml - This contains the X509-only security implementation for OWF. It uses a PKI certificate for client authentication. If no authentication is provided, the user is denied access to the system.

2.3.2.3 OWFsecurityContext_x509_CAS Security

OWFsecurityContext_x509_CAS.xml—This contains an example of how the CAS and x509 authentication mechanisms can be combined. This configuration first attempts to authenticate the user using a PKI certificate. If a certificate is not provided, this configuration will redirect the

user to a CAS server for authentication via whatever mechanism that CAS server is configured to use.

2.3.2.4 CAS-Only Security

OWFsecurityContext_CAS_only.xml - This contains the CAS-only security implementation for OWF. If the sign-in is invalid, the user will be denied access to the system.

2.3.2.5 X509/LDAP

OWFsecurityContext_cert_ldap.xml - This contains an X509/LDAP security implementation that uses X509 for authentication and then performs an LDAP-based lookup to determine the user's authorization.

2.3.2.6 Basic Spring Login

OWFsecurityContext_BasicSpringLogin.xml - This contains a basic form log-in based implementation that uses names populated in the **XML** file to determine the user's access.

2.3.2.7 OWF Security Project

ozone-security-project.zip - This bundle contains the source code, configuration files and library files needed to build the security files which are used by OWF. Additionally, an Apache **ANT** build script is included for the building of the **JAR** file which is used by the following security **XML** files.
The **ozone-security-project.zip** contains the following supporting resource files:

- **src/main/resources/conf/apache-ds-server.xml**, a sample **XML** file used by Apache Directory Server (ApacheDS, an open-source LDAP v3 compliant embeddable directory server) that sets up the initial directory service partitions for the test data.
- **src/main/resources/conf/testUsers.ldif**, an LDAP Data Interchange Format test file that can be imported to set up test user entries that match the certificates bundled with OWF.
- **lib/spring-security-ldap-3.0.2.RELEASE.jar**, a file which provides LDAP functionality used by the Ozone-LDAP-Security plugin.

Note: While the same PKI files used in sections 2.3.2.1: Default Authentication and 2.3.2.2: X509-Only Security are used for the authentication of users, the LDAP files in section 2.3.2.5(above) are used for their authorization.

2.3.3 Sample Widgets

OWF provides sample widgets in the **owf-sample-widgets.zip** file and the **owf\examples** directory located inside the unpacked **owf.war**.
The samples employ various Web technologies. They can be used as a starting point for investigating different widget implementation strategies. The follow table references a few specific examples that demonstrate how to invoke or integrate JavaScript with different technologies. Additional example widgets are included in the **examples** directory in **owf.war**. Also, the OWF Developer's Guide includes specific examples and walkthroughs regarding the widget APIs.
Table 2: Example Widgets

Widget	WebTechnology	Description	Widget Location
DotNet	NET	A Web page for adding eventing channels and a Web service for storing received messages.	**owf-sample-widget.zip**
Flex Pan	Flex	Displays a large image of the Earth and allows users to zoom and scroll around the image.	**owf-sample-widget.zip**

Widget	WebTechnology	Description	Widget Location
Flex Direct	Flex	Connects to the Flex Pan widget via the eventing mechanism to control the panning and zooming; also displays the current mouse position.	**owf-sample-widget.zip**
Channel Shouter Flex	Flex	Demonstrates eventing and widget launcher APIs created in Flex.	**owf-sample-widget.zip**
Channel Listener Flex	Flex	Demonstrates eventing and widget launcher APIs created in Flex.	**owf-sample-widget.zip**
OWF Silverlight Demo	Silverlight	Allows a user to send messages to other widgets in the framework, register channels to listen on, and track the frequency of the messages within the registered channels on either a pie chart or a bar chart.	**owf-sample-widget.zip**
My Chess Viewer	Java Applet	Reads and animates a chess **PGN** file, incorporates it into the framework, broadcasts each forward move selected on a channel named "mychess", and stores the current position in the game to the Preference Service.	**owf-sample-widget.zip**
Announcing Clock	HTML	A clock that broadcasts to an HTML page; optionally displays military time.	**owf-sample-widget.zip**
Stockwatcher	GWT	Broadcasts a message on the "stockwatcher" channel when stocks are added or removed; saves stock symbol picks to the Preference Service.	**owf-sample-widget.zip**
NYSE Widget	GSP	Part of the widget intents example, this widget sends "view" and "graph" intents to receiving widgets.	**owf.war**
Stock Chart	GSP	Part of the widget intents example, this widget receives data from graph intents.	**owf.war**
HTML Viewer	GSP	Part of the widget intents example, this widget receives data from view intents.	**owf.war**

3 Installation

3.1 Dependencies

Listed below are the dependencies for OWF:

- Java 8 or higher.
- A Relational Database Management System (RDBMS). OWF currently ships with an in-memory HyperSQL (HSQLDB) database for testing and development purposes, but it is expected that a live deployment will use a more robust RDBMS such as Oracle or MySQL.

3.2 Supported Browsers

OWF is tested against the following browsers:
Table 3: Tested Browsers

Browsers	Versions
Internet Explorer	11
Firefox	57
Chrome	43
Edge	38

3.3 OWF Bundle Description

The distribution of OWF consists of a **ZIP** file containing the necessary components to set up and run OWF in a development environment. The bundle contains the following:

- Tomcat (Simple Java Web Container)
- Sample PKI Certificates for SSL (sample user certificates and server certificate)
- OWF Web application (**owf.war**)
- Externalized Security Configurations found in the ozone-security directory located inside **OWF-bundle-7.17.2.0-RC1.zip**.
- Tomcat start scripts (**start.sh** or **start.bat**)
- The following developer-configurable externalized properties files:
    - **OzoneConfig.properties**

The following example shows how an administrator might copy, unzip and start OWF from the bundle deployment on ***nix** operating systems, assuming the bundle is named **OWF-bundle-7.17.2.0-RC1.zip**:

```
cp OWF-bundle-7.17.2.0-RC1.zip/opt/.
cd /opt
unzip OWF-bundle-7.17.2.0-RC1.zip
```

```
cd tomcat
./start.sh
```

The following example shows how an administrator might copy, unzip, and start OWF from the bundle on **Windows** operating systems, assuming the bundle is named **OWF-bundle-7.17.0.zip**:

1. Create a new directory from where OWF will be run. This can be done via the *Windows* UI or a command prompt.
2. Copy **OWF-bundle-7.17.2.0-RC1.zip** to the new directory created in step 1.
3. Right-click on **OWF-bundle-7.17.2.0-RC1.zip** and select **open**, **explore** or the command for the system's default zip/unzip program.
4. Unzip/unpack the bundle into the new directory created in step 1.
5. From a command-line, run **start.bat** from within the **tomcat directory.**

The use of the bundled deployment archive provides all of the necessary mechanisms to deploy and run the Tomcat Web container on any Java 8+ enabled system.

3.4 Default Installation

Running the OWF Bundle via the included Tomcat web server with the default values requires minimal installation. With standard configuration, OWF makes use of the default authentication module, which provides X509 authentication/authorization.
Note: If OWF 7.15 is installed as an u pgrade, please see Appendix A: Upgrading OWF.
The application uses a **KeyStore** and a **Truststore** which are local to the installation. There is no need to install any certificates into the server's Java installation. The default certificates contained in the OWF Bundle only function for **localhost** communications. When accessed from a remote machine with a name that differs from **localhost**, while using the included certificates, OWF will not function correctly. Accordingly, see 3.5.5: Server Certificate Creation and Installation for information about creating additional certificates.

3.4.1 Installing User PKI Certificates

By default, the security infrastructure of the OWF Bundle is configured to use client certificates. In order to identify themselves via certificates, clients need to install a PKI certificate into their Web browser. The client certificates that are included with the OWF bundle will be recognized immediately and can be used in the default security configuration. The certificates are located in the **\tomcat\certs**directory of the OWF bundle.
The default client certificates can be used by importing the included **testUser1.p12** or **testAdmin1.p12** certificate into the user's browser. In Internet Explorer, client certificates can be added by selecting Tools Internet Options Content Certificates Personal, and then clicking the Import button. The certificate **testUser1** grants rights to use the application, while **testAdmin1** is a certificate for a user granted both user rights and administrator rights. The private key password for both certificates is **password**.
In Firefox this menu is accessed via Tools Options Advanced Encryption View Certificates Your Certificates Import.

3.5 Custom Installation

OWF can be customized to run in a variety of environments. The following sections detail how to change default database settings and set up security.
Note: If OWF 7.15 is being installed as an upgrade, please see Appendix A: Upgrading OWF.

3.5.1 Database Setup

While the full extent of administering databases is outside the scope of this guide, this section provides information on how to work with databases for OWF.

Application.yml is a configuration file that allows an administrator to modify database connectivity information. It is located in the **\tomcat\webapps\owf\WEB-INF\classes** directory. Once changes are made, restart OWF to apply them. Developers comfortable with the Groovy language and the Grails Web application framework should be comfortable writing additional code for the file.
Listed below are the variable database elements that need to be modified to customize the OWF database. A detailed explanation of each field follows in Table 4: OWF Externalized Database Properties:

dataSource:
pooled: true
dbCreate: "none"
username = "sa"
password: ""
driverClassName: "org.hsqldb.jdbcDriver"
url: "jdbc:hsqldb:file:prodDb;shutdown=true"
pooled: true
properties:
minEvictableIdleTimeMillis: 180000
timeBetweenEvictionRunsMillis: 180000
numTestsPerEvictionRun: 3
testOnBorrow: true
testWhileIdle: true
testOnReturn: true
validationQuery: "SELECT 1 FROM INFORMATION_SCHEMA.SYSTEM_USERS"

Table 4: OWF Externalized Database Properties

Property	Purpose	Example
dbCreate	The way the database is created/updated when the server is started *Note: Use the appropriate database creation script in the dbscript folder before running OWF.*	None
username	The Username for the database connection	admin
password	The password for the database connection	Password

Property	Purpose	Example
driverClassName	JDBC driver	org.hsqldb.jdbcDriver
url	JDBC Connection String	jdbc:hsqldb: file:prodDb . ; shutdown=true
pooled	Enable database connection pooling when true	True
minEvictableIdleTimeMillis	Minimum amount of time in milliseconds an object can be idle in the pool before becoming eligible for eviction	18000
timeBetweenEvictionRunsMillis	Time in milliseconds to sleep between runs of the idle object evictor thread	18000
numTestsPerEvictionRun	Number of objects to be examined on each run of the idle evictor thread	3
testOnBorrow	When true, objects are validated before borrowed from the pool	true
testWhileIdle	When true, objects are validated by the idle object evictor thread	true
testOnReturn	When true, objects are validated before returned to the pool	true
validationQuery	Validation query, used to test connections before use *Note: Syntax*	SELECT 1 FROM INFORMATION_SCHEMA.SYSTEM_USERS

Property	Purpose	Example
	varies by database, see the examples included in this document.	

Note: When setting up databases for OWF, be mindful of the database's Lexical sorting mechanism. For some instances of OWF, with a small handful of users, this may not be much of an issue, but as the database becomes more populated, sorting may become increasingly difficult to manage.

3.5.1.1 Using Oracle

1. Create an Oracle database user for OWF. It is recommended that there be a dedicated user for OWF to avoid database object name collisions. The OWF team recommends using UTF-8 encoding.
2. Due to licensing issues, OWF does not provide a JDBC driver for Oracle. Obtain the appropriate JDBC driver and place it into the Web server's classpath. For example, if running Tomcat, the driver can be placed in the **\tomcat\lib** directory.
3. Open the **\tomcat\webapps\owf\WEB-INF\classes\application.yml** file and modify the **environments à production à dataSource** section using the values that are appropriate for the OWF environment. For example:

```
dataSource:
    pooled: true
    dbCreate: "none"
    username: "owf_user"
    password: "owf_password"
    dialect: "org.hibernate.dialect.Oracle10gDialect"
    driverClassName: "oracle.jdbc.driver.OracleDriver"
    url: "jdbc:oracle:thin:@myhost.somewhere.org:1521:DEVDB1"
    properties:
        minEvictableIdleTimeMillis: 180000
        timeBetweenEvictionRunsMillis: 180000
        numTestsPerEvictionRun: 3
        testOnBorrow: true
        testWhileIdle: true
        testOnReturn: true
        validationQuery: "SELECT 1 FROM DUAL"
```

In the example above, an Oracle database-user named **owf_user** with a password of **owf_password** is used for a database named DEVDB1.

There are several different types of Oracle drivers (thin, OCI, kprb) and connection options (service, SID, TNSName) available. Please consult the Oracle DBA and Oracle's JDBC documentation to create the connection most appropriate for the installed environment.

1. To create the schema, run the **\dbscripts\oracle\OracleCreate.sql** script, prior to starting OWF.
2. Ensure that the transaction is committed.

*Note: If running a production environment, no additional steps are necessary. However, if sample widgets are to be installed, **OraclePrefsUpdate_v7.14.0.sql** must be run prior to logging in. Logging in between the execution of these scripts can cause system failure.*
Note: The OWF team is aware of a known issue with the Oracle Web-based Admin Console

returning truncated characters when dealing with large data sets. Accordingly, using SQLPlus to run the script mentioned above is recommended.

Note: Old versions of the SQL scripts are included under \dbscripts\archive. These are included for convenience for people upgrading from older versions of OWF.

3.5.1.2 Using MySQL

- Create a schema within MySQL for use with OWF. It is recommended that there be a dedicated schema for OWF to avoid database object name collisions. The OWF team recommends using UTF-8 encoding.
- Create a MySQL User with full access to the OWF schema created above.
- OWF does not provide a JDBC driver for MySQL. Obtain the appropriate JDBC driver and place it into the Web server's classpath. For example, if running Tomcat, the driver can be placed in the **\tomcat\lib** directory.
- Open the **\tomcat\webapps\owf\WEB-INF\classes\application.yml** file and modify the **environments à production à dataSource** section using the values that are appropriate for the OWF environment.

For example:

```
dataSource:
pooled: true
dbCreate: "none"
driverClassName: "com.mysql.jdbc.Driver"
url: "jdbc:mysql://myhost.somewhere.org/owf"
username: "owf_user"
password: "owf_password"
dialect: "org.hibernate.dialect.MySQL5InnoDBDialect"
properties:
minEvictableIdleTimeMillis: 180000
timeBetweenEvictionRunsMillis: 180000
numTestsPerEvictionRun: 3
testOnBorrow: true
testWhileIdle: true
testOnReturn: true
validationQuery: "SELECT 1"
```

In the example above, a MySQL database-user named **owf_user** with a password of **owf_password** is used, for a database named **owf**. The dialect **org.hibernate.dialect.MySQL5InnoDBDialect** will use the **InnoDB** engine which is recommended for interactive **webapps** and explicitly used as the engine on OWF create and upgrade scripts.

- Create the schema by running the **\dbscripts\mysql\MySqlCreate.sql** script, prior to starting OWF.

*Note: If manually creating the database objects, be sure to modify the **SQL** script (mentioned above) with the appropriate schema name. For example:*
use owf;
*Note: If running a production environment, no additional steps are necessary. However, if sample widgets are to be installed, **MySqlPrefsUpdate_v7.14.0.sql** must be run prior to logging in. Logging in between the execution of these scripts can cause system failure.*

Note: Old versions of the SQL scripts are included under \dbscripts\archive. These are included for convenience for people upgrading from older versions of OWF.

3.5.1.2.1 Configuring MySQL JDBC to use SSL

By default MySQL communicates over an unencrypted connection - however, in most cases, it can be configured to use SSL. This is somewhat implementation specific.

Note: This capability completely depends on how the implementation's MySQL binaries were compiled, packaged, etc.

The following procedure covers how to configure an SSL capable MySQL server to work with an OWF bundle. The starting point for the server implementation used in this example is:

- Operating System: CentOS 6.4, 64 bit minimal installation (no updates were applied)
- MySQL Server v5.1.69 (achieved by installing the "mysql-server" package with yum)

This procedure was developed from the MySQL 5.1 documentation, specifically the following sections:

- Using SSL for Secure Connections
- Connector/J (JDBC) Reference

This procedure relies on self-signed certificates. It is for testing and demonstration purposes only. The following three sections explain the configuration steps:

Step 1: Creating MySQL Server's CA and Server Certificates

The following steps guide a developer through creating the CA and Server certificates for a MySQL server:

1. **Create the CA Key and Certificate** From a shell prompt on a MySQL server, type the following commands:

```
prompt> openssl genrsa 2048 > ca-key.pem
prompt> openssl req -new -x509 -nodes -days 365 -key ca-key.pem -
out cacert.pem
```

Note: After the second command, the system prompts the developer to provide basic identity information. For the purpose of this demonstration, it is not important what information the developer provides here. However, it will be necessary to provide the same information in the next step.

2. Create the Server Certificate From a shell prompt on your MySQL server, type the following commands. After the first command, the developer will again be prompted to provide identity information. The developer must provide the same information that they provided in Step 1. However, when prompted for the Common Name, provide the MySQL server's hostname (e.g. mysql.mydomain.com).

```
prompt> openssl req -newkey rsa:2048 -days 365 -nodes -keyout server-
key.pem -out server-req.pem
prompt> openssl rsa -in server-key.pem -out server-key.pem
prompt> openssl x509 -req -in server-req.pem -days 365 -CA cacert.pem -
CAkey ca-key.pem -set_serial 01 -out server-cert.pem
```

3. Consolidate the Output from Steps 1 & 2

Copy the following files (produced in the preceding two steps) to a location where at a minimum the MySQL user has read access. In this example, it is **/etc/ssl/certs/**

```
cacert.pem
server-key.pem
server-cert.pem
```

3.5.1.2.1.1 Step 2: Configure MySQL

1. **Edit my.cnf** Edit the MySQL configuration file. For this example, the file is located at **/etc/my.cnf**. Add the lines shown in bold to the [mysqld] section of the file.

```
ssl-ca=/etc/ssl/certs/cacert.pem
ssl-cert=/etc/ssl/certs/server-cert.pem
ssl-key=/etc/ssl/certs/server-key.pem
```

2. **Restart MySQL**

```
sudo service mysqld restart
```

3. **Create the MySQL Franchise Store Schema** At a minimum, create the schema and assign permissions as you would normally, following the steps in section Adding Users/Roles/Groups. This will leave the choice of using an encrypted connection up to the connecting client. To enforce the use of SSL from the database server side, add REQUIRE SSL to the end of the grant statement where user permissions to the Franchise Store schema are assigned. For example:

```
GRANT ALL ON franchise.* TO 'franchise'@'storehost.mydomain.com'
IDENTIFIED BY 'franchise' REQUIRE SSL;
```

3.5.1.2.1.2 Step 3: Configure the Franchise Store Bundle

1. Modify application.yml Add **useSSL=true** to the **dataSource URL** configuration. This can also be enforced from the client side with the **requireSSL** option. For example:

```
url = """jdbc:mysql://192.168.56.11/franchise?useSSL=true&
requireSSL=true&trustCertificateKeyStoreUrl=
file://$\{System.properties['javax.net.ssl.trustStore']}&
trustCertificateKeyStorePassword=changeit"""
```

2. Modify the Application TrustStore Add the CA certificate, created above (**cacert.pem**) to the application's trust store. In the case of the franchise bundle, the trust store is a file called **keystore.jks**found in **$BUNDLE_PATH/tomcat/certs**. Do this with the following command (assuming you have the JDK installed and JAVA_HOME/bin on your PATH, if they aren't there, add them first).

```
prompt> keytool -import -alias mysqlCAcert -file cacert.pem -keystore
keystore.jks
```

3. Start the Application

```
prompt> ./start.sh
```

3.5.1.3 Using PostgreSQL

1. Create either a new login role or a new schema in order to avoid database object name collisions between OWF and other database applications.
2. Edit the user so that it can create database objects.
3. Create a new database. Use UTF-8 as encoding (default).
4. OWF does not provide a JDBC driver for PostgreSQL. Obtain the appropriate JDBC driver and place it into the Web server's classpath. For example, if running Tomcat, the driver can be placed in the **\tomcat\lib** directory.
5. Open the **\tomcat\webapps\owf\WEB-INF\classes\application.yml** file and modify the **environments -> production -> dataSource** section using the values that are appropriate for the OWF environment. For example:

```
dataSource:
    pooled: true
    dbCreate: "none"
    username: "owf_user"
    password: "owf"
    driverClassName: "org.postgresql.Driver"
    url: "jdbc:postgresql://localhost:5432/OWF"
    dialect: "org.hibernate.dialect.PostgreSQLDialect"
    properties:
        minEvictableIdleTimeMillis: 180000
        timeBetweenEvictionRunsMillis: 180000
        numTestsPerEvictionRun: 3
        testOnBorrow: true
        testWhileIdle: true
        testOnReturn: true
        validationQuery: "SELECT 1"
```

In the example above, a PostgreSQL database user named **owf_user** with a password of **owf** is used, for a database named OWF.

1. Create the schema by running the **\dbscript\postgres\PostgresqlCreate.sql** script before starting OWF.
2. If OWF is being installed as a production environment, this can serve as the final step. However, if sample data is required (e.g., creating a testing environment), the remaining steps can be followed.

Note: If running a production environment, no additional steps are necessary. However, if sample widgets are to be installed, PostgreSQLPrefsUpdate_v7.14.0.sql must be run prior to logging in. Logging in between the execution of these scripts can cause system failure.

Note: Old versions of the SQL scripts are included under \dbscripts\archive. These are included for convenience for people upgrading from older versions of OWF.

3.5.1.4 Using SQL Server

1. Create a new SQL Server database for use with OWF.
2. Create a SQL Server user with full access to the OWF database created above.
3. OWF does not provide a JDBC driver for SQL Server. Obtain the appropriate JDBC driver and place it on the Web server's classpath. For example, if running Tomcat, the driver can be placed in the **\tomcat\lib** directory.
4. Open the **\tomcat\webapps\owf\WEB-INF\classes\application.yml** file and modify the **environments -> production -> dataSource** section using the values that are appropriate for the OWF environment. For example:

```
dataSource:
    pooled: true
    dbCreate: "none"
    username: "owf_user"
    password: "owf"
    driverClassName: "net.sourceforge.jtds.jdbc.Driver"
    url: "jdbc:jtds:sqlserver://localhost:1443/OWF"
    dialect: "ozone.owf.hibernate.OWFSQLServerDialect"
    properties:
        minEvictableIdleTimeMillis: 180000
        timeBetweenEvictionRunsMillis: 180000
        numTestsPerEvictionRun: 3
        testOnBorrow: true
        testWhileIdle: true
        testOnReturn: true
        validationQuery: "SELECT 1"
```

In the example above the SQL Server database user named **owf_user** with password of **owf** is used, to access a database named OWF.

1. Create the schema by running the **\dbscripts\sqlserver\SQLServerCreate.sql** script, prior to starting OWF.

If sample data is required (e.g., creating a testing environment), the remaining steps can be followed.

Note: If sample data scripts (as mentioned in step 5) are to be run, script execution must take place before logging into OWF. Logging in before the execution of the script can cause system failure.

1. Open Microsoft SQL Server Management Studio (or another database editing tool) and select File Open File.
2. Navigate to **\dbscripts\SQLServerPrefsUpdate_v7.0.0.sql** in the bundle.

*Note: This script should **ONLY** be run against an empty database as it will delete pre-existing data.*

1. Select the OWF database, and execute the script.

Note: Old versions of the SQL scripts are included under \dbscripts\archive. These are included for convenience for people upgrading from older versions of OWF.

3.5.2 Security Setup

OWF provides a modular security approach that is based on Spring Security. All of the provided options supply both a Spring Security configuration file and Java classes that are written to Spring's security interfaces in order to perform authentication and authorization. For more details, please refer to section 5: OWF Security.

Installing the Security Module

The OZONE-security files, provided in the distribution bundle, offer multiple examples of security options. These are intended as examples and should in no way be used in a production environment. The default security implementation provides an X509 certificate authentication. If the user does not provide a client X509 certificate, they are presented with a dialog box allowing them to log in with a username and password.

For each available security option, there is a specific **XML** file which must be installed. Installing a new security module is accomplished in just a few simple steps. For more details, please refer to section 5: OWF Security.

3.5.3 Operating OWF From Different/Multiple Ports

Initial OWF configuration is set up so that Tomcat can be run from a local installation. Throughout this document, **servername:port** implies a **localhost:8080** or **localhost:8443** location. The example below shows how to set up OWF so that it can be used on **5050/5443**.

To enable ports other than **8080/8443**, the desired ports need to be explicitly edited in the Web server configuration file: **conf/server.xml**.
Note: In the event that OWF is running on a server where a port number is already in use, OWF must run from a different port number. Two applications cannot bind to the same port.

1. For example, in Tomcat, change the port numbers in **conf/server.xml**, as shown below to the following port number:

```
<Connector port="5050" protocol="HTTP/1.1"
connectionTimeout="20000"
redirectPort="5443" />
Also modify the following sections of the file:
<Connector port="5443" protocol="HTTP/1.1" SSLEnabled="true"
maxThreads="150" scheme="https" secure="true"
keystoreFile="certs/keystore.jks" keystorePass="changeit"
clientAuth="false" sslProtocol="TLS" />
```

1. Ports 5050 and 5443 are just examples and can be changed to whatever is needed.If OWF was running on a server where a port number was already in use, the **shutdown** port must also be changed. To do this, change the port number in the Tomcat Web server configuration file **\tomcat\conf\server.xml** to another port, in the following example the default shutdown port was changed from 8005 to 8006:

```
<Server port="8006" shutdown="SHUTDOWN">
```

b. Ensure that the port value used in the Web server configuration file matches the port value used in **\tomcat\lib OzoneConfig.properties** which is shown below, displaying the default port and host information:

```
ozone.host = localhost
ozone.port = 5443
ozone.unsecurePort = 5050
```

1. Save both files.
2. Restart the OWF server.

3.5.4 Adding the Store or Metrics Service To OWF

The flexible and scalable nature of OWF allow for applications used in concert (such as the Store or the Metrics Service) to be included in OWF's deployment. This allows a user to develop with the products working together, without having to activate multiple ports via configuration. To include the Store or Metrics Service in the OWF bundle, do the following:

1. Unpack the zipped bundles containing the applications to be included.
2. Navigate to **tomcat/webapps** in each unpacked bundle.
3. Copy the appropriate **WAR** files into the **tomcat/webapps** directory where OWF was deployed.
4. Copy the additional applications' configuration files into **tomcat/lib** where OWF was deployed. For the Store, the configuration file is **MarketplaceConfig.groovy**. For the metrics service, it is **MetricConfig.groovy**
5. Restart the OWF server.

*Note: If using a Marketplace release earlier than version 5, the following file must also be copied to the deployed OWF's **/tomcat/lib** directory:*
/tomcat/lib/MPsecurityContext.xml

3.5.5 Server Certificate Creation and Installation

Valid server certificates are needed for configuring the server to allow https authentication.
Note: Self-signed certificates will produce warnings in a user's browser. This is because a self-signed certificate, not signed by a recognized certificate authority, has no one authorizing its validity. In a production environment, certificates should be signed by a recognized certificate authority, such as an organization's internal certificate authority.

3.5.5.1 Generating a New Self-Signed Server Certificate

A new self-signed certificate can be generated by navigating to the **\etc\tools** directory and executing **create-certificates.bat** or .**sh**, depending on the operating system in use.

Follow the on-screen prompts and create the necessary certificates for the installation.
Make sure to enter the FULLY QUALIFIED server name. This needs to match the domain name of the machine exactly or the certificate will not work correctly.
If using an IP address as the Common Name (CN), an entry must be added to the Subject Alternative Name entry in the certificate. The better alternative to using an IP address is to add a name/IP pair to the hosts file and register the name as the CN.

3.5.5.2 Configuring OWF For a Different Truststore/Keystore

1. For server-to-server calls (OWF-to-CAS communications, for example) the newly created self-signed certificate should be imported into the truststore. If the truststore is a separate file from the keystore, the certificate can be copied from the keystore to the truststore as follows:

    a. Export the certificate from the KeyStore into a file:

```
keytool -export -file servername.crt -keystore servername.jks -alias
servernam
```

1. Import the file into the Truststore:

```
keytool -import -alias servername -keystore mytruststore.jks -file
servername.crt
```

2. Modify the JVM Parameters that are used to start the Web application server in order to use the new Truststore shown above. If a Tomcat server is being used, the parameters can be found in the **setenv.bat** (or .**sh**, depending on the operating system in use) script found within the **\tomcat\bin** folder inside of the unpacked **OWF-bundle-7.17.0.zip**. If an application server other than Tomcat is being used, the parameters will need to be added to the JVM parameters which are loaded when the application server is started.

Table 5: Custom JVM Parameters

Parameter	Note
-Djavax.net.ssl.keyStore= "%CATALINA_HOME%\certs\keystore.jks"	Replace 'certs/keystore.jks' with the path and filename to the keystore.
-Djavax.net.ssl.trustStore= "%CATALINA_HOME%\certs\keystore.jks"	Replace 'certs/keystore.jks' with the path of the truststore (May be the same as the keystore).
-Djavax.net.ssl.keyStorePassword= changeit	Replace 'changeit' with the keystore's password (if applicable)
-Djavax.net.ssl.trustStorePassword=changeit	Replace 'changelt' with the truststore's password.

Finally, the server configuration must be modified to use the new KeyStore/Truststore in SSL. Below is the relevant section from the Tomcat configuration script found in **\tomcat\conf\server.xml**:

```
<Connector port="8443"
protocol="HTTP/1.1"
SSLEnabled="true"
maxThreads="150"
scheme="https"
secure="true"
keystoreFile="certs/keystore.jks"
keystorePass="changeit"
truststoreFile="certs/truststore.jks"
truststorePass="changeit"
clientAuth="want"
sslProtocol="TLS" />
```

The distribution of OWF consists of a ZIP file containing the necessary components to set up and run OWF in a development environment. The bundle contains the following:

· Tomcat (Simple Java Web Container)

· Sample PKI Certificates for SSL (sample user certificates and server certificate)

· OWF Web application (owf.war)

· Externalized Security Configurations found in the ozone-security directory located inside **OWF-bundle-7.17.2.0-RC1.zip** .

· Tomcat start scripts (start.sh or start.bat)

· The following developer-configurable externalized properties files:

o OzoneConfig.properties

The following example shows how an administrator might copy, unzip and start OWF from the bundle deployment on *nix operating systems, assuming the bundle is named **OWF-bundle-7.17.2.0-RC1.zip**:

cp **OWF-bundle-7.17.2.0-RC1.zip**/opt/.

cd /opt

unzip **OWF-bundle-7.17.2.0-RC1.zip**

cd tomcat

./start.sh

The following example shows how an administrator might copy, unzip, and start OWF from the bundle on **Windows** operating systems, assuming the bundle is named **OWF-bundle-7.17.2.0-RC1.zip**:

1) Create a new directory from where OWF will be run. This can be done via the *Windows* UI or a command prompt.

2) Copy **OWF-bundle-7.17.2.0-RC1.zip** to the new directory created in step 1.

3) Right-click on **OWF-bundle-7.17.2.0-RC1.zip** and select open, explore or the command for the system's default zip/unzip program.

4) Unzip/unpack the bundle into the new directory created in step 1.

5) From a command-line, run start.bat from within the tomcat directory.

The use of the bundled deployment archive provides all of the necessary mechanisms to deploy and run the Tomcat Web container on any Java 8+ enabled system.

4 Configuration

4.1 Default Configuration

The OWF Bundle is configured to run by default on localhost with a predefined set of users. When using the default Tomcat bundle, externalized configuration files should be placed in the folder **\tomcat\lib**. If using an application server other than Tomcat, copy the override files into the directory that will include them in the classpath for that specific application server.

4.1.1 Adding Users/Roles/Groups

The addition of users, groups, and roles into OWF depends on the choice of security implementation. The following example outlines the procedures for adding users, groups, and roles to the sample OWF X509-only, CAS-only, and X509-with-CAS security modules:
Note: The sample security modules are included as examples and should NOT be used in a production environment. For more information, please refer to section 5: OWF Security.

1. Edit **\tomcat\lib\users.properties**

```
…
testUser1=password,ROLE_USER,Test User 1,[group1;I am a sample Group 1
from users.properties;test@gmail.com;active]
testUser2=password,ROLE_USER,Test User 2
testUser3=password,ROLE_USER,Test User 3
testAdmin1=password,ROLE_ADMIN,Test Admin 1,[group1;I am a sample Group 1
from users.properties;test@email.com;active],[group2;I am a sample Group
2 from users.properties;test2@email.com;active],[group3;I am a sample
Group 3 from users.properties;test3@email.com;inactive]
…
```

Note: To have actual spaces between names and numbers, escape spaces using the "\" (do not include the quotation marks) character.

2. Add users to the file in accordance with the following rules:

1. Data Format: Username=password, role, display name,[group name, group description, group contact email, active/inactive status].
2. All of the information for a single user, including group information, should be on a single line.
3. Multiple groups may be delimited by commas.
4. Group information is optional, and may be left out for any single user.
5. Once a group has been created for the first time, the description, contact email, and active/inactive status will not affect those values within OWF—that information must be managed through the OWF Group Manager administrative widget.

3. Save the file and restart the OWF server.

Any user added to **users.properties** will be granted access to OWF upon restart.
Any user deleted from **users.properties** will be denied access to OWF upon restart.
*Note: If a custom **webserver** is being used along with the provided example security, the **users.properties** file can be copied to any directory that is on the classpath of the Web server in use. For example, if using Jetty, the file can be copied to the **\<jetty root>\resources** directory.*
To add users to any security module use X509 authentication, generated a PKI User Certificate that can be recognized by OWF.

4.1.2 Help Content Configuration

When a user clicks the question mark button in the toolbar, OWF offers online help:

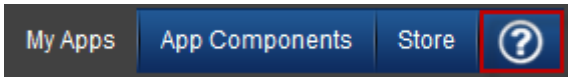


Figure 1: Help Button on Toolbar
Out of the bundle, the Help window contains:

- Instructions for Configuring Help
- Keyboard Navigation Shortcuts

4.1.2.1 Changing the Location of Help

The help directory location is defined by the **helpPath** property in **application.yml**. By default help files are located on the classpath. To change the directory location, replace **classpath:help** with a location in one of the following supported formats, then restart the server:

```
// 'file:/some/absolute/path' ('file:C:/some/absolute/path' on Windows)
// 'classpath:location/under/classpath'
// 'location/within/OWF/war/file'
```

4.2 Custom Configuration

OWF externalized configuration files are employed in **application.yml**. When OWF is deployed to a non-localhost environment, all externalized configuration files must be deployed and modified. In order to deploy to a non-localhost environment, changes need to be made to each file. Those changes are explained in the individual sections about each file.
Use of a production quality database (such as Oracle or MySQL), instead of the default HSQLDB, will require a change to the **application.yml** file, detailed in the following section.
*Note: In previous versions of OWF, the **OwfConfig.xml** file housed many of the application's customizable values. Beginning with OWF 7.17.2, these values have been moved to the **application.yml** file described below.*

4.2.1 Application.yml File

Application.yml is an OWF configuration file that allows an administrator to modify database connectivity information and other OWF variables. Once changes are made, restart the system to apply the changes. Developers comfortable with the Groovy language and the Grails application framework should be comfortable writing additional code for this file.
For full descriptions on database variables, please see Table 4: OWF Externalized Database Properties.

```
Environments:
    Production:
        dataSource:
            dbCreate: "none"
            username: "sa"
            password: ""
```

```
            driverClassName: "org.hsqldb.jdbcDriver"
            url: "jdbc:hsqldb:file:prodDb;shutdown=true"
            pooled: true
            properties:
                minEvictableIdleTimeMillis: 180000
                timeBetweenEvictionRunsMillis: 180000
                numTestsPerEvictionRun: 3
                testOnBorrow: true
                testWhileIdle: true
                testOnReturn: true
                validationQuery: "SELECT 1 FROM
INFORMATION_SCHEMA.SYSTEM_USERS"
        //enable uiperformance plugin which bundles and compresses
javascript
        uiperformance.enabled: true

        notifications:
            enabled: false
            query.interval: 30

            xmpp:
                username: ''
                password: ''
                host: ''
                room: ''
                port: 5222

//this section may modify any existing spring beans
Beans:

//main owf config object
Owf:


    // log4j file watch interval in milliseconds
    log4jWatchTime: 180000; // 3 minutes

    sendWidgetLoadTimesToServer: true
    publishWidgetLoadTimes: true

    defaultTheme: "a_default"

    showAccessAlert: "true"
    accessAlertMsg: "Lorem ipsum dolor sit amet, consectetur adipiscing
elit. Nulla interdum eleifend sapien dignissim malesuada. Sed imperdiet
augue vitae justo feugiat eget porta est blandit. Proin ipsum ipsum,
rutrum ac gravida in, ullamcorper a augue. Sed at scelerisque augue.
Morbi scelerisque gravida sapien ut feugiat. Donec dictum, nisl commodo
dapibus pellentesque, enim quam consectetur quam, at dictum dui augue at
risus. Ut id nunc in justo molestie semper. Curabitur magna velit, varius
eu porttitor et, tempor pulvinar nulla. Nam at tellus nec felis tincidunt
fringilla. Nunc nisi sem, egestas ut consequat eget, luctus et nisi.
Nulla et lorem odio, vitae pretium ipsum. Integer tellus libero, molestie
a feugiat a, imperdiet sit amet metus. Aenean auctor fringilla eros, sit
amet suscipit felis eleifend a."

    //use to specify a logout url
    logoutURL: "/logout"

    //sets the autoSave interval for saving dashboards in milliseconds
900000 is 15 minutes
    autoSaveInterval: 900000

    helpFileRegex: '^.*\\.(htm|html|gsp|jsp|pdf|doc|docx|mov|mp4|wmv)$'

    //this value controls whether the OWF UI uses shims on floating
```

```
elements, setting this to true will make
    //Applet/Flex have less zindex issues, but browser performance may
suffer due to the additional shim frames being created
    useShims: false

    //Locations for the optional external themes and help directories.
    //Default: 'themes', 'help', and 'js-plugins' directories on the
classpath.
    //Can be configured to an arbitrary file path.  The following
    //path styles are supported:
    //  'file:/some/absolute/path' ('file:C:/some/absolute/path' on
Windows)
    //  'classpath:location/under/classpath'
    //  'location/within/OWF/war/file'
    External:
        themePath: 'classpath:themes'
        helpPath: 'classpath:help'
        jsPluginPath: 'classpath:js-plugins'

    metric:
        enabled: false
        url: 'https://localhost:8443/metric/metric'

        //Optional additional properties with default values shown
        //keystorePath = System.properties['javax.net.ssl.keyStore']
        //keystorePass =
System.properties['javax.net.ssl.keyStorePassword']
        //truststorePath = System.properties['javax.net.ssl.trustStore']
        //timeout = 1800000


    Dataguard:



// Option to restrict messages between widgets based on access levels.
        // If this option is set to false, all other dataguard options
are ignored.
        restrictMessages: false

        // Option to audit all messages between widgets, not just failed
messages.
        // restrictMessages must be set to true
        auditAllMessages: false

        // Option to allow widgets to send messages without specifying
their access level.
        // restrictMessages must be set to true
        allowMessagesWithoutAccessLevel: true

        // The amount of time (in milliseconds) to cache a widget's
access level.
        // restrictMessages must be set to true
        accessLevelCacheTimeout: 3600000

    // OZP-476: Marketplace (MP) Synchronization
    mpSync:
        // Process listing change notifications from Marketplace(s)
        Enabled: false

        // Change notification from MP will cause a new widget to be
created
        // in OWF if it does not already exist
        autoCreateWidget: false

// Variables used for CEF logging. These fields are combined to form the
```

```
CEF prefix.
// For more information about CEF logging, check ArcSight documentation
Cef:
    Device:
        // The device vendor or organization
        Vendor: "OZONE"
        // The device product name
        Product: "OWF"
        // The device version
        Version: "500-27_L2::1.3"
    // The CEF version
    Version: 0
```

The application.yml file contains customizable functionality which can be modified by an administrator. Restarting the system once the files have been modified will apply the changes.

4.2.1.1 Notifications

The OWF Notifications system connects an OWF and Store to an XMPP server through which it can receive information about app components. This feature was originally designed to be used by administrators to monitor the changelogs' of Store listings, but may be used to allow any app components to communicate with OWF users. App components that connect to this XMPP server can use it to publish notification messages for consumption by OWF users. Messages published to the chat room must follow a specific format that designates both the message content and the OWF usernames of the intended recipient users. The OWF server will then receive the messages from the chat room and distribute them to individual users.
To configure OWF to use notifications, follow these steps:

1. Create a chat room on an XMPP server that serves as a repository for alerts to/from OWF and the Store, respectively.
2. Create user accounts on the XMPP server for the Store and OWF servers.
3. Use the XMPP server information to populate the XMPP Settings in the Notifications section of **application.yml** defined below: *Note: Use the user accounts and their passwords from step two as the Username and Password fields on the Notifications configurations in OWF and the Store.*

- **Enabled** – Turn on or turn off notifications.
- **Query.interval** – How often OWF fetches new notifications.
- **XMPP Settings**
    - **username** – The domain name for the XMPP server that administers notifications.
    - **password** – The domain password for the XMPP server that administers notifications.
    - **host** – The hostname of the XMPP server.
    - **room** –The XMPP chat room that receives notifications.
    - **port** –The TCP port the XMPP server uses.

```
Notifications:
 enabled: false
 query.interval: 30
```

```
    xmpp:
      username: ''
    password: ''
    host: ''
    room: ''
    port: 5222
```

4.2.1.2 Custom log4jWatchTime

The values associated with the **log4j WatchTime** offers a time-based value as to how often the **log4j** logging system looks for updates in the **owf-override-log4j.xml** file. The 180000 value is stored in milliseconds. Accordingly, the value of 180,000 as presented below is actually a 3-minute timer.

```
log4j {
// log4j file watch interval in milliseconds
log4jWatchTime = 180000; // 3 minutes
}
```

*Note: When changes are made to the **log4jwatchTime** property, the server does NOT need to be restarted.*

4.2.1.3 Custom Data Guard and Widget Security

OWF 7.1 enhanced widget security with a data guard system that addresses widget security in the following ways:

- Restricts messages between widgets based on their access levels. Disabled by default, to activate the feature set **restrictMessages** to **true**.
- Audits all messages between widgets. To enable this feature, ensure that **restrictMessages** is set to **true**, then set the **auditAllMessages** to **true**.
- Restricts widgets ability to send messages until their access level is specified. To enable this feature, ensure that **restrictMessages** is set to **true**, then change **allowMessagesWithoutAccessLevel**to **false**.
- Specifies the amount of time the system allows to cache a widget's access level. To modify the duration, ensure that **restrictMessages** is set to **true** and quantify the **accessLevelCacheTimeout** in milliseconds.

To change this format, update the respective properties in **application.yml**:

```
Dataguard:
  // Option to restrict messages between widgets based on access levels.
  // If this option is set to false, all other dataguard options are
ignored.
  restrictMessages: false

  // Option to audit all messages between widgets, not just failed
messages.
  // restrictMessages must be set to true
  auditAllMessages: false

  // Option to allow widgets to send messages without specifying their
access level.
  // restrictMessages must be set to true
  allowMessagesWithoutAccessLevel: true

  // The amount of time (in milliseconds) to cache a widget's access
level.
```

```
// restrictMessages must be set to true
accessLevelCacheTimeout: 3600000
```

4.2.1.4 Store Synchronization

The synchronization feature allows **the Store** to automatically send app components and their subsequent updates to OWF. Synchronization is also necessary to push OWF applications to the Store. To use this feature, developers must do the following:

- Configure **application.yml** to accept synchronization messages from the Store.
- Configure OWFsecurityContext.xml and MPSecurityContext.xml (see below).
- Synchronize the OWF server with the Store through the Store's Administration Configuration pages.

To implement the synchronization feature with the OWF sample security plugin, configure the following OWF and Store files (found in the ***apache-tomcat-7.0.21*** □ ***lib*** directory):
In OWF:

- **OWFsecurityContext.xml** must include the following in the top-level ***<beans>*** element:

<sec:http pattern="/marketplace/sync/" security="none" />
In the Store:

- **MPSecurityContext.xml** must include the following in the top-level ***<beans>*** element:

<sec:http pattern="/public/descriptor/" security="none" />
The Store will only synchronize with OWF servers linked to that Store. The Store will not synchronize with unspecified OWF systems or offline OWF systems. Synchronize OWF servers with the Store through the Store Administration Configuration pages. For more information, see the OZONE Store Administrator's Guide.

4.2.1.4.1 Automatically Add the Store Listings and Updates

The Marketplace Synchronization feature allows OWF administrators to automatically receive app components and their updates from the Store. The synchronization feature is disabled by default. When enabled, OWF compatible listings are directly added from the Store to the App Component Manager in OWF.
To enable this feature, edit the following property values in **application.yml** as indicated in the table.

Table 6: Explanation of Synchronization Properties

Property Name	Description	Value
autoCreateWidget	Denotes whether new Store listings are automatically added to OWF.	true
enabled	Denotes whether OWF will process listing requests from the Store. Set this to true to receive listings and updates from the Store.	true

Note: Updates from the Store will overwrite any changes made to the corresponding widgets in OWF.
When **autoCreateWidget = true**, OWF automatically adds app components to the App Component Manager that have the following listing criteria in the Store:

- Type is App Component
- Listing status is Approved
- Listing is Enabled
- A State that has "Is Published" set to true

Initially, the added app components are not associated with users or groups. However, users or administrators can see and add them from the Store. Administrators will not automatically receive all OWF compatible app components when they sign into OWF. Only new and updated

Store listings that meet the criteria described above will automatically appear in the App Component Manager after enabling the this feature.

4.2.1.4.2 Enabling Auto-Updates from the Store

Using synchronization, users and administrators can automatically receive updates for app components previously added to OWF through the Store app component. Store updates will overwrite changes made to the corresponding app component in OWF. For example, if an administrator adds an Intent to the app component, the update from the Store will remove this data. To allow Store synchronization in OWF, set the **enabled** property to **true** in the **application.yml** file.

To disable the synchronization between the Store listings and their OWF counterpart widgets, set the **enabled** property to **false**; by default, this property is set to **false**. Alternatively, disable the listing in the Store or remove the OWF connection from the Store.

If a developer chooses not to enable the automatic widget synchronization, updating a widget will require the user to delete their existing version of the widget and add the updated widget from the Store Widget in their OWF.

4.2.1.5 App Component Load Times

Because the speed with which app components launch and operate is related to an individual system's resources, OWF has implemented ways to track the time between a component's rendering and the time it is actually ready to be used. The following values can be active at the same time:

```
sendWidgetLoadTimeToServer = true
publishWidgetLoadTimes = true
```

sendWidgetLoadTimeToServer sends app component load time data to a system log file where it is written and stored.

publishWidgetLoadTimes will send the data to the Widget Log App Component, which can be opened from the Favorites Menu, after its assigned to a user's instance of OWF.

4.2.1.6 Using Shims

From the **useShims** property in **application.yml**, developers can configure OWF to use iframe shims. The property is set to **false** by default. Shims can be used to improve the stability of floating element features. To use shims, set the **useShims** property to **true** as shown in the following code example:

```
//this value controls whether the OWF UI uses shims on floating elements,
// setting this to true will make
//Applet/Flex have less zindex issues, but browser performance may suffer
// due to the additional shim frames being created
useShims = true
```

4.2.1.7 Custom Show Access Alert

Depending on the individual security requirements where OWF is being deployed, users may be required to agree to the specific terms of a security warning. Deploying a security warning is accomplished via a custom access alert. There are two sets of properties (and their values) which govern the access alert message:

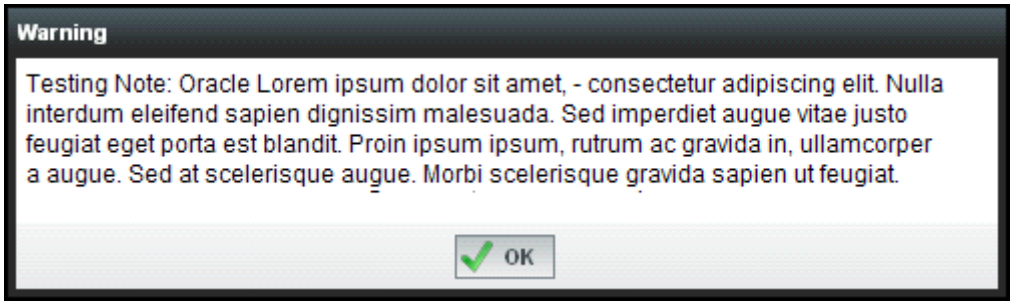


Figure 2: Customizable Warning Message

OWF ships with placeholder text, that is in the following configuration properties:

```
showAccessAlert = "true"
accessAlertMsg = "Lorem ipsum dolor sit amet, consectetur adipiscing
elit. Nulla interdum eleifend sapien dignissim malesuada. Sed imperdiet
augue vitae justo feugiat eget porta est blandit. Proin ipsum ipsum,
rutrum ac gravida in, ullamcorper a augue. Sed at scelerisque augue.
Morbi scelerisque gravida sapien ut feugiat. Donec dictum, nisl commodo
dapibus pellentesque, enim quam consectetur quam, at dictum dui augue at
risus. Ut id nunc in justo molestie semper. Curabitur magna velit, varius
eu porttitor et, tempor pulvinar nulla. Nam at tellus nec felis tincidunt
fringilla. Nunc nisi sem, egestas ut consequat eget, luctus et nisi.
Nulla et lorem odio, vitae pretium ipsum. Integer tellus libero, molestie
a feugiat a, imperdiet sit amet metus. Aenean auctor fringilla eros, sit
amet suscipit felis eleifend a."
```

1. The value for the property **showAccessAlert** must be true or false. These values are case sensitive and will only work as **true** or **false** (all lowercase) values. True will display the warning message, as created in the value tag for the property name **accessAlertMsg**. The value can be made false, which will stop the warning from triggering. This allows an administrator to keep a warning message saved, regardless of whether it is being used or not.
2. When the **showAccessAlert** value is set to true, upon sign in, users will be presented with the new access alert message and will not be granted access to OWF until they click the OK button.

4.2.1.8 Custom Logout

OWF allows users to sign out of OWF in accordance with their instance's security plugins. Any modifications made to the URL will take effect when the system is restarted.

```
//use to specify a logout url
logoutURL: "/logout"
```

Note: The security plugin being used must be customized so that the value entered above will work as expected.

4.2.1.9 Custom "Auto Save" Interval

The OWF user interface will automatically save the user's dashboard at a configurable time interval. The default is to save every 15 minutes. Any modifications to the default save interval will take effect when the **application.yml** is saved and the system is restarted.

```
//sets the autoSave interval for saving dashboards in milliseconds 900000
is 15 minutes
autoSaveInterval: 900000
```

Note: The value for the above setting is in milliseconds. It can be changed to any millisecond value.
Note: The Auto Save will keep the user session alive as long as a dashboard is visible in the browser.

4.2.1.10 Custom Help File Types

Only files in the **\help** directory with specific file extensions will appear in the user interface. By default, files with the following file extensions will appear: **HTM**, **HTML**, **GSP**, **JSP**, **PDF**, **DOC**, **DOCX**, **MOV**, **MP4**, and **WMV**. To modify the file types that will appear in the Help window on the OWF Toolbar, the administrator must restart the server after updating the following list of values in **application.yml**:

```
helpFileRegex = '^.*.(htm|html|gsp|jsp|pdf|doc|docx|mov|mp4|wmv)$'
```

4.2.1.11 Enabling Ozone Metrics Service

To enable communication with the Metrics Service change the **metric.enabled** property to true. If the metrics service is located on a different server than OWF, enter its URL as shown in the following example:

```
metric:
  enabled: true
  url: 'https://servername:port/metric/metric'
```

*Note: Find detailed instructions regarding Metrics Service security, configuration and installation (including database information) in the Metrics Service Guide which is located in the **metric-bundle.zip**.*

4.2.1.11.1 Configuring the Metrics App Component in OWF

To view metrics data using an OWF app component, make metrics data available in OWF:

1. Integrate the metrics server into OWF, as described in the previous section.
2. Click My Apps in the upper-left corner.
3. From the My Apps window, click Administration, then click App Components to open the App Component Manager. *Note: If the Administration Application does not exist, you can access Administrative functions from the drop-down User Menu. Click Administration, then select App Components.*
4. Use the following information to create the View Metrics App Component:

Table 7: Data for Metrics App Component Definition

Definition	Data Input
URL	https://widget-servername:port/metric/admin/Metrics.gsp
Large Icon	https:// widget-servername:port/metric/themes/common/images/icons/16x16_metrics.png
Small Icon	https:// widget-servername:port/metric/themes/common/images/icons/64x64_metrics.png

Definition	Data Input
Width	700
Height	500
Widget Type	Metric

4.2.1.11.2 Viewing the Metrics App Component in OWF

The Metrics Service can only be viewed in OWF. After following the steps in the previous section and assigning the View Metrics component to users, the component will appear under the Metrics button on the toolbar.

Note: OWF users who signed in using CAS will be prompted to sign in to additional security for the Metrics Service.

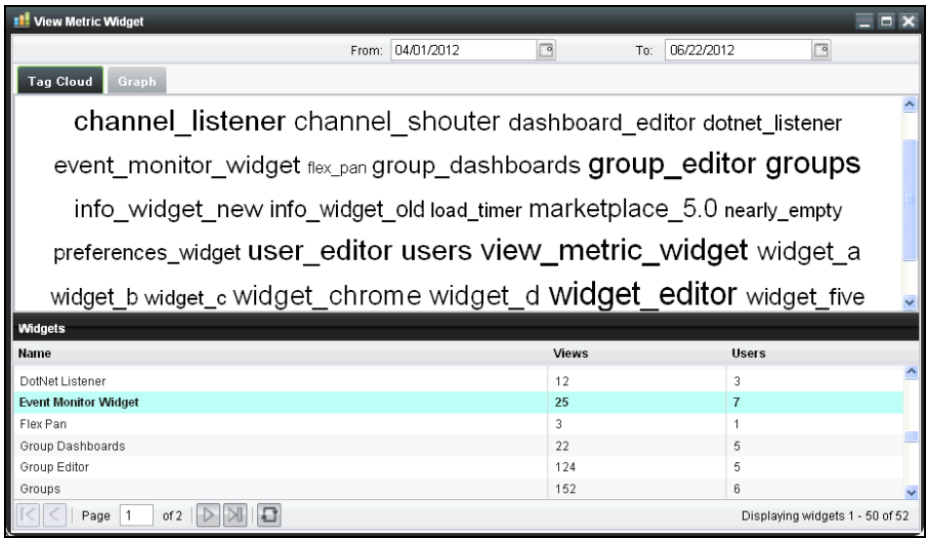


Figure 3: Metric Widget

The app component opens to a Tag Cloud tab that lists App Component views (components that are viewed more will be larger in the Tag Cloud) and a grid of app component data. Clicking the name of a widget from the Tag Cloud or the grid switches to the widget's Graph tab. This displays how many times that widget was viewed. To change the monitoring dates, click the calendar icon(s) above the graph. Use the arrows or click directly on the month and year to select different date durations. Clicking the month and year (identified in the image below) opens a drop-down date selector. After clicking OK, click the highlighted date on the calendar to complete the change.

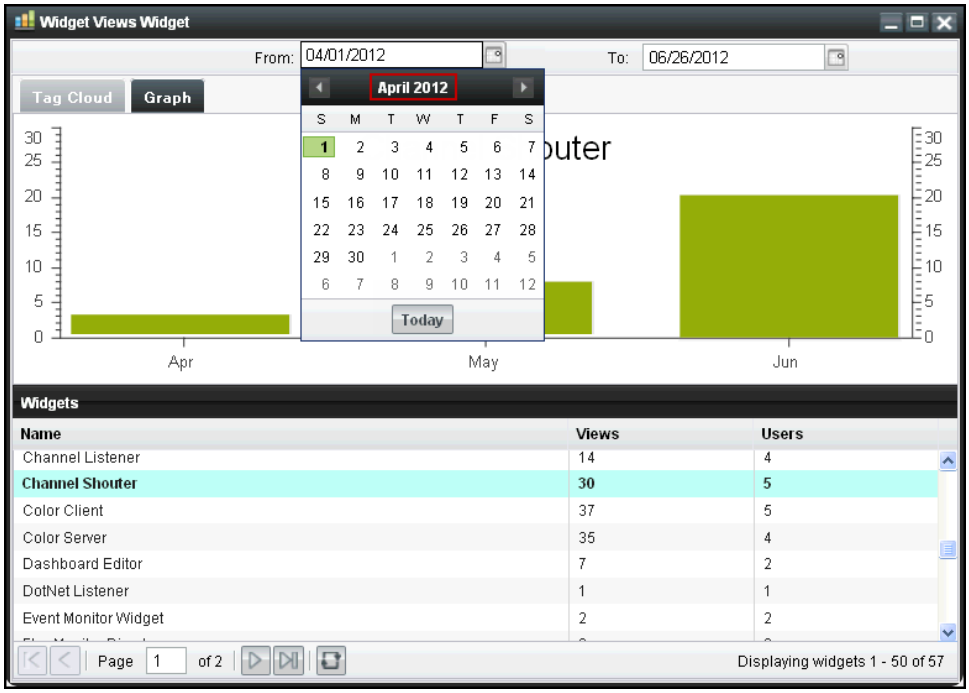


Figure 4: Graph Tab Displaying Date Switcher

4.2.2 Server Settings

All references to OWF must match the settings of the current installation. Based on the settings in **OzoneConfig.properties**, the variables (e.g. **${ozone.host}**) are filled in at runtime. Also, **OzoneConfig.properties** includes optional CAS configurations for OWF, the Store and the Metrics Service.

Table 8: OWFCasBeans.xml Server Settings

Property	Purpose	Example
casProcessingFilterEntryPoint.loginUrl	Must point to the CAS login page.	https://${ozone.host}:${ozone.port}/${ozone.cas.serverLoginLocation}(e.g.https://servername:port/cas/login)
serviceProperties.Service	Must point to the OWF web server.	https://${ozone.host}:${ozone.port}/${ozone.cas.owf.jSpringCasSecurityCheckLocation}(e.g.https://servername:port/owf/j_spring_cas_security_check)

Property	Purpose	Example
ticketValidatorFactory.casServiceUrl	Must point to the CAS server.	https://${ozone.host}:${ozone.port}/${ozone.cas.serverName}(e.g.https://servername:port/cas)
ticketValidatorFactory.proxyCallbackUrl	Must point to the OWF web server.	https://${ozone.host}:${ozone.port}/${ozone.cas.owf.serverSecureReceptorLocation}(e.g.https://servername:port/prefs/secure/receptor)

Table 9: OWFLogInOutBeans.xml Server Setting

Property	Purpose	Example
OzoneLogoutSuccessHandler/constructor-arg/index=1	Must point to the CAS logout page.	https://${ozone.host}:${ozone.port}/${ozone.cas.serverLogoutLocation} (e.g.https://servername:port/cas/logout)

4.2.3 Setting up the CAS server

As of version 7.5.0 of the OZONE Store, the sample security plugin no longer uses CAS by default, and the CAS web application is no longer included in the bundle. In order to use a security plugin configuration that does use CAS, a CAS server must first be downloaded, installed, and configured. For CAS downloads and detailed instructions, see the CAS web site here: *http://www.jasig.org/cas*

4.2.4 JVM Memory Settings

Adjusting a server's memory settings can increase performance or resolve permgen OutOfMemory errors. To adjust memory settings:

1. In the Tomcat start script (**tomcat\lib\start.sh** or **start.bat**) set the initial **permgen** size to at least 256 MB. This can be accomplished by adding **-XX:PermSize=256m** to the Java options. If more server memory is available, increasing this **permgen** size may increase performance.
2. Set the maximum **permgen** size to at least 384 MB. This can be accomplished by adding **-XX:MaxPermSize=384m** to the Java options. If you have more memory available on your server increasing this **permgen** size.

4.3 Application Logging

4.3.1 Custom Logging Functions

General logging can be enabled by editing the **owf-override-log4j.xml** file which can be found in the **tomcat\lib** directory:

```
<logger name="AuditOWFWebRequestsLogger" additivity="false">
 <level value="error" />
 <appender-ref ref="ozone-async" />
</logger>


<!-- For security logging, set this log level to "info". -->
<logger
name="ozone.securitysample.authentication.audit.SecurityAuditLogger"
additivity="false">
 <level value="info" />
 <appender-ref ref="ozone-async-audit" />
</logger>


<!—Add this to enable general OWF Debug logging -->
<logger name="grails.app" additivity="false">
 <level value="trace" />
 <appender-ref ref="ozone-async" />
</logger>
```

*Note: The **owf-override-log4j.xml** file shown above does not ship with the code shown at the bottom of the sample. However, it can be pasted into the file at an administrator's discretion in order to enable the logging of general OWF server debug messages.*

To confirm that the log files are being written, examine the **tomcat\logs** directory. Developers familiar with **Log4j** configurations should be comfortable with this file.

*Note: Useful configurations/common requests are called out in comments in the file. For example, audit logging describing each user's Web calls can be enabled by setting **AuditOWFWebRequesterLogger**and **ozone.filter** to logging level **info**.*

Different third party libraries within OWF have also been called out so that administrators can easily modify logging levels.

4.3.2 Audit Logging

OWF includes an option to audit all user entry and exit in the system. The OWF Bundle ships with this feature enabled by default. The Audit Log tracks the following types of changes:

- Both successful and unsuccessful sign-in attempts
- User Sign-out Events:
    - A user signing out on purpose
    - A session times out

Note: References to CAS and OWF must match the settings of the current installation.

4.3.2.1 Sign-in Events

A sign-in failure will occur and be recorded in the log if a user has a valid PKI Certificate but the associated username is not registered as a valid user within OWF. A failed sign in produces the following log statement at the info level:

```
INFO [02/15/2011 15:24:04 -0500] IP: 127.0.0.1 User: testAdmin1 [USER
LOGIN]: LOGIN FAILURE - ACCESS DENIED with FAILURE MSG [Login for
'testAdmin1' attempted with authenticated credentials [CERTIFICATE
LOGIN]; However, the Provider was not found. Access is DENIED.]
```

A failed sign in produces the following log statement at the debug level:

```
DEBUG [02/15/2011 15:27:18 -0500] IP: 127.0.0.1 User: testAdmin1 [USER
LOGIN]:LOGIN FAILURE - ACCESS DENIED with FAILURE MSG [Login for
'testAdmin1' attempted with authenticated credentials [CERTIFICATE LOGIN
>> Signature Algorithm: [SHA1withRSA, OID = 1.2.840.113549.1.1.5];
Subject: [EMAILADDRESS=testAdmin1@nowhere.com, CN=testAdmin1, OU=Ozone,
O=Ozone, L=Columbia, ST=Maryland, C=US]; Validity: [From: Thu Feb 04
13:58:52 EST 2010, To: Sun Feb 03 13:58:52 EST 2013]; Issuer:
[EMAILADDRESS=ozone@nowhere.com, CN=localhost, OU=Ozone, O=Ozone,
L=Columbia, ST=Maryland, C=US]; ]; However, the Provider was not found.
Access is DENIED. Login Exception Message: [No AuthenticationProvider
found for
org.springframework.security.web.authentication.preauth.PreAuthenticatedA
uthenticationToken]]
```

A successful PKI Certificate sign in produces the following log statement at the info level:

```
INFO [02/15/2011 15:39:13 -0500] IP: 127.0.0.1 User: testAdmin1 [USER
LOGIN]: LOGIN SUCCESS - ACCESS GRANTED USER [testAdmin1], with DISPLAY
NAME [Test Admin 1], with AUTHORITIES [ROLE_ADMIN,ROLE_USER], with
ORGANIZATION [Test Admin Organization], with EMAIL
[testAdmin1@nowhere.com]with CREDENTIALS [CERTIFICATE LOGIN]
```

A successful PKI Certificate sign-in statement produces the following log statement at the debug level:

```
DEBUG [02/15/2011 15:42:10 -0500] IP: 127.0.0.1 User: testAdmin1 [USER
LOGIN]:LOGIN SUCCESS - ACCESS GRANTED USER [testAdmin1], with DISPLAY
NAME [Test Admin 1], with AUTHORITIES [ROLE_ADMIN,ROLE_USER], with
ORGANIZATION [Test Admin Organization], with EMAIL
[testAdmin1@nowhere.com] with CREDENTIALS [CERTIFICATE LOGIN >> Signature
Algorithm: [SHA1withRSA, OID = 1.2.840.113549.1.1.5]; Subject:
[EMAILADDRESS=testAdmin1@nowhere.com, CN=testAdmin1, OU=Ozone, O=Ozone,
L=Columbia, ST=Maryland, C=US]; Validity: [From: Thu Feb 04 13:58:52 EST
2010, To: Sun Feb 03 13:58:52 EST 2013]; Issuer:
[EMAILADDRESS=ozone@nowhere.com, CN=localhost, OU=Ozone, O=Ozone,
L=Columbia, ST=Maryland, C=US]; ]
```

Logout Events

Sign-out events are logged by the **ozone.securitysample.authentication.audit.SecurityAuditLogger** logger. This logger supports two levels of logging: info and debug, with the latter providing more detailed information about each sign-out event.

Below is a typical user-initiated sign-out event which has been saved as a log entry, with the log level set to info:

```
INFO [02/03/2011 16:13:35 -0500] IP: 127.0.0.1 SessionID: 8ki2ttimdxc
User: testAdmin1 [USER LOGOUT]:
```

Below is a typical user-initiated sign-out event which has been saved as a log entry, with the log level set to debug:

```
DEBUG [02/03/2011 15:59:53 -0500] IP: 127.0.0.1 SessionID: 1tjefhsxz1x6t
User: testUser1 [USER SESSION TIMEOUT] with ID [2], with EMAIL
[testUser1@nowhere.com], with ACCOUNT CREATED DATE [02/03/2011 15:58:50 -
0500], with LAST LOGIN DATE [02/03/2011 15:58:50 -0500]
```

A user can also be forced to sign-out when their session times out. Below are info and debug log statements:

```
INFO [02/07/2011 10:08:21 -0500] IP: 127.0.0.1 SessionID: 1b4nvaqnb0qx8
User: testAdmin1 [USER SESSION TIMEOUT]
DEBUG [02/07/2011 10:24:21 -0500] IP: 127.0.0.1 SessionID: d0pq3g4xguv3
User: testAdmin1 [USER SESSION TIMEOUT] with ID [1], with EMAIL
[testAdmin1@nowhere.com], with ACCOUNT CREATED DATE [02/07/2011 10:23:18
-0500], with LAST LOGIN DATE [02/07/2011 10:23:18 -0500]
```

4.3.3 Common Event Format (CEF) Auditing

Common Event Format (CEF) auditing capabilities are available in OWF and the Store. The variables used for OWF CEF logging are defined in the **application.yml**. To enable/disable them, sign into OWF and the Store as an administrator and navigate to the auditing configurations.
CEF auditing is turned ON by default, the toggle controls for both CEF and Object Access auditing are found in OWF's Application Configurations which is located on the drop-down User Menu in the user interface. For more information, see the OWF Administrator's Guide.
When enabled, CEF auditing records common user events:

- Sign in and out (Sign out - Marketplace only)
- Create, Read*, Edit and Delete
- Search
- Import and Export

*Note: * Object Access auditing, is a separate CEF auditing feature that records users' Read events. Read events are logged when **both** the CEF auditing global flag and the Object Access flag are ON in OWF's Application Configurations. If the Object Access auditing is ON and the CEF auditing is OFF, no Read events are logged with CEF auditing.*
CEF auditing and Audit Logging (see section 4.3.2: Audit Logging) record overlapping database events (ex. edit, delete and log out events). When CEF auditing is ON, CEF auditing is the recorder of these database events; these events are not recorded with Audit Logging. When CEF is turned OFF, Audit Logging records these database events.
The following are two log examples using CEF auditing.

CEF auditing from an object modification event:

```
26 Jun 2013 12:31:37,217 EDT CEF:0|ORG|MARKETPLACE|500-
27_L2::IC::1.3|FILEOBJ_MODIFY|Object was updated|7|cat=FILEOBJ_MODIFY
suid=MikePAdmin shost=10.10.16.12 requestMethod=USER_INITIATED
outcome=SUCCESS deviceFacility=0CCB5827C819E1A4AC9BE5BD4C6F9FE9.mp02
reason=UNKNOWN cs5=UNKNOWN act=7.2 deviceExternalId=UNKNOWN
dhost=amlqa02.goss.owfgoss.org cs4=UNKNOWN start=06:26:2013 12:31:37
[.037]cs3=UNKNOWN fname=[CLASS:marketplace.OwfProperties, stackContext:,
stackDescriptor:] filePermission=UNKNOWN fileId=16 fsize=2
fileType=OBJECT oldFilename=[CLASS:marketplace.OwfProperties,
stackContext:null, stackDescriptor:null]oldFilePermission=UNKNOWN
oldFileId=16 oldFileSizesize=2 oldFileType=OBJECT
```

CEF auditing from a log on event:

```
26 Jun 2013 08:57:51,974 EDT CEF:0|ORG| MARKETPLACE L|500-
27_L2::IC::1.3|LOGON|A logon event occured.|7|cat=LOGON suid=MikePAdmin
shost=10.10.16.12 requestMethod=USER_INITIATED outcome=SUCCESS
deviceFacility=8C24A08B7E9848C80F929791DA40F734.mp02 reason=UNKNOWN
cs5=UNKNOWN act=7.2 deviceExternalId=UNKNOWN
dhost=amlqa02.goss.owfgoss.org cs4=UNKNOWN start=06:26:2013 08:57:51
[.051]cs3=UNKNOWN
```

5 OWF Security

OWF allows an administrator to customize the type of security that is implemented for authentication and authorization. OWF uses a pluggable Spring Security 3.0.2 solution and ships with sample security plugins that can be used as a basis for building a custom security plugin. Familiarity with Spring Security will help administrators customize OWF.

5.1 Basic Security Concepts and OWF

While this guide is not intended as a comprehensive guide to basic security concepts, Web security, or Spring Security, there are a few key concepts that must be understood in order to use the sample OWF security plugins and the OWF security plugin architecture.
First are the concepts of authentication and authorization, known colloquially as auth & auth. Authentication essentially means providing proof that the user is exactly who they are presenting themselves to be. Some authentication techniques include a username/password combination, an X509 certificate, a CAC card and card reader, or various biometric solutions. Authorization, on the other hand, is determining the specific access rights that an individual user should have. Consider the following:

- "Bill is allowed to log into the system – prove that you are Bill," is a matter of authentication.
- "Bill has access to resources," is a question of authorization.

By necessity, authentication occurs before authorization. Once authentication is satisfied, OWF moves to authorize. OWF has two authorization concepts at this time. First, OWF needs to know whether or not a user has OWF administrative access via ROLE_ADMIN or is only a regular user, via ROLE_USER. Administrative access provides a user access to the administrative widgets and the administrative console. Regular users have access only to the framework and their assigned dashboards.
Second, OWF needs to know what external OWF user groups (if any) the user has been assigned. There are two kinds of user groups; automatic user groups, which are pulled in from an external authorization source, such as LDAP or a configuration file, and manual user groups, which are set up from within OWF. If an automatic user group is new to OWF, all of the automatic user groups' details such as description, active/inactive status, contact email address, and name come from the external source. But after the initial creation of the group in OWF, no further updates to the description, status, etc. are made.

5.2 Requirements for Customizing Security

The Spring Security Framework allows individual deployments to customize the OWF authentication and authorization mechanism. Developers can use the security plugin to integrate with any available enterprise security solutions. When customizing the security plugin, it is important to remember OWF/Store requirements for the plugin. These five requirements are described in this section.
Note: The OWF/Store requirements are in addition to any general Web application requirements relating to Spring Security.

1. User principal implements the **UserDetails** interface and (optionally) the **OWFUswerDetails** interface.

Like all Spring Security Web applications, OWF expects its security plugin to provide a **UserDetails** object which represents the logged-in user. A custom plugin should set this object as the principal on the Authentication object stored within the active **SecurityContext**. Optionally, the provided object may implement the **OWFUserDetails** interface. In addition to the fields supported by the **UserDetails** interface, the **OWFUserDetails** interface supports access to the user's OWF display name, organization and email. The source code for **OWFUserDetails** can be found in **ozone-security-project.zip**.

2. **ROLE_USER** granted to all users

The user principal object's **getAuthorities()** method must return a collection that includes the **ROLE_USER GrantedAuthority**.

3. **ROLE_ADMIN** granted to OWF administrators

The user principal object's **getAuthorities()** method must return a collection that includes the **ROLE_ADMIN GrantedAuthority** if the user is to have administrative access.

4. **OZONELOGIN** cookie set when the user signs in and deleted on sign out

The user interface performs a check for the existence of a cookie named **OZONELOGIN** during the page load. If the cookie does not exist, the interface will not load, but will instead present a message indicating that the user is not signed in. It is up to the security plugin to create this cookie when the user signs in, and to delete it when they sign out. This mechanism prevents users from signing out, and then pressing the browser's Back button to get back into an OWF instance that cannot communicate with the server due to failed authentication. The sample security plug-in configurations contain filters that manage this process. It is recommended that custom configurations include this default implementation of the cookie behavior by using the same **ozoneCookieFilter** and **OzoneLogoutCookieHandler**beans that are included in the sample configuration, in **OWFsecurityContext.xml** and **OWFLogInOutBeans.xml**.

5. Session management configurations must be present. These configurations include the **concurrencyFilter** bean, the **concurrentSessionControlStrategy** bean, the session **Registry** bean as well as a **<session-management>** element and a **<custom-filter>** element which references the **concurrencyFilter**. For examples of the required settings for these elements, see **OWFSecurityContext.xml** and **ozone-security-beans\SessionManagementBeans.xml**. It is important not to change the **id** of the **concurrentSessionControlStrategy**, as it is referenced by **id** from within the application. *Note: The maximumSessions setting contained in the xml configuration will be overwritten at runtime, since the maximum number of sessions is configured in the Application Configuration UI.*

5.3 Custom Security Logout

The OWF sample security plugins can perform single sign out if the user signed in using CAS authentication. PKI authentication is handled by the browser and requires that the user close the browser to completely sign out, though their session with the application can be reset. To sign out from a custom authentication, the system administrator must implement their own single sign out or instruct the user to close the browser after signout – see section 5: OWF Security Use the following lines in the OWF Security Context file to invoke CAS's single sign out process.

```
<sec:custom-filter ref="casSingleSignOutFilter" after="LOGOUT_FILTER"/>
…
<!—OWFCasBeans.xml contains the casSingleSignOutFilter bean definition --
>
<import resource="ozone-security-beans/OWFCasBeans.xml" />
```

5.4 Production Deployments

The samples included with OWF are not production quality samples. They are intended to provide examples on how to easily integrate various security solutions with OWF, not to provide a comprehensive security solution out of the box or a comprehensive tutorial on Spring Security. It is expected that each organization using OWF will examine its security guidelines and enterprise-wide authentication/authorization solutions and produce an OWF security plugin that is both secure and meets its standards. That solution can then be shared among OWF deployments within the organization.

Most of the examples provided contain various obvious security hazards—for example, the CAS-only, X509-only, and CAS + X509 plugins all contain a list of usernames, roles, and user groups on the hard drive in plain text in a properties file. **The CAS-only and CAS+X509 files contain the passwords in plain text. These are undeniable security hazards.** Keep this in mind when using the samples.

5.5 Installing the Security Module

The OZONE-security files offer multiple examples of security options. These are intended as examples and should in no way be used in a production environment. As mentioned previously, the default security implementation provides an X509 certificate authentication. When using the default security module in a testing environment, the user must present a valid X509 certificate, or a valid username and password, in order to gain access to OWF.
For each available security option, there is a specific **XML** file which must be installed. Installing a new security module is accomplished in just a few simple steps:
*Note: The following instructions act as a summary for installing individual security modules. Depending on the module being used or tested, module-specific instructions may be needed. See **\ozone-security\ozone-security-project.zip\readme.txt** for the installation details specific to each module type. Additionally the summary instructions below assume that the default installation is being used with Tomcat as the app server/container.*

1. Stop the application server. An administrator can accomplish this by clicking the **\tomcat\bin\shutdown.bat** or **\shutdown.sh** file, depending on the operating system in use.
2. Delete any security-based **XML** (**OWFsecurityContext**.xml**) files that might currently be present in the *\tomcat\lib** directory.
3. Copy the appropriate **XML** file from **\ozone-security** to the application server's class path. When running Tomcat, the classpath is the **\tomcat\lib** directory.
4. Restart the application server by clicking either **\tomcat\start.bat** or **\start.sh** file, depending on the operating system in use.

5.5.1 X509-Only Specific Instructions

The **OWFsecurityContext_cert_only.xml** file eliminates HTTP-BASIC as a fallback to authentication. If the user does not present a valid X509 certificate, they will be denied access to the system. Authorization is provided by the **users.properties** file. The format of this file is described in 4.1.1: Adding Users/Roles/Groups.
To use this security plugin, replace the active security-based **XML** file (e.g., **\tomcat\lib\OWFsecurityContext.xml**) with the **OWFsecurityContext_cert_only.xml** file, which can be found in the **\ozone-security** directory. Follow the directions to stop and restart OWF and follow the directions above, to remove CAS.

5.5.2 CAS-Only Specific Instructions

To use just the CAS server without any X509 certificate authentication, replace the provided **OWFsecurityContext.xml** file with **OWFsecurityContext_CAS_only.xml**, and follow the steps in 5.5: Installing the Security Module to stop and restart the server. When using the CAS only security implementation, if the user fails authentication to CAS, they will be denied access to the system.
When using the CAS only security implementation, the server in use may need to be adjusted to eliminate the prompt for a certificate. For example, the bundle's Tomcat instance is set up to ask for certificate authentication, but not require it. To eliminate the certificate prompt, edit the **\tomcat\conf\server.xml** file and change **clientAuth** property to false:

```
<Connector port="8443"
protocol="HTTP/1.1"
SSLEnabled="true"
maxThreads="150"
scheme="https"
secure="true"
keystoreFile="certs/keystore.jks"
keystorePass="changeit"
clientAuth="false"
sslProtocol="TLS" />
```

5.5.3 X509/LDAP

The **OWFsecurityContext_cert_ldap.xml** file provides X509 client authentication with an LDAP-based lookup to determine the user's authorization. The default configuration attempts to connect to a local installation of Apache Directory Server on port 10389, using the default system account. It determines the user's authorization by searching on the full distinguished name presented in the X509 certificate.
Sample configuration files are provided to set up an Apache Directory Server with user information that matches the X509 certificates provided with OWF, including a server configuration **.xml** file and an LDAP Data Interchange Format file **(**.ldif**) which loads users to match the distinguished names in the certificates. For more information about LDAP, refer to {+}http://directory.apache.org/+. Included is a sample Apache DS *server.xml** file, called **\ozone-security\ozone-security-project\src\main\resources\conf\apache-ds-server.xml**. It adds the partition owf-1 to Apache DS. To do so, it adds the following line of **XML** to the XPATH **spring:beans/defaultDirectoryService/partitions**:

```
<jdbmPartition id="owf-1" suffix="o=Ozone,l=Columbia,st=Maryland,c=US" />
```

It is also necessary to load the sample data into the directory service. The OWF team has provided a sample **LDIF** file, called **\ozone-security\ozone-security-project\src\main\resources\conf\testUsers.ldif**.
Note: Downloading the Apache Directory Studio may be helpful.
It is straightforward to modify how the LDAP search is conducted for both user roles and user groups. No adjustment is required in order to run the plugin with the default data. However, to modify the plugin to run off of a different data set, adjust **ozone-security-beans\LdapBeans.xml**. It is recommended that the administrator get the plugin working off of the default data set before trying to migrate to a different data set by modifying the LDAP queries.
To use the X509(cert)/LDAP security implementation, replace the provided **OWF-securityContext.xml** file with **OWFsecurityContext_cert_ldap.xml**, and follow the directions in section 5.5: Installing the Security Module to stop and restart the server and to remove CAS.

5.6 Custom Security and Sample Source Code

Along with the security-related **XML** files, there is also a **ZIP** file which contains the source and configuration files for the pluggable security modules. Additionally an Apache ANT build script is included. The build script allows for a quick and simple rebuild of the security project (**JAR**) files for greater customization, if needed.
There are two packages in the security project. The first package is **ozone.securitysample**. This package contains the code required for the security sample projects. This code is probably not extremely reusable, or reusable only with modifications. The second package is called

ozone.security. This package is very important, and is required by OWF to run.
The most important classes are defined in the following sections.

5.7 OWFUserDetails

Package:ozone.security.authentication
This interface defines interactions for a data model that OWF requires in order to handle OWF user groups. Using an implementation of this interface (and implementations may vary) will ensure that OWF user groups work.
This interface extends the classic **UserDetails** interface as defined by Spring Security 3.0. Please refer to the Spring Security **UserDetails** API to read about the interface **UserDetails**. In order to understand how **UserDetails** works in the Spring Security 3 program flow, refer to the Spring Security 3 guide.

```
publicinterface OWFUserDetails extends UserDetails{
/**
getOwfGroups
@return a Collection containing information about the OWF Groups that
this user is a part of.
*/
public Collection<OwfGroup> getOwfGroups();
/**
getDisplayName
@return String the Display Name of the user. This name is displayed in
the upper right hand corner of the banner. If not set, the username is
used instead. It is an optional field.
*/
public String getDisplayName();
}
```

5.7.1 OWFUserDetailsImpl

<<**ozone.security.authentication**>>

This is a sample implementation of the OWFUserDetails interface. It is not mandatory to use this implementation, it is only a sample.

Note: If a custom implementation is written, authorities must be write-accessible; create a setAuthorities method.

5.7.2 OWFGroup

<<**ozone.security.authorization.target**>>

This interface describes a single OWF user group. A group is a way of collecting OWF users and being able to assign widgets and other behaviors to them collectively**:**

```
package ozone.security.authorization.target;

/**
 * WidgetGroup
 *
 *
 * This interface describes a single OWF user group. A group is a way of
collecting
 * OWF users and being able to assign widgets and other behaviors to them
collectively.
 *
 * A group has one attribute, a name that should not change. The other
attributes are optional.
 *
 * Consider this similar to GrantedAuthorities.
 *
 */
public interface OwfGroup {
         /**
          *
          * @return the name of the Owf Group
          */
         public String getOwfGroupName();
       /**
          *
          * @return a drescription of the OWF group
          */
         public String getOwfGroupDescription();
        /**
          *
          * @return an email address that will reach someone if there are
problems with the group.
          * This is displayed to a group administrator in OWF
          */
         public String getOwfGroupEmail();
         /**
          * @return true if this is an active group
          */
         public boolean isActive();
}
```

5.7.3 OwfGroupImpl

<<**ozone.security.authorization.model**>>

This class implements an **OwfGroup**. It can be used as-is in a security implementation, or one can be created as-needed.

5.7.4 GrantedAuthorityImpl

<<**ozone.security.authorization.model**>>
This class implements the Spring Security 3.0 interface **GrantedAuthority**. It can be used as-is in a security implementation, or one can be created as-needed.

6 Themes

OWF includes three pre-made themes: a default OWF theme, and two high-contrast themes. OWF developers can create their own themes to include in the OWF user interface. End-users can switch between themes by selecting the Settings button on the toolbar and then choosing Themes.
Since the OWF 4 redesign, one line of code can change the overall color of OWF. Font size and font family are also adjustable. However, changing the entire look and feel of OWF requires additional work.

6.1 Changing the Default Theme

To change the default theme, change the **defaultTheme** value in the **tomcat\webapps\owf\WEB-INF\classes\application.yml** file.

6.2 Creating and Modifying Themes

OWF uses Compass, an open-source CSS stylesheet framework built on top of the SASS family of stylesheet languages. Two languages comprise SASS. OWF uses SCSS, the newer of the two languages. SCSS is a superset of CSS. It compiles into CSS. Compass is a framework for managing large SASS projects as well as augmenting and managing the SASS compilation process. For more information, see the SASS and Compass guide.

6.2.1 Prerequisites

To create and modify themes, the developer will need:

- Compass versions 0.11.3 *Note: Previous versions of OWF allowed for the use of Compass versions 0.11.3 to 0.11.7. However, style changes require the use of Compass 0.11.3 because newer versions of Compass automatically upgrade SASS to versions that cause issues with* ***acccessibility-bow.scss****.*
- SASS 3.1.3 (required by Compass)
- Ruby 1.9.2 (required by SASS and Compass)

To obtain these dependencies:

1. Install Ruby.
2. Use the included "gem" tool to install SASS and Compass by running **gem install compass –v 0.11.3** as an administrator.
3. Confirm that SASS and Compass are on the system PATH.

6.2.2 Layout of Themes Directory

To locate the theme files unzip the **tomcat/webapps/owf.war** and open the **themes** directory. The following figure shows the layout of the theme files:

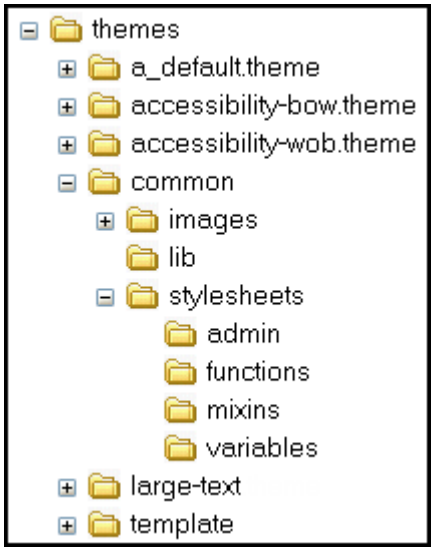


Figure 5: Themes Directory Structure

The following table provides a brief description of the themes folder and the files found in **owf.war/themes**:

Table 10: Theme File and Folder Description

File or folder name	Description
/compile_all_themes.bat or /compile_all_themes.sh	Shell scripts for Windows or UNIX that automate the process of compiling all of the themes
/watch_all_themes.bat or /watch_all_themes.sh	Shell scripts for Windows or UNIX that start watch processes on all themes, which will automatically recompile the stylesheets whenever a change is detected
/a_default.theme	Parent folder for the OWF default theme
/accessibility-bow.theme	Parent folder for the black-on-white theme
/large-text	Parent folder for the large text theme *Note: This feature is currently inactive in OWF.*
/accessibility-wob.theme	Parent folder for the white-on-black theme
/common	Parent directory for files that are likely to be used by most or all themes
/common/images	Directory for images that are common to many themes or can serve as defaults
/common/lib/owf_utils.rb	Functions (written in Ruby) that are useable within the SCSS file (should not need to be modified)
/common/stylesheets/	SCSS "partials" that build OWF themes
/common/stylesheets/_owf_all.scss	Central import file that imports all other files in the directory. If creating new SCSS partials, include them in this file.
/common/stylesheets/variables/	Contains variables used within the SCSS stylesheets
/common/stylesheets/variables/ _constants.scss	Variables values that should generally not change

File or folder name	Description
/common/stylesheets/variables/ _ext_overrides.scss *Note: Modifying this file may require an Ext JS developer license.*	Overridden default values for Ext JS's SCSS files
/common/stylesheets/variables/ *all other files	Variables that control aspects of the stylesheet generation and default values. The values of these variables may be overridden in a given theme
/template	Directory containing a theme template that contains every file listed in this table except for ***.css**. These files are as complete as possible without including properties that differentiates themes. To differentiate, developers must enter data.

The following table explains the files and folders that comprise a theme. The files and folders are found under a specific theme's directory like the black-on-white theme shown in Figure 6.

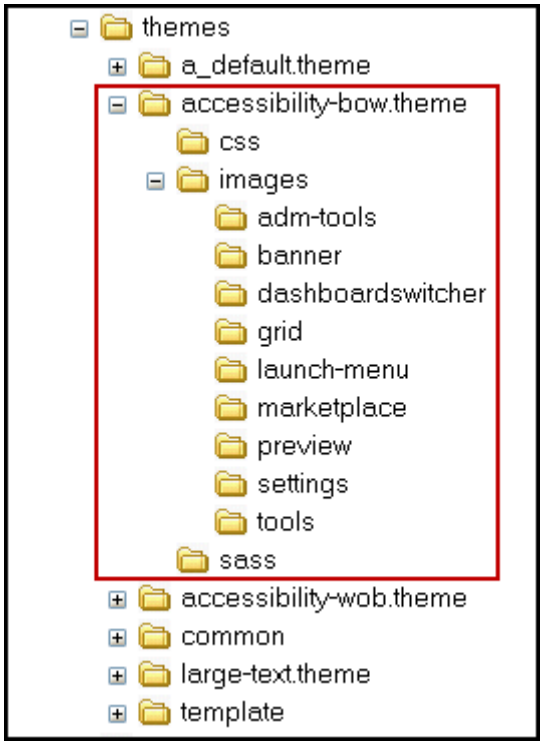


Figure 6: Black-on-White Theme Directory Structure

In OWF, the parent theme directories include **.theme** in their naming convention. Example: **accessibility-bow.theme.** Table 11: Theming Conventions uses an example theme named **example.theme** (this example is not included in the bundle).

Table 11: Theming Conventions

Convention	Description
example.theme/theme.json	Contains the theme metadata description that tells OWF where to find the theme's files at runtime and provides information about the description, author, and display name
example.theme/css/example.css	**The result of compiling a SCSS file from the SASS directory is stored here, in a file with the same name but with a** .css **extension instead of .scss**

Convention	Description
example.theme/images	**Location of theme-specific images. OWF searches for images here first. If they are not found, OWF searches** common/images
example.theme/images/preview/	Directory for theme screenshots
example.theme/sass/example.scss	**The main** .scss **file for a theme – overrides any desired variables from the common/stylesheets/variables/example files. It defines the theme background and imports the desired files from common/stylesheets. This file is essentially where the theme is defined.**

6.3 Creating a New Theme

1. Choose a theme name.The name should not have any spaces. It should be all lowercase. Words can be separated by hyphens.
2. Copy the **template** directory to **<theme_name>.theme**; substitute the name created in step 1 for the theme for **<theme_name>**.
3. Navigate into the **<theme_name>.theme** directory.
4. Open **theme.json** in a text editor and edit the following fields:

- The **name** attribute must use the **<theme_name>** chosen in step 1.
- The **display_name** attribute should contain a user-friendly, readable name for the theme (It can include spaces and capital letters.).
- The **CSS** attribute must include the **<theme_name>** as show below: **themes/<theme_name>.theme/css/<theme_name>.css.**
- All URL properties are relative to the context root. For now, ignore the **thumb** and **screenshots** fields, because images of a theme cannot exist until the theme is created.

1. Rename **sass/theme.scss** to **sass/<theme_name>.scss**
2. Edit **sass/<theme_name>.scss** (MANDATORY)
3. Set the **$theme-name** variable to the **<theme_name>** chosen in step 1.
4. Set the overall background. If the background is not set, it defaults to plain white.

The **sass/<theme_name>.scss** file is the primary place to create a custom theme by overriding variables. The files within **themes/common/stylesheets/variables** contain lists of variables that are available for overriding. Custom values for these variables should be defined directly below the **$theme-name** declaration (BEFORE importing **variables/**{*}). The lower part of this file imports all the SCSS partials that use the variable values to construct the stylesheets. *Note: For complex customizations, import statements can be deleted if equivalent functionality is custom-implemented by the theme.*

1. By default, a theme will search the **common/images** directory for image files. To use custom images in the new theme, override those images by placing new images in the **<theme_name>.theme/images** directory. Those new images will only apply to **<theme_name>**. New images must have identical pathnames and file names as the images being overridden relative to the **images** directory.
2. Compile your theme. This can be done in several ways.

- To do a one-time compile of only this theme, navigate to the **sass** directory and run **compass compile** in a terminal. You may pass in the **--force** option to force a recompile even when source files do not appear to be changed.
- To start a process that will continue to watch your SCSS for changes and recompile as needed, navigate to the **sass** directory and run **'compass watch'** in a terminal.
- To do a one-time compile of ALL themes, navigate to the **themes** directory and run the appropriate script.
    a. On Windows, this is done by running **compile_all_themes.bat**
    b. On UNIX, this is done by running **sh compile_all_themes.sh** *Note: The scripts assume that their current directory is the themes directory.*
- To start watches on ALL themes, navigate to the themes directory and run the appropriate script. These scripts assume that the current directory is the **themes** directory.
    a.
        i. On Windows, run **watch_all_themes.bat**. This will open a new, minimized command prompt for each theme, in which **compass watch** will be running.
        ii. On UNIX, run **sh watch_all_themes.sh** in a terminal. This will start **compass watch** as a background process for each theme. The script will not exit until the watches exit, and so is generally terminated with an interrupt signal.

1. Once the theme successfully compiles, verify that **<theme_name.css>** has been created in the **css** directory, and that it does not contain any error messages (These messages replace the entire normal output, so if errors exist they will be obvious).
2. Deploy OWF with the newly created theme. Do this one of two ways:

- For a development instance, run **grails –Duser=testAdmin1 run-app –https** from the top directory of the source tree.
- Build OWF and start the Tomcat server, as described in 3.3: OWF Bundle Description. To build OWF, run **ant** from the top directory of the source tree. To start the build server, run **start.bat** or **start.sh**located in **/tomcat**. *Note: Before running **ant**, the developer may need to run **ant init-build**.*

1. Log into OWF as a user or admin, and open the theme selector window which is located under the settings button on the toolbar.
2. Select and apply the new theme in the theme selector.

Note: Currently, there are no screenshots for the newly created theme.

1. Once the new theme is running, take some screenshots of it. Screenshots should be saved in the **<theme_name>.theme /images/preview** directory.
2. Edit **<theme_name>.json**. Set the **thumb** attribute to one of the theme screenshots of the entire browser viewing area. Next, add screenshot items to the **screenshots** array. These items are **JSON**objects that contain the attributes **url** and **description**. For an example of screenshot objects, see the **theme.json** files of other themes or the comment section at the bottom of the template **theme.json**.
3. In the OWF user interface, open the theme selector again and verify that the screenshots display.

6.4 Making Themes Usable Outside of owf.war

Once the creation of a custom theme is complete, (see 6.3: Creating a New Theme) it can be housed outside of the **owf.war** in a customizable location on the classpath. The following section, added to the **tomcat\webapps\owf\WEB-INF\classes\application.yml** file, allows an administrator to define where themes, help files, and js-plugins will reside.

```
Owf:
 //Locations for the  optional external themes and help directories.
 //Default: 'themes', 'help', and 'js-plugins' directories on the
classpath.
 //Can be configured to an arbitrary file path.  The followingpath styles
are supported:
 // 'file:/some/absolute/path' ('file:C:/some/absolute/path'on Windows)
    //  'classpath:location/under/classpath'
 //  'location/within/OWF/war/file'
 External:
    themePath: 'classpath:themes'
       helpPath: 'classpath:help'
       jsPluginPath:f 'classpath:js-plugins'
```

In order to make the themes usable from OWF, one of two scenarios will need to be implemented. Either the **\etc\tools\create-web-bundles.bat**(or **.sh**) command will need to be run in order to create gzipped **CSS** files, or uiperformance will need to be disabled.
When running the script, note that it takes the following arguments on the command line, the first of which is required:

- -**js externalJsLocation** (required)The location of the external js plugin folder.
- **-o owfLocation** The location of either the OWF **WAR** file or a directory where the **WAR** was extracted. Giving it the **WAR** file extracts it to a temp directory. It is then recreated where it was originally located after the rebundling is complete.
- **-e externalThemesLocation** (Optional)The location of the external themes folder. If it is unspecified, external themes are not bundled.

Once the **create-web-bundles.bat** (or **.sh**) has been executed, the included themes should be available from the theme selector in OWF.

6.5 Non-themable OWF Components

The sign-out page and its components are **not** theme-able.

6.6 Themable OWF Components

This section lists the various **SCSS** files that are used to construct the OWF stylesheets and the themable components that are controlled by those files. Changes to these files will affect **all** themes. All themable components are located in the **owf.war/themes/common/stylesheets** directory.

Table 12: Themable Components

File that must be modified	Themable Component(s)
_aboutWindow.scss	About Window
_adminWidget.scss	Admin Editors

File that must be modified	Themable Component(s)
_banner.scss	Toolbar User Menu
_buttons.scss	Buttons
_dashboardSwitcher.scss	Dashboard Switcher
_editWidget.scss	Admin Editors
_grid.scss	Grids
_launchMenu.scss	Favorites Menu
_main.scss	Drop-down Menu Form Headers: Window, Panel and Taskbar Keyboard Focus Loadmask Message Box Progress Bar Tooltips Etc.
_manageWindowContainers.scss	Dashboard Editors Widget Managers
_marketplaceWindow.scss	Marketplace Window
_portal.scss	Widget Portlets
_settingsWindow.scss	Administration Tools Window Settings Window
_systemWindow.scss	About Window Administration Tools Window Dashboard Switcher Settings Window
_themeSwitcher.scss	Theme Switcher Window
_widgetChrome.scss	Widget Chrome Menu Bar Widget Chrome Buttons Widget Chrome Sample Widget
_widget.scss	Favorites Menu Widget Icons Widget Switcher Widget Icons

Note: .scss files that apply to themes will be listed in the table above, other ****.scss*** files that are included in the bundle but do not pertain to themes may not appear in the table.*

7 Customizing OWF JavaScript

OWF uses JavaScript, CSS bundling, minification, compression and caching in order to optimize the downloading of resources to the browser. Nearly all JavaScript and CSS resources which get downloaded to the browser have a unique version string in their names. This allows the OWF server to tell the browser to permanently cache these name-specific resources.
Where it was previously necessary to open the **owf.war** file to edit the **JS** files in order to customize OWF, it is now possible to make many common modifications externally without extracting the **owf.war** file.
An external JavaScript plugins folder for customizations is included in the bundle at **tomcat/lib/js-plugins**, already containing the most commonly modified OWF JavaScript files. Any code inside the **init** method of these files will be run following the initialization of their parent file of the same name inside the **owf.war** file.
The following files are included in the bundle and ready to customize. The table explains which components in OWF will change when the corresponding files are customized:

Table 13: External JavaScript Files for Customizing OWF

Filename	Description of Customization
Banner.js	OWF banner on initialization, as well as functions that listen to the banner's dock and undock events
Dashboard.js	All dashboard types on initialization through their parent class
DashboardContainer.js	The container that holds all dashboard types on initialization
WidgetBase.js	The widget base, which includes the header
WidgetPanel.js	The inner panel of widgets, below the widget header
WidgetWindow.js	The containing window for widgets

*Note: Though it is still possible to open the **owf.war** file, edit files, and recreate the **WAR** to customize OWF, the external method is recommended to ensure code separation as well as to make it easier to upgrade to future versions of OWF. Accordingly, only modify JavaScript inside the **owf.war** file if the desired file to customize is not in the **js-plugins** folder.*

7.1 Configuring the External JavaScript Files

The external JavaScript plugins folder will only be included in OWF if the steps in 7.2: Deploy or Recreate External JavaScript/CSS are followed to include them, unless OWF is opened in debug mode. Since it requires action on the part of the implementer to utilize, there is no configuration property to quickly turn it on or off.
There is, however, a configuration parameter to set the name and location of the external folder to use in debug mode, which is located in the **application.yml** file located at **tomcat\webapps\owf\WEB-INF\classes\application.yml**. The **owf.external.jsPluginPath** property controls where OWF looks for the external JavaScript files when in debug mode. The default setting of this property is "classpath:js-plugins," so when OWF is rendered in debug mode, it will look in the classpath, which includes the **tomcat/lib/** folder, for the **js-plugins** folder and include its JavaScript files in the list returned to the client. This default location is depicted in the code snippet below, along with examples of other configuration options:

```
...
Owf:
  //Locations for theoptional external themes and help directories.
  //Default: 'themes','help', and 'js-plugins' directories on the
classpath.
```

```
  //Can be configured to an arbitrary file path.  The following
  //path styles are supported:
  // 'file:/some/absolute/path' ('file:C:/some/absolute/path' on Windows)
  //  'classpath:location/under/classpath'
  //  'location/within/OWF/war/file'
  External:
themePath: 'classpath:themes'
   helpPath: 'classpath:help'
   jsPluginPath: 'classpath:js-plugins'
...
...
```

7.2 Deploy or Recreate External JavaScript/CSS

In order to deploy any changes made to external files inside the **js-plugins** folder to the OWF server, the minified JavaScript files must be recompiled to include the external files. The instructions for including the external JavaScript files are as follows:

1. Shut down the OWF server if it is running.
2. From the command line, change the working directory to the **etc/tools**.
3. Execute the **create-web-bundles** script to include the external files in the minified JavaScript files inside the **owf.war** file.
    a. On a Windows system, use the following command:

```
create-web-bundles.bat -o ..\..\tomcat\webapps\owf.war
-js .. \..\tomcat\lib\js-plugins
```

b. On a Unix system, use the following command:

```
sh create-web-bundles.sh -o .. \..\tomcat\webapps\owf.war -js ..
\..\tomcat\lib\js-plugins
```

1. Shut down the OWF server if it is currently running.
2. Navigate to the **tomcat\webapps** folder and delete the **owf** folder if it exists.
3. Start OWF via **start.bat** or **start.sh**. Be sure to clear the browser's cache to ensure the most recent minified files are being received.

Note: For testing purposes, changes made to external JavaScript files can immediately be viewed by opening OWF in debug mode, instructions for which can be found in 7.5: Debugging JavaScript/CSS Problems.

7.3 JavaScript/CSS Bundle Naming Convention

Individual JavaScript and CSS source files are concatenated into bundle files. The bundle files are minified or compressed and versioned. The original source JavaScript and CSS files which make up these bundle files are also included in the **WAR** for reference and extension. Every JavaScript or CSS source file has a compressed and versioned file, and a **GZIP** compressed and versioned file. For example the OWF-server JavaScript bundle source file is **js/owf-**

server.js. The file is a concatenation of JavaScript files used for the main OWF page and it is readable. However, there are two other versions of the file referenced above:

- Main Source: **js/owf-server.js**
- Versioned and Minified: **js/owf-server__v6.0-GA-24385.js**
- Versioned and Gzip Compressed: **js/owf-server__v6.0-GA-24385.gz.js**

Note: The versioned, minified and Gzip compressed file names include a unique id number that changes with each release. In the examples above, the unique id number is 24385.

Every JavaScript and CSS source file follows the above naming convention.If the filename uses the following naming convention, it is versioned and minified:

```
_v<version>.<ext>
```

If the filename uses the following naming convention, it is versioned and **GZIP** compressed:

```
_v<version>.gz.<ext>
```

At runtime the OWF Web server will use the versioned and compressed bundle files. If the browser supports **GZIP** compression the compressed bundle file will be used, otherwise the minified bundle file will be used.

7.4 Toolbar Customization Walkthrough

One of the most commonly customized components of OWF is the toolbar, found at the top of the screen. For a specific implementation of OWF, it may be desirable to alter the OWF Toolbar by changing the logo or adding a new component such as a button or search box. This section discusses these modifications and puts forward the recommended approach to achieve them.

7.4.1 Toolbar Overview

The external toolbar object, **tomcat/lib/js-plugins/Banner.js**, corresponds to its parent, **/js/components/banner/Banner.js**, inside the **owf.war** file. When making customizations to the toolbar, the former, external **Banner.js** should always be the file being modified.
Note: For the purposes of this document, the term toolbar refers to the specific section of OWF that is directly above the dashboard space.
The default toolbar is shown below:

Figure 7: Toolbar, Logo and User Menu

7.4.2 Customizing the Toolbar Logo

The recommended way to modify the logo across all themes is through the external themes folder located at **tomcat/lib/themes**. The first step is to mimic the structure of the themes folder inside the **owf.war** file by creating folders to achieve the following structure in the external themes folder: **tomcat/lib/themes/common/images/banner/**. Inside the top-most **banner** folder, the custom logo file must be saved as **logo.png** in order to override the internal file of the same name.

7.4.3 Adding/Removing Toolbar Components

Through the external **Banner.js** file, developers can add Ext components, such as buttons, menus, and form fields, to the toolbar, as well as modify the appearance and functionality of existing components.

NOTE: If you are creating your own software developer kit (SDK), web application builder or website builder based on Sencha Ext JS, Sencha GXT, or Sencha Touch, then you need an OEM agreement with Sencha.

Note: Although the toolbar's default buttons are modifiable, they provide access to core OWF functionality and should be modified with extreme caution. The OWF team does not recommend removing these buttons.

To place custom components on the toolbar, add them to the **items** array as Ext components. The following example of the external **Banner.js** file shows an implementation that adds a search box component to both the docked and floating toolbars:

```
Ext.define('Ozone.plugins.Banner', {
 extend: 'Ext.AbstractPlugin',

 //Called after Banner.js initComponent() method inside the owf.war file
 init: function(cmp) {
    var banner = cmp;
    var searchAddedToPopOut = false;

    //Add a searchbox to the main banner
    this.addSearchBox(banner, banner.items.length - 1);
 },


 //Function added to add a searchbox to any passed in component at the
given position.
 addSearchBox: function(cmp, position) {
       cmp.insert(position, {
    xtype: 'searchbox', //OWF Custom Search Box Component
 cls: 'custom-searchbox',
 style: {
    margin: 5
 },
 listeners: {
    searchChanged: {
       fn: function(cmp, value) {
          var trimmedVal = Ext.String.trim(value);
          if (trimmedVal.length > 0) {
             alert("You searched on '" + trimmedVal +"'.");
          }
       },
       scope: this
    }
 }
 });
 }
});
```

7.4.4 Custom CSS

To customize the toolbar **CSS**:

- If the change applies to all themes: modify the following stylesheet **owf.war/themes/common/stylesheets/_banner.scss**
- If the change is theme specific: modify the following file **owf.war/themes/<theme_name>.theme/sass/<theme_name>.scss**

7.5 Debugging JavaScript/CSS Problems

OWF may be configured to not use JavaScript and CSS bundling and compression. Disabling this feature is useful for debugging because the source JavaScript and CSS will be included into the page instead of the bundles. To disable this feature add the line below to **application.yml** and restart OWF.

```
uiperformance.enabled = false
```

It is also possible to disable the JavaScript and CSS bundles at runtime in the browser. To disable the bundling and compression dynamically add **debug=true** to the OWF URL. See below: https://localhost:8443/owf/? debug=true

*Note: If the OWF URL returns an error, it may have appended the dashboard id to include a GUID number. If this error is returned, try adding **#guid= and the GUID** number after **true**.*
_For example: *https://localhost:8443/owf/?debug=true#guid=123456789_*h1.

8 Upgrading OWF

There are three main versions of OWF from which most people will be upgrading. . In order to install version 7.17.2, you must first upgrade to 7.17.1. Use the upgrade instructions specific to your version to upgrade to 7.17.1. Once 7.17.1 has been installed, follow the standard instructions for installing 7.17.2.

- A.2: Upgrading from version 7.0.1
- A.3: Upgrading from version 7.3.0
- A.4: Upgrading from version 7.14.2

Note: Due to resource constraints, Oracle and SQLServer database scripts have not been tested.

8.1 Upgrading from version 7.0.1

8.1.1 Part 1: Back up and update the security files

1. Backup everything:

Before starting the upgrade, backup the entire deployment of OWF and the corresponding database. Make sure all custom override configuration files have been included in the backup (In Tomcat, they are normally located in the **/lib** directory).

2. Install OWF:

The following example shows how an administrator might copy and unzip OWF from the bundle on **Unix**-type operating systems:

```
mkdir /opt/OWF
cp OWF-bundle-7.17.1.zip/opt/OWF
cd /opt/OWF
unzip OWF-bundle-7.17.1.zip
cd tomcat
```

The following example shows how an administrator might copy and unzip OWF from the bundle on a **Windows** operating systems:

1. Create a new directory from where OWF will run. This can be done via the *Windows* UI or the command prompt.
2. Copy **OWF-bundle-7.17.1.zip** to the new directory created in step a.
3. Right-click on **OWF-bundle-7.17.1.zip**, and select "open," "explore" or the command for the system's default zip/unzip program.
4. Unzip/unpack the bundle into the new directory created in step a.

The use of the bundled deployment archive provides all of the necessary mechanisms to deploy and run the Tomcat Web container on any Java 1.7 enabled system.

1. Update the OWFsecurityContext.xml and MPSecurityContext.xml configurations *(found in the* **tomcat** □ **lib** *directory):*

Update the security context file syntax:

1. **intercept-url** elements which include the **filters="none**" attribute are no longer supported, instead use additional '**http**' elements which contain a **security="none"** attribute and a **'pattern'**attribute equivalent to the **'pattern'** attribute on the old **'intercept-url'** element.

2. The **xsi:schemaLocation** attribute on the parent '**beans**' element must be updated to the following:

```
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
http://www.springframework.org/schema/security
http://www.springframework.org/schema/security/spring-security-3.2.xsd"
```

8.1.2 Part 2: Upgrade the Database

Upgrade database:
Before starting the database upgrade, shut down the older version of OWF's server. Then run the upgrade script(s) that correspond to the database in use and the appropriate starting and ending versions of OWF compatibility:
dbscripts\archive\MySqlPrefsUpgrade_v7.0.1_v7.16.sql
*Note: If updating with MySQL, be sure to modify the **.sql** script (mentioned above) with the appropriate schema name. For example:*
use OWF;
After upgrading the database, you have to accommodate significant changes to the dashboard data, as described in the following section.

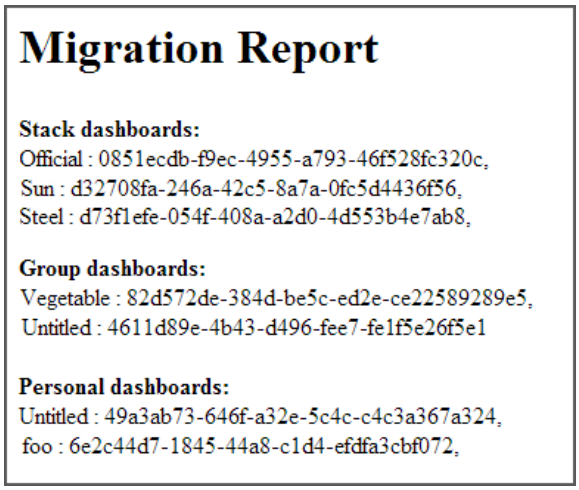
8.1.2.1 Migrate Legacy Dashboard Data

OWF 7.3 introduced major UI changes including the replacement of stacks and dashboards with applications and pages. Administrators can access a migration script to convert existing dashboards to Application Pages. It is important to note that group dashboards and stack dashboards, when migrated to applications, are not assigned an owner. This does not affect the application's function, but users and administrators cannot edit the application. Only the application's owner has permission to edit an application and its pages. Editing an application includes pushing the application to the Store and adding a page. To become the owner of a migrated application, an administrator must assign it to themselves using the Applications Manager (see "Assign to Me" in the OWF Administrator's Guide). Once the application is assigned to an administrator, they are its owner and have the ability to edit the application.
To update the legacy data, run the following migration script.

1. Open an Internet browser window and log into OWF as an administrator.
2. In a new browser tab, type in the following URL:

https://www.yourcompany.com:8443/owf/administration/migrateToApps

1. The migration script will automatically run. When finished, the migration service provides the developer with a brief report in the Web browser where it was executed. An example report is shown below:



Migration Report

Stack dashboards:
Official : 0851ecdb-f9ec-4955-a793-46f528fc320c,
Sun : d32708fa-246a-42c5-8a7a-0fc5d4436f56,
Steel : d73f1efe-054f-408a-a2d0-4d553b4e7ab8,

Group dashboards:
Vegetable : 82d572de-384d-be5c-ed2e-ce22589289e5,
Untitled : 4611d89e-4b43-d496-fee7-fe1f5e26f5e1

Personal dashboards:
Untitled : 49a3ab73-646f-a32e-5c4c-c4c3a367a324,
foo : 6e2c44d7-1845-44a8-c1d4-efdfa3cbf072,

Figure 8: Migration Report
If this migration was performed previously, and all legacy dashboards were converted to applications, then the output report will state "No dashboards to process."

8.1.3 Part 3: Completing the Upgrade

1. Reconfigure and re-apply customizations to the **\tomcat\lib** directory.
2. If a custom keystore was deployed in a previous build, the keystore for OWF will need to be manually configured in the same manner.
3. Copy any custom certificates from previous build to OWF 7.17.0. Usually this is found in **\tomcat\certs**.
4. If custom changes were applied to a previous **owf.war** file, such as skinning, reapply them.
5. The OWF upgrade is now complete.

8.2 Upgrading from version 7.3.0

8.2.1 Part 1: Back up and update the security files

1. Backup everything:

Before starting the upgrade, backup the entire deployment of OWF and the corresponding database. Make sure all custom override configuration files have been included in the backup (In Tomcat, they are normally located in the **/lib** directory).

1. Install OWF:

The following example shows how an administrator might copy and unzip OWF from the bundle on **Unix**-type operating systems:

```
mkdir /opt/OWF
cp OWF-bundle-7.17.1.zip/opt/OWF
cd /opt/OWF
unzip OWF-bundle-7.17.1.zip
cd tomcat
```

The following example shows how an administrator might copy and unzip OWF from the bundle on a **Windows** operating systems:

1. Create a new directory from where OWF will run. This can be done via the *Windows* UI or the command prompt.
2. Copy **OWF-bundle-7.17.1.zip** to the new directory created in step a.
3. Right-click on **OWF-bundle-7.17.1.zip**, and select "open," "explore" or the command for the system's default zip/unzip program.
4. Unzip/unpack the bundle into the new directory created in step a.

The use of the bundled deployment archive provides all of the necessary mechanisms to deploy and run the Tomcat Web container on any Java 1.7 enabled system.

1. Update the OWFsecurityContext.xml and MPSecurityContext.xml configurations *(found in the* **tomcat** **lib** *directory):*

Update the security context file syntax:

1. **intercept-url** elements which include the **filters="none**" attribute are no longer supported, instead use additional '**http**' elements which contain a **security="none"** attribute and a **'pattern'**attribute equivalent to the **'pattern'** attribute on the old **'intercept-url'** element.
2. The **xsi:schemaLocation** attribute on the parent '**beans**' element must be updated to the following:

```
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
http://www.springframework.org/schema/security
http://www.springframework.org/schema/security/spring-security-3.2.xsd
```

8.2.2 Part 2: Upgrade the Database

Upgrade database:
Before starting the database upgrade, shut down the older version of OWF's server. Then run the upgrade script(s) that correspond to the database in use and the appropriate starting and ending versions of OWF compatibility:
dbscripts\archive\MySqlPrefsUpgrade_v7.3.0_v7.16.0.sql
*Note: If updating with MySQL, be sure to modify the **.sql** script (mentioned above) with the appropriate schema name. For example:*
use OWF;

8.2.3 Part 3: Completing the Upgrade

1. Reconfigure and re-apply customizations to the **\tomcat\lib** directory.
2. If a custom keystore was deployed in a previous build, the keystore for OWF will need to be manually configured in the same manner.
3. Copy any custom certificates from previous build to OWF 7.17.0. Usually this is found in **\tomcat\certs**.
4. If custom changes were applied to a previous **owf.war** file, such as skinning, reapply them.
5. The OWF upgrade is now complete.

8.3 Upgrading from version 7.14.2

8.3.1 Part 1: Back up and update the security files

1. Backup everything:

Before starting the upgrade, backup the entire deployment of OWF and the corresponding database**.** Make sure all custom override configuration files have been included in the backup (In Tomcat, they are normally located in the **/lib** directory).

1. Install OWF:

The following example shows how an administrator might copy and unzip OWF from the bundle on **Unix**-type operating systems:

```
mkdir /opt/OWF
cp OWF-bundle-7.17.1.zip/opt/OWF
cd /opt/OWF
unzip OWF-bundle-7.17.1.zip
cd tomcat
```

The following example shows how an administrator might copy and unzip OWF from the bundle on a **Windows** operating systems:

1. Create a new directory from where OWF will run. This can be done via the *Windows* UI or the command prompt.
2. Copy **OWF-bundle-7.17.1.zip** to the new directory created in step a.
3. Right-click on **OWF-bundle-7.17.1.zip**, and select "open," "explore" or the command for the system's default zip/unzip program.
4. Unzip/unpack the bundle into the new directory created in step a.

The use of the bundled deployment archive provides all of the necessary mechanisms to deploy and run the Tomcat Web container on any Java 1.7 enabled system.

1. Update the OWFsecurityContext.xml and MPSecurityContext.xml configurations *(found in the* **tomcat** **lib** *directory):*

Update the security context file syntax:

1. **intercept-url** elements which include the **filters="none**" attribute are no longer supported, instead use additional '**http**' elements which contain a **security="none"** attribute and a **'pattern'**attribute equivalent to the **'pattern'** attribute on the old **'intercept-url'** element.
2. The **xsi:schemaLocation** attribute on the parent '**beans**' element must be updated to the following:

```
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
http://www.springframework.org/schema/security
http://www.springframework.org/schema/security/spring-security-3.2.xsd"
```

8.3.2 Part 2: Upgrade the Database

Upgrade database:
Before starting the database upgrade, shut down the older version of OWF's server. Then run the upgrade script(s) that correspond to the database in use and the appropriate starting and ending versions of OWF compatibility:
dbscripts\archive\MySqlPrefsUpgrade_v7.14.2_v7.16.0.sql

Note: If updating with MySQL, be sure to modify the ***.sql*** *script (mentioned above) with the appropriate schema name. For example:*
use OWF;

8.3.3 Part 3: Completing the Upgrade

1. Reconfigure and re-apply customizations to the **\tomcat\lib** directory.
2. If a custom keystore was deployed in a previous build, the keystore for OWF will need to be manually configured in the same manner.
3. Copy any custom certificates from previous build to OWF 7.17.1. Usually this is found in **\tomcat\certs**.
4. If custom changes were applied to a previous **owf.war** file, such as skinning, reapply them.
5. The OWF upgrade is now complete.

9 Clustering an OWF Environment

This following section details how to modify Tomcat and Apache HTTP Server configurations for clustering OWF. Configuring OWF in a clustered environment tends to differ widely between specific web application servers. Be sure to refer to an individual application server's documentation for specific details on how it should be configured for clustering.
Find general installation instructions and configurations required for Tomcat clustering at: {+}http://tomcat.apache.org/tomcat-6.0-doc/cluster-howto.html+.
For purposes of this document, assume that the Apache HTTP Server and Load Balancer have been installed on a host named **cluster** and OWF is on two separate hosts named **node1** and **node2**.

9.1 Tomcat Configuration

Navigate to **/tomcat/conf/server.xml** , and make the following changes:

1. Add the **jvmRoute** attribute to the **Engine** element. The attribute should be a string that uniquely identifies the node in the cluster:

```
<Engine name="Catalina" defaultHost="localhost" jvmRoute="node1">
```

1. Uncomment the **Cluster** element that is in the **Engine** element:

```
<Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster"/>
```

1. Comment out any **Connector** elements with their protocol attribute set to HTTP/1.1 and uncomment the **Connector** element with its protocol set to AJP/1.3. Use port="8009" and redirectPort="8443":

```
<Connector port="8009" protocol="AJP/1.3" redirectPort="8443"/>
```

Note: For max performance, uncomment the **AprLifecycleListener** (this isn't strictly clustering related).

9.2 OWF Configuration

Modify **tomcat/lib/ehcache.xml** to enable clustering of the second-level cache. By default, clustering is turned on, but it is configured to NOT communicate with other machines. To enable clustering, modify the **properties** attribute of the **cacheManagerPeerProviderFactory** element. This property should be a JGroups configuration string that configures the mechanism used to discover other cluster nodes. This can be configured explicitly with the host names of the other nodes, or it can be configured for automatic node discovery via multicast. See the JGroups documentation (http://www.jgroups.org/) for details and examples.
Configure **tomcat/lib/OzoneConfig.properties** to use the host and port through which clients will be accessing OWF and not the host and port of each particular node:

```
ozone.host = cluster
ozone.port = 8443
```

9.3 OS Configuration

For best performance, make sure the Apache Portable Runtime (APR) and Apache Tomcat Native libraries are installed on the node servers. The procedure is optional and installation varies by OS. Accordingly, be sure to refer to {+}http://tomcat.apache.org/native-doc/+ and {+}http://tomcat.apache.org/tomcat-6.0-doc/apr.html+ for information on setting up APR/Native on a specific OS.
Note: If the associated load balancer and node servers are behind a firewall, make sure all relevant ports are open for TCP/UDP (Examples: 8009, 8080, 8443, etc).

9.4 Apache HTTP / Load Balancer Configuration

In the following installation example, Apache Web Server (httpd) 2.x is used. Refer to {+}http://httpd.apache.org/+ for download access, installation instructions, and configuration information.

1. Install **mod_jk** (The Apache Tomcat Connector and load balancer). Please refer to {+}http://tomcat.apache.org/connectors-doc/index.html+ for download, install, and configuration. Depending on the distribution installed, the exact layout of configuration files vary, but the following changes need to be made to the server's httpd configuration:
2. Change the Listen configurations to 8080 and 8443, instead of 80 and 443:

```
Listen 8080
Listen 8443
```

3. Use a LoadModule directive to load mod_jk:

```
LoadModule jk_module modules/mod_jk.so
```

4. Configure mod_jk with the following properties:

- **JkWorkersFile** - This should be the location of a **workers.properties** file (described below)
- **JkLogLevel info** - Can be set to **'debug'** if extra logging is desired
- **JkOptions +ForwardDirectories** - allows paths like +https://localhost:8443/owf/+ to work
- **JkMount /owf** balancer* - tells **mod_jk** to foward requests for **/owf** to the worker named balancer (defined in **workers.properties**)

5. Make sure that **mod_ssl** gets loaded:

```
LoadModule ssl_module modules/mod_ssl.so
```

6. Configure SSL with the following properties:

- **SSLEngine** on
- **SSLCertificateFile** - the certificate for the server
- **SSLCertificateKeyFile** - the private key for the server
- **SSLCertificateChainFile** - the CA chain for the server
- **SSLCACertificateFile** - the CA used to validate client certs
- **SSLVerifyClient optional** - for behavior equivalent to tomcat's clientAuth="want" (can be changed as desired)

- **SSLOptions +ExportCertData** - needed for OWF to obtain cert information

7. Create a **workers.properties** file, which needs to be referenced in the **JkWorkersFile** property described above.

The names of workers need to be the same as the **jvmRoute** that was set on the corresponding node. The **worker.list** entry defines which workers are accessible from within the Apache server's httpd configuration.
The sample below includes three workers: the actual load balancer (whose type is set to 'lb') and a worker for each node.

```
#Expose the balanced worker, and also owfcluster02 so we can route
worker.list=balancer,node1
worker.balance.type=lb
worker.balancer.balance_workers=node1,node2
worker.node1.type=ajp13
worker.node1.host=node1
worker.node1.port=8009
worker.node2.type=ajp13
worker.node2.host=node2
worker.node2.port=8009
```

10 Deploying OWF to an Existing Tomcat Instance

The OWF development team uses Tomcat for all internal testing. The following instructions explain how to deploy a new instance of OWF to an already established Tomcat instance. For the purposes of documentation, we have named the tomcat instances "production" and "temporary."

1. Unzip the OWF bundle into a directory called **/owf-temp**. Some of the files from this bundle will be used in the Tomcat installation named production. *Note: If you are using the default HSQL database (which is not advised), you must copy the **prod.Db.script** too.*
2. Copy **/owf-temp/tomcat/bin/setenv.bat** and **/owf-temp/tomcat/bin/setenv.sh** into the corresponding folder in the production Tomcat installation.
3. Copy **/owf-temp/tomcat/webapps/owf.war** to the production Tomcat webapps directory.
4. To use the sample key store and client certificate in a local development or test environment, copy the **/owf-temp/tomcat/certs** directory to the production Tomcat **certs** directory (This may need to be created.).
5. Modify the production Tomcat file with the following configurations. Ensure that the file contains the "**Connector**" configuration (In the example below, port 8443 has been used.):

```
<Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true"
maxThreads="150" scheme="https" secure="true"
keystoreFile="certs/keystore.jks" keystorePass="changeit"
clientAuth="want" sslProtocol="TLS" />
```

The default security module uses the user's client certificate for authentication if it is available. To use the default security module (PKI only security) OR a custom security module with a user's client certificate for authorization, set the clientAuth to **want** or **true**. True requires a client certificate like the PKI only security module.
Set the key store and trust store parameters to the values that are appropriate for the production environment. The example above is using the default key store shipped with OWF. It is **only** appropriate for local development and testing environments.

6. Start production Tomcat which allows it to unpack the **WAR** files.

7. Stop Tomcat.

8. Copy the following from the temporary **tomcat/lib** directory in the distribution bundle to the production Tomcat **webapps/owf/WEB-INF/classes** directory:

- **OWFsecurityContext.xml**
- **owf-override-log4j.xml**
- **OzoneConfig.properties**
- **ozone-security-beans** directory
- **ehcache.xml**

9. If the sample security modules are in use, copy **/owf-temp/tomcat/lib/users.properties** to the *production* **/lib** directory.

10. Copy the **themes** and **help** directories from temporary **tomcat/lib** folder to the production **tomcat/lib** folder.

11. Start Tomcat.

11 Known Issues

11.1 Browser Issues

Launching any JavaScript-heavy app components, for example the Manager and Editor App Components accessed via the Administration App, in Internet Explorer versions 7 and 8 consumes system memory that won't be flushed or released until Internet Explorer is exited and restarted. Again, this is currently only an Internet Explorer issue.

11.2 User Interface Issues

Changes in screen resolution may render widgets unviewable.
The positioning of the widgets is absolute. This means that when changing from a larger monitor to a smaller monitor, or when changing from a higher screen resolution to a lower screen resolution, some floating windows may be either partially or fully off the viewable region of the screen. Currently there is no remedy for this issue; however, closing and re-adding the widgets (from the Favorites Menu) will reset its position and, therefore, render it viewable again.

Large widgets in Accordion layout may cause unexpected behavior.
The accordion layout currently may enter an unexpected state when widgets with height exceeding the maximum screen real estate are added to the right upper region. This can be remedied if a system administrator modifies the dashboards state. This issue will be fixed in a future release.

11.3 Widget Technology Issues

Java Applet Widgets always sit on top of other Widgets (z-index issue).
There is a documented issue with Java applets not obeying proper z-indexing, the effect being that an applet will appear over everything else in OWF:
{+}http://bugs.sun.com/bugdatabase/view_bug.do;jsessionid=6a434ce1408465ffffffff87e84af5d233a32?bug_id=6646289+

Flex Widgets always sit on top of other widgets (z-index issue).
Flex has a known issue with DHTML and z-index ordering. The default wmode for flex is window with two other options; transparent and opaque. In order for Flex Widgets to adhere to the proper z-index ordering the wmode must be set to something other than the default.

Silverlight Widgets always sit on top of other widgets (z-index issue).
Silverlight has a known issue with DHTML and z-index ordering. The default windowless mode for Silverlight is false. In order for Silverlight Widgets to adhere to the proper z-index ordering the windowless mode must be set to true.

Google Earth Plugin Widgets always sit on top of other widgets (z-index issue).
The Google Earth browser plugin currently does not conform to the normal z-index rules of html. This will cause the plugin to remain on top of any other floating windows that may be on the screen. If this plugin is being used, it is recommended not to use it in the Desktop Layout. It can be used in any of the other static layouts but windows launched from the toolbars may be rendered unreachable by the plugin.

11.4 Database Issues

Oracle scripts should be executed via the SQL*Plus command line.
There have been reported issues using Oracle's browser-based administration console to upload and run the OWF create and update scripts. Stray characters are getting inserted into

the database, causing **JSON** parsing errors at runtime. Executing the scripts through the SQL*Plus command line utility eliminates this issue.

12 Deploying OWF to JBoss EAP

Version	Status
6.2.0 GA	Tested

12.1 JBoss Enterprise Application Platform (EAP) v6.2.0 GA

The following guide assumes that the JBoss installation is located in the `/opt/jboss` folder; adjust the instructions accordingly if it is in another location.

12.1.1 Generate Keystore and Self-Signed Certificate

```
keytool -genkey -noprompt \
    -keystore /opt/jboss/standalone/configuration/default.jks -storepass
password \
    -alias owf -validity 10950 -keyalg RSA -keysize 2048 -keypass
password \
    -dname "CN=owf, OU=owf, OU=owf, L=owf, ST=owf, C=us"
```

12.1.2 Configure Network Bindings and Ports

Add the HTTPS connector configuration to the web subsystem element of `/opt/jboss/standalone/configuration/standalone.xml`

```
<subsystem xmlns="urn:jboss:domain:web:1.5" default-virtual-
server="default-host" native="false">
 ...
    <connector name="https" protocol="HTTP/1.1" scheme="https" socket-
binding="https" enabled="true">
        <ssl name="ssl" key-alias="owf" password="password" certificate-
key-file="/opt/jboss/standalone/configuration/default.jks"
protocol="TLSv1" verify-client="false"/>
    </connector>
 ...
</subsystem>
```

12.1.3 Configure Data Source

Edit the `datasources` configuration of `/opt/jboss/standalone/configuration/standalone.xml` to include a data source for OWF, as in the example below.

```
<subsystem xmlns="urn:jboss:domain:datasources:1.1">
    <datasources>
```

```
        <datasource jndi-name="java:jboss/datasources/OWF" pool-
name="OWF" enabled="true" use-java-context="true">
            <connection-url>jdbc:h2:mem:test;DB_CLOSE_DELAY=-
1;DB_CLOSE_ON_EXIT=FALSE</connection-url>
            <driver>h2</driver>
            <security>
                <user-name>sa</user-name>
                <password>sa</password>
            </security>
        </datasource>
        <drivers>
            <driver name="h2" module="com.h2database.h2">
                <xa-datasource-class>org.h2.jdbcx.JdbcDataSource</xa-
datasource-class>
            </driver>
        </drivers>
    </datasources>
</subsystem>
```

```
# Docker Postgres standalone.xml
<subsystem xmlns="urn:jboss:domain:datasources:1.1">
    <datasources>
        <datasource jndi-name="java:jboss/datasources/OWF"
                    pool-name="OWF"
                    enabled="true"
                    use-java-context="true"
                    use-ccm="true">
            <connection-
url>jdbc:postgresql://database:5432/owf</connection-url>
            <driver-class>org.postgresql.Driver</driver-class>
            <driver>postgresql-jdbc4</driver>
            <pool>
                <min-pool-size>2</min-pool-size>
                <max-pool-size>20</max-pool-size>
                <prefill>true</prefill>
            </pool>
            <security>
                <user-name>owf</user-name>
                <password>owf</password>
            </security>
            <validation>
                <check-valid-connection-sql>SELECT 1</check-valid-
connection-sql>
                <validate-on-match>false</validate-on-match>
                <background-validation>false</background-validation>
                <use-fast-fail>false</use-fast-fail>
            </validation>
        </datasource>
        <drivers>
            <driver name="postgresql-jdbc4" module="org.postgresql"/>
        </drivers>
    </datasources>
</subsystem>
```

12.1.4 Extract the OWF bundle

1. Unzip the OWF bundle into a temporary directory such as `/tmp/ozone/`

12.1.5 Copy OWF JBoss configuration module

1. Unzip the `owf-jboss-modules.zip` file inside `/tmp/ozone` into `/opt/jboss/modules/system/layers/base`
    a. The main OWF configuration file is located at: `/opt/jboss/modules/system/layers/base/ozone/framework/main/application.yml`
    b. Additional configuration files, including the Spring bean bindings and Spring Security settings, are located in: `/opt/jboss/modules/system/layers/base/ozone/framework/main/ozone/framework/`

12.1.6 Deploy the OWF WAR

1. Copy the `owf.war` file from `/tmp/ozone/tomcat/webapps/owf.war` to the `/opt/jboss/standalone/deployments` folder.
2. Start the JBoss application server.