User Interface Guide
Oracle Banking Digital Experience
Release 21.1.0.0.0

Part No. F40800-01

May 2021

ORACLE

User Interface Guide

May 2021

Oracle Financial Services Software Limited

Oracle Park

Off Western Express Highway

Goregaon (East)

Mumbai, Maharashtra 400 063

India

Worldwide Inquiries:

Phone: +91 22 6718 3000

Fax:+91 22 6718 3001

www.oracle.com/financialservices/

Table of Contents

1. Preface

1.1 Intended Audience

This document is intended for the following audience:

- Customers
- Partners

1.2 Documentation Accessibility

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

1.3 Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit

http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit

http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

1.4 Structure

This manual is organized into the following categories:

Preface gives information on the intended audience. It also describes the overall structure of the User Manual.

The subsequent chapters describes following details:

- Introduction
- Preferences & Database
- Configuration / Installation.

1.5 Related Information Sources

For more information on Oracle Banking Digital Experience Release 21.1.0.0.0, refer to the following documents:

- Oracle Banking Digital Experience Installation Manuals

2. Pre-requisite

OHS software along with instance should be available for use.

For further detailed configuration of Oracle HTTP Server, please refer to
https://docs.oracle.com/middleware/12213/webtier/administer-ohs/toc.htm

Home

3. User Interface Build

The current GUI build is based on Grunt.

Grunt is a JavaScript Task Runner - an automation tool for performing repetitive tasks like minification, compilation, unit testing, linting etc.

The tasks performed during a typical GUI build are:

- Pre Build checks (For some development rules)
- ESLint for the JS files.
- SCSS compilation to CSS
- CSS optimization
- HTML minification
- JS minification
- Require JS optimization to pack all the dependencies of a component into single file.
- Generate integrity for all component files.
- Cache Busting for resources.

Creating component artifacts for metadata generated by UI Workbench:

Follow steps below to generates the artifacts from metadata

- First make sure that NodeJS is installed on the machine
- Make sure that swagger documentation (JSON) is hosted and available on some server.
- Place **com.ofss.digx.utils.uiworkbench** and **obdx-ui-workbench-core** as sibling directory to **channel**, thus making all three directories in the same level.
- Open a terminal inside **obdx-ui-workbench-core** directory and run following commands
    - **npm install**
    - **npm run-script build**
- In **com.ofss.digx.utils.uiworkbench** directory open the package.json file and remove dependency of **@obdx/uiworkbench-core**
- In **com.ofss.digx.utils.uiworkbench** directory open the swagger/mapping.json file and replace all the instances of **http://mum00boa.in.oracle.com:18777/swagger/json/openapi.json** to locally available **openapi.json** URL.
- Inside **com.ofss.digx.utils.uiworkbench** directory and run following commands
    - **npm install**
    - **npm link ../obdx-ui-workbench-core**

- Remove npm install @obdx/uiworkbench-core in **generate-artifacts.sh** which is present under com.ofss.digx.utils.uiworkbench
- Execute **./generate-artifacts.sh**

Running UI Build:

Follow steps below to run UI Build:

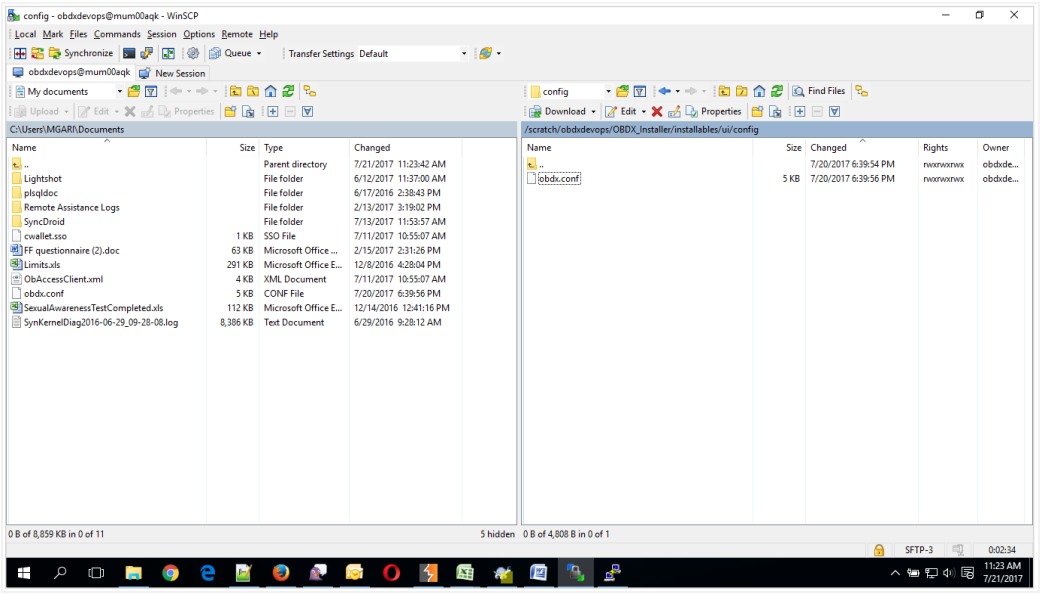
- First make sure that NodeJS is installed on the machine and grunt is available in global scope. Make sure to refer _**build/package.json** to check the apt NodeJS version for the OBDX Release.
- Open terminal inside channel/_build folder and run **npm install** to setup the UI Workspace**.**
- Now run **build.sh** to run the build.

Home

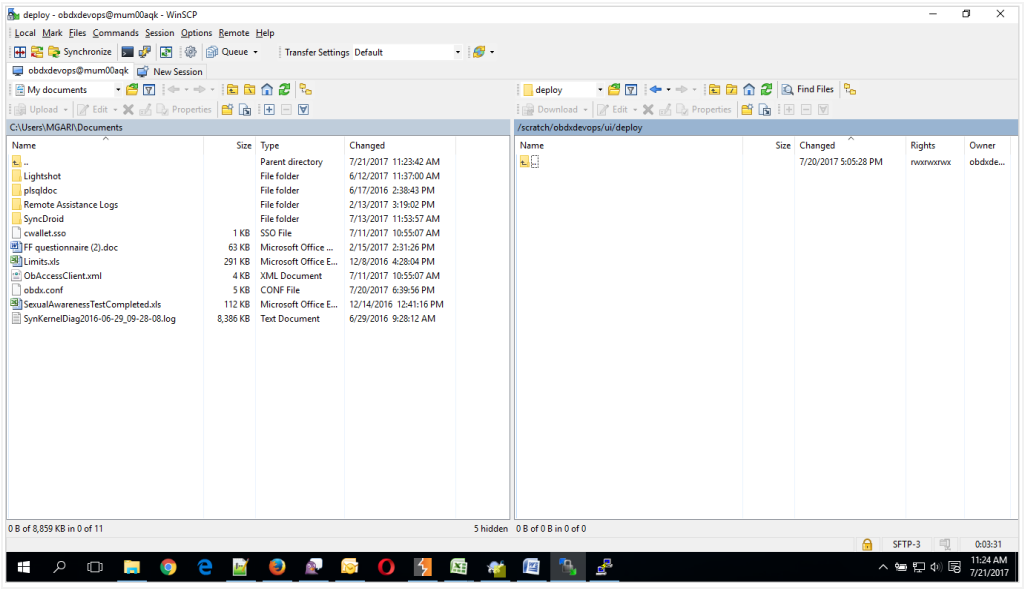
4. UI deployment

Below steps needs to be performed for UI deployment on OHS server.

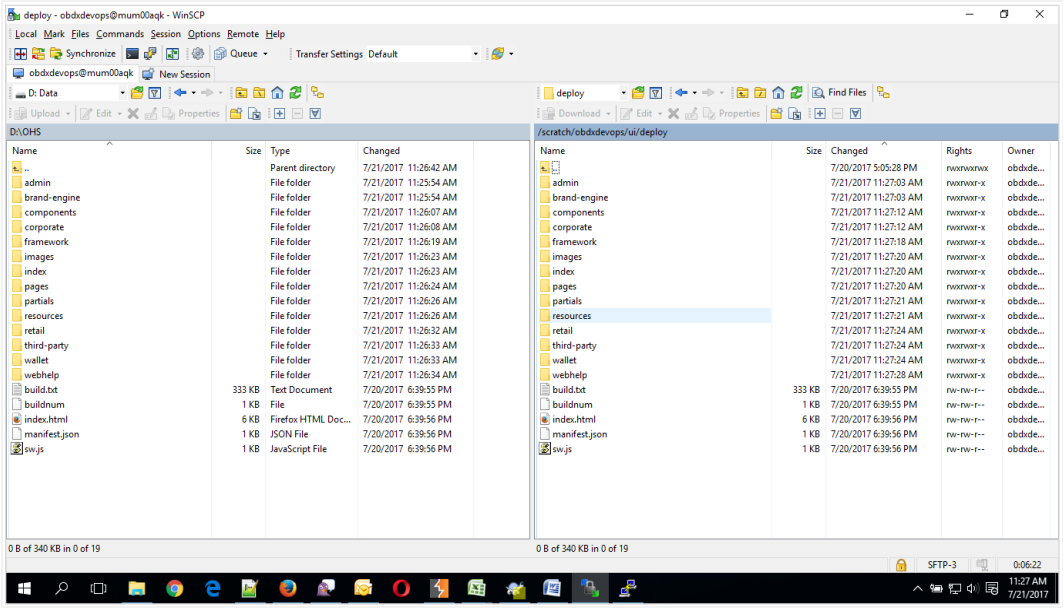
Copy the obdx.conf from OBDX_Installer/installables/ui/config directory into the instance config directory (where httpd.conf is present). httpd.conf file is present at **{DOMAIN_HOME}/config/fmwconfig/components/OHS/{componentName}**



- Create a directory where obdx UI files would be deployed on OHS server.



- Copy all files / directories from OBDX_Installer/installables/ui/deploy into newly created directory.



Home

5. Configuration to run UI on Oracle HTTP Server

Make sure following OHS modules must be loaded

- mod_rewrite.so
- mod_deflate.so
- mod_expires.so
- mod_mime.so
- mod_headers.so

Following are the changes needed to be done in the obdx.conf file and place this file in same folder where httpd.conf file exists.

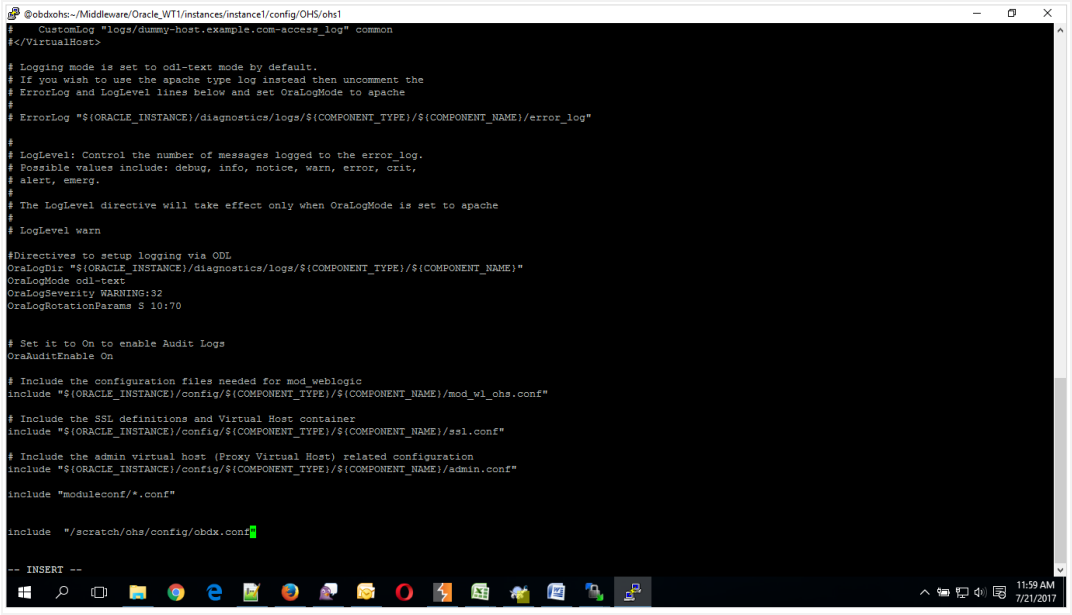
1. Replace the <CHANNEL_PATH> (all occurrences) with the newly created directory (from previous UI deployment step).
2. Configuration for Content Security Policy, refer to the below document

   Oracle Banking Digital Experience Security Guide

Include the obdx.conf into httpd.conf using below configuration

```
include “obdx.conf" (needs to be added in httpd.conf)
```

Read obdx.conf for inline documentation.



Following are the changes need to be done in mod_wl_ohs.conf which is present at {**DOMAIN_HOME**}**/config/fmwconfig/components/OHS/{componentName}**

Copy below configuration into mod_wl_ohs.conf

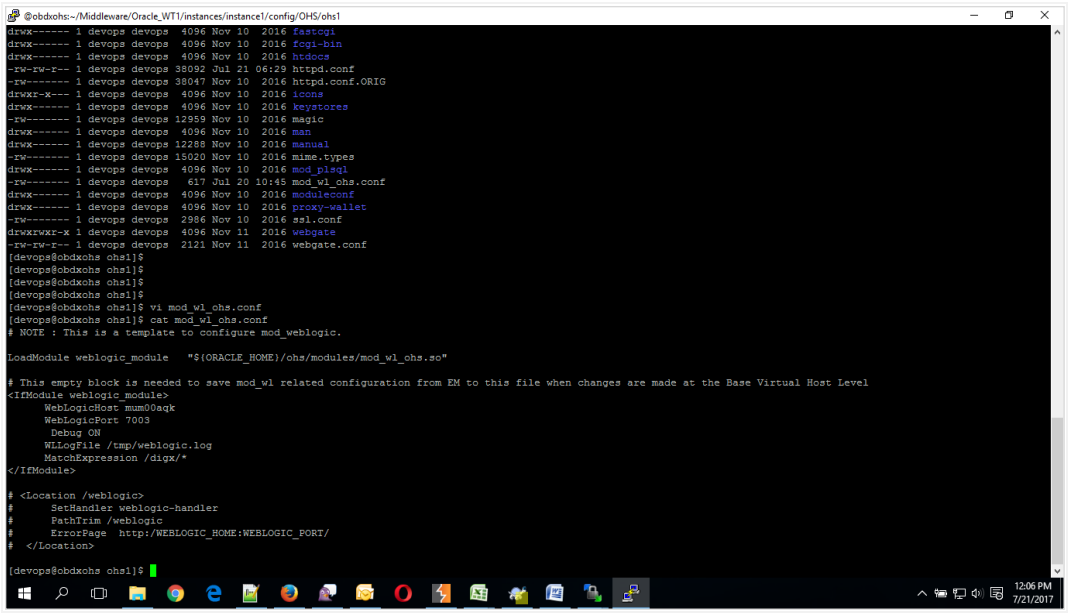
```
<IfModule weblogic_module>
    WebLogicHost HOSTNAME
     WebLogicPort MANAGE_SERVER_PORT
     Debug ON
  WLLogFile DIR/FILENAEME
     MatchExpression /digx/*
     MatchExpression /digx-auth/*
     MatchExpression /digx-social/*
          </IfModule>
```

Configure below properties

a. HOSTNAME – Weblogic server hostname (where OBDX weblogic domain is configured)
b. MANAGE_SERVER_PORT – Weblogic manage server port (where OBDX application is deployed)
c. DIR / FILENAME – Path where log file should be generated

Sample configuration (for reference purpose only)

```
<IfModule weblogic_module>
          WebLogicHost wls_server1
         WebLogicPort 7003
          Debug ON
          WLLogFile /tmp/weblogic_obp.log
          MatchExpression /digx/*
</IfModule>
```


@obdxohs:~/Middleware/Oracle_WT1/instances/instance1/config/OHS/ohs1
drwx------ 1 devops devops 4096 Nov 10 2016 fastcgi
drwx------ 1 devops devops 4096 Nov 10 2016 fcgi-bin
drwx------ 1 devops devops 4096 Nov 10 2016 htdocs
-rw-rw-r-- 1 devops devops 38092 Jul 21 06:29 httpd.conf
-rw------- 1 devops devops 38047 Nov 10 2016 httpd.conf.ORIG
drwxr-x--- 1 devops devops 4096 Nov 10 2016 icons
drwx------ 1 devops devops 4096 Nov 10 2016 keystores
-rw------- 1 devops devops 12959 Nov 10 2016 magic
drwx------ 1 devops devops 4096 Nov 10 2016 man
drwx------ 1 devops devops 12288 Nov 10 2016 manual
-rw------- 1 devops devops 15020 Nov 10 2016 mime.types
drwx------ 1 devops devops 4096 Nov 10 2016 mod_plsql
-rw------- 1 devops devops   617 Jul 20 10:45 mod_wl_ohs.conf
drwx------ 1 devops devops 4096 Nov 10 2016 moduleconf
drwx------ 1 devops devops 4096 Nov 10 2016 proxy-wallet
-rw------- 1 devops devops 2986 Nov 10 2016 ssl.conf
drwxrwxr-x 1 devops devops 4096 Nov 11 2016 webgate
-rw-rw-r-- 1 devops devops 2121 Nov 11 2016 webgate.conf
[devops@obdxohs ohs1]$
[devops@obdxohs ohs1]$
[devops@obdxohs ohs1]$
[devops@obdxohs ohs1]$
[devops@obdxohs ohs1]$ vi mod_wl_ohs.conf
[devops@obdxohs ohs1]$ cat mod_wl_ohs.conf
NOTE : This is a template to configure mod_weblogic.

LoadModule weblogic_module   "${ORACLE_HOME}/ohs/modules/mod_wl_ohs.so"

This empty block is needed to save mod_wl related configuration from EM to this file when changes are made at the Base Virtual Host Level
<IfModule weblogic_module>
      WebLogicHost mum00aqk
      WebLogicPort 7003
       Debug ON
      WLLogFile /tmp/weblogic.log
      MatchExpression /digx/*
</IfModule>

<Location /weblogic>
SetHandler weblogic-handler
PathTrim /weblogic
ErrorPage http:/WEBLOGIC_HOME:WEBLOGIC_PORT/
</Location>

[devops@obdxohs ohs1]$
12:06 PM
7/21/2017

Home

6. Oracle HTTP Server Commands

6.1 Starting Oracle HTTP Server Instances from the Command Line

You can start up Oracle HTTP Server instances from the command line via a script.

1. Ensure that Node Manager is running.
2. Enter the following command:

```
Linux or UNIX: $DOMAIN_HOME/bin/startComponent.sh componentName

Windows: %DOMAIN_HOME%\bin\startComponent.cmd componentName
```

For example:

```
$DOMAIN_HOME/bin/startComponent.sh ohs1
```

The startComponent script contacts the Node Manager and runs the nmStart() command.

When prompted, enter your Node Manager password. The system responds with these messages:

```
Successfully started server componentName...

Successfully disconnected from Node Manager...



Exiting WebLogic Scripting Tool.
```

6.2 Stopping Oracle HTTP Server Instances from the Command Line

You can stop Oracle HTTP Server instances from the command line via a script.

Enter the following command:

```
Linux or UNIX: $DOMAIN_HOME/bin/stopComponent.sh componentName

Windows: %DOMAIN_HOME%\bin\stopComponent.cmd componentName
```

For example:

```
$DOMAIN_HOME/bin/stopComponent.sh ohs1
```

This command invokes WLST and executes the nmKill() command.
The stopComponent command will not function if the Node Manager is not running.

For more commands refer the following URL:

https://docs.oracle.com/middleware/1221/webtier/administer-ohs/getstart.htm

Home

7. Configuring User Interface

All the UI configurations are available in config.js while which is present under the <CHANNEL_PATH>\framework\js\configurations directory. JavaScript object for the configuration is declare by the name “configuration”. Application freeze this object so its value cannot be change in running memory.

Category of the configuration:

i18n: All the internalization specific configuration mentioned in this. Currently this category have list of rtl locales

```
i18n: {

        rtlLocales: ["ar", "he", "ku", "fa", "ur", "dv", "ha", "ps", "yi"]

    }
```

Sharding: Domain sharding is a technique used to increase the amount of simultaneously downloaded resources for a particular website by using multiple domains. This allows websites to be delivered **faster** to users as they do not have to wait for the previous set of resources to be downloaded before beginning the next set. Implementer can introduce 3 additional domains for the UI

1. apiBaseURL: If the HTTP server and the application server are on same host, the property is set as “” otherwise set to host name and port of the application server. imageResourcePath: The base path from which the image resources are to be fetched. It can also be a relative path pointing to the same domain the page is running on or a fully qualified path to different server on which images are hosted
2. webHelpContentURL: Domain where the web help content is hosted.

```
sharding: {

        imageResourcePath: "./images",

        apiBaseURL: "",

        webHelpContentURL : ""

    }
```

Service Worker: A service worker is a script that your browser runs in the background, separate from a web page, opening the door to features that don't need a web page or user interaction. The core feature discussed in this tutorial is the ability to intercept and handle network requests, including programmatically managing a cache of responses. Implementer can enable or disable it by changing this property.

```
serviceWorker: {
```

```
        enabled: true

    }
```

Authentication: OBDX product ships with two type of authentication methods:

1. OAM Authentication
2. Non OAM Authentication (OBDXAuthenticator)

Configuring OAM Authentication set type as OAM and also provide the provider URL of OAM in providerURL property.

For Non OAM set type as OBDXAuthenticator.

In the application, setting secure and public page is required. For this two properties are exposed as pages.securePage and pages.publicPage. As name suggest pages.securePage have the pathname of secure page and pages.publicPage have the pathname of public/unsecure page.

```
authentication: {

        type: "OBDXAuthenticator",

        providerURL: "",

        pages: {

           securePage: "home.html",

           publicPage: "index.html"

        }

    }
```

Third Party API's: Some of the application module required integration with third party provider like facebook, linkedin, google etc. So in this category we maintained all the sdk url, api keys and provider url of third party api's

```
thirdPartyAPIs: {

        facebook: {

           url: "",

           sdkURL: "",
```

```
            apiKey: ""
        },
        linkedin: {
            sdkURL: "",
            apiKey: ""
        },
        googleMap: {
            url: "",
            sdkURL: "",
            apiKey: ""
        }
}
```

Oracle Jet: OBDX UI used Oracle Jet as the library. Oracle Jet also exposed over the CDN (content delivery network). So implementer has the choice to Oracle Jet as local deployment or from CDN. In hostedAt property supports two values “**cdn**” or “**local**”. **baseUrl** property used for base url of the Oracle Jet and version property used for the used Oracle Jet Version.

```
oracleJet: {
        hostedAt: "cdn",
        baseUrl: "https://mumaa012.in.oracle.com/jet",
        version: "7.1.0"
    }
```

API Catalogue: This category used for several context root available in OBDX API's and their default versions.

```
apiCatalogue: {
        base: {
            contextRoot: "digx",
            defaultVersion: "v1"
        },
        extended: {
            contextRoot: "digx/ext",
            defaultVersion: "v1"
        },
        social: {
            contextRoot: "digx-social",
            defaultVersion: "v1"
        },
        "digx-auth": {
            contextRoot: "digx-auth/ext",
            defaultVersion: "v1"
        },
        "digx-auth-extended": {
            contextRoot: "digx-auth",
            defaultVersion: "v1"
        }
    }
```

System Configuration: This category of configuration is used for system level properties. Brief description of properties are below:

componentAccessControlEnabled: Component access check(through role transaction mapping) is enabled or not. Depending of this property menu or link will filtered.

requestThrottleSeconds: OBDX UI can cached service responses and it also distribute one API response to several caller. For example if 3 widgets calling same API, in this case application fire only one API and distribute its response to all the callers. requestThrottleSeconds property used for caching time of the response. Unit is in second. It means if you set requestThrottleSeconds as 5(second) it means if application fire same API within 5 second application return the same response which it fire earlier.

defaultEntity: Default entity if entity cannot be derived.

sslEnabled: SSL is enabled or not.

loggingLevel: Logging level of OBDX UI.

buildTimestamp: Time stamp of the build.

```
system: {
    componentAccessControlEnabled: true,
    requestThrottleSeconds: 5,
    defaultEntity: "",
    sslEnabled: true,
    loggingLevel: "LEVEL_ERROR",
    buildTimestamp: BuildFingerPrint.timeStamp
}
```

Analytics: This category of configuration is used for enabling or disabling third party and OBDX analytics in application.

```
analytics: {
    thirdPartyAnalytics: {
        enabled: false,
        analyticsProvider: ""
    },
```

```
    obdxAnalytics: {
      enabled: false,
      eventsThreshold: 5,
      inactivityTimeout: 10 * 60 * 1000
    }
  }
```

Development Configuration: This category of configuration is used during development phase. This should be disabled (development.enabled set as false) in the production mode. In this category we also have property for enabling accessibility checks during run time.

```
development: {
    enabled: false,
    checkAccessibility: false,
    axeUrl: "https://cdnjs.cloudflare.com/ajax/libs/axe-core/3.3.2/axe.min.js"
  }
```

Home

8. Language Pack

Out of box OBDX comes with two language i.e. French and Arabic. Language pack of these languages are shipped along with the product. Please note since translation is a continuous process so some or the translation can be missing in the language pack which will be updated in next patch set release. The resource bundle key which translation is missing, you find the English string in place of the actual translated string.

8.1 Adding new Language

Implementer can add new language in the application by adding new row in **digx_fw_locale** table. This table has two columns locale code the description which comes in the drop down.

Example: For Arabic and French implementer can run following script respectively on OBDX Schema

```
insert into digx_fw_locale (code, description) values ('ar', 'Arabic');

insert into digx_fw_locale (code, description) values ('fr', 'Français');
```

8.2 Deployment of the Language pack

Language pack can be classified in the following types

Database Scripts:

1. Login to OBDX Schema
2. Execute following SQL files :

```
OBDX_<VERSION>_TRANSLATION_PACK\<LOCALE>\seed\digx_fw_error_messages.sql
OBDX_<VERSION>_TRANSLATION_PACK\<LOCALE>\seed\digx_fw_info_messages.sql
```

3. Commit the changes

```
commit;
```

Weblogic Configuration:

1. Copy all files/ directories from OBDX_<VERSION>_TRANSLATION_PACK\<LOCALE>\config to ${OBDX_HOME}\config hosted on Weblogic Server

---

Note: The path for ${OBDX_HOME}\config can be found under Managed Server classpath which is accessible via Weblogic Administration

---

UI Configuration:

1. Copy complete OBDX_<VERSION>_TRANSLATION_PACK\<LOCALE>\channel\resources\nls\<LOCALE> directory to <CHANNEL_PATH>/resources/nls/
2. Create a new <LOCALE> directory in <CHANNEL_PATH>/partials/help
3. Copy all existing files from <CHANNEL_PATH>/partials/help to <CHANNEL_PATH}/partials/help/<LOCALE>
4. Override all help files from OBDX_<VERSION>_TRANSLATION_PACK\<LOCALE>\channel\partials\help\<LOCALE> to <CHANNEL_PATH>/partials/help/<LOCALE>

Home

9. Configuring Different URL's on the Basis of Enterprise roles

To enable URL separation based on enterprise roles using custom header name and value, the following queries needs to be executed in **DIGX_FW_CONFIG_ALL_B** table

```
Insert into DIGX_FW_CONFIG_ALL_B (PROP_ID, CATEGORY_ID,
PROP_VALUE,FACTORY_SHIPPED_FLAG, PROP_COMMENTS, SUMMARY_TEXT,
CREATED_BY, CREATION_DATE,

LAST_UPDATED_BY, LAST_UPDATED_DATE, OBJECT_STATUS,
OBJECT_VERSION_NUMBER)

values
('IS_LOGIN_SEPARATION_ENABLED','SecurityConstants','true','N',null,'Is
login separation enabled','ofssuser',sysdate,'ofssuser',sysdate,'Y',1);
```

This query enables the URL separation mechanism. By default the URL separation mechanism is not enabled.

```
Insert into DIGX_FW_CONFIG_ALL_B (PROP_ID, CATEGORY_ID, PROP_VALUE,
FACTORY_SHIPPED_FLAG, PROP_COMMENTS, SUMMARY_TEXT, CREATED_BY,
CREATION_DATE, LAST_UPDATED_BY, LAST_UPDATED_DATE, OBJECT_STATUS,
OBJECT_VERSION_NUMBER) values
('LOGIN_HEADER_NAME','SecurityConstants',<HEADER_NAME>,'Y',null,'Header
name for login
separation','ofssuser',sysdate,'ofssuser',sysdate,'Y',1);
```

This query is used to provide entry for the custom header name.

```
Insert into DIGX_FW_CONFIG_ALL_B (PROP_ID, CATEGORY_ID, PROP_VALUE,
FACTORY_SHIPPED_FLAG, PROP_COMMENTS, SUMMARY_TEXT, CREATED_BY,
CREATION_DATE, LAST_UPDATED_BY, LAST_UPDATED_DATE, OBJECT_STATUS,
OBJECT_VERSION_NUMBER) values
(<HEADER_NAME>,'SecurityConstants',<HEADER_VALUE>,'Y',null,'login
separation header name and value
pair','ofssuser',sysdate,'ofssuser',sysdate,'Y',1);
```

This query is used for mapping the custom header name with its corresponding value.

```
Insert into DIGX FW CONFIG ALL B (PROP ID, CATEGORY ID, PROP VALUE,
FACTORY_SHIPPED_FLAG, PROP_COMMENTS, SUMMARY_TEXT, CREATED_BY,
CREATION_DATE, LAST_UPDATED_BY, LAST_UPDATED_DATE, OBJECT_STATUS,
OBJECT_VERSION_NUMBER) values
(<HEADER_VALUE>,'SecurityConstants',<ENTERPRISE_ROLE>,'Y',null,'Enables
login separation for given enterprise
role','ofssuser',sysdate,'ofssuser',sysdate,'Y',1);
```

This query is used for mapping the custom header value with the enterprise role for which the URL separation has be to achieve.

In the above queries, <HEADER_NAME> field denotes the custom header name, <HEADER_VALUE> denotes the custom header value, and <ENTERPRISE_ROLE> field denotes the enterprise role. These fields need to be replaced with own custom values before executing the queries.

OHS Configuration:
To support it OHS need to send an additional header to Weblogic server. To enable this Implementer need to configure a new port and create a virtual host where that custom header is added in the request. Sample snippet is below

```
Listen PORT_NO

<VirtualHost *:PORT_NO >
RequestHeader add <HEADER_NAME> "<HEADER_VALUE> "
<Location /digx>
SetHandler weblogic-handler
WebLogicCluster WEBLOGIC_HOST:WEBLOGIC_PORT
</Location>
</VirtualHost>
```

Home