Oracle Banking Digital Experience

OBDX UI Extension & Configurations Guide
Release 19.2.0.0.0

Part No. F25153-01

December 2019

ORACLE®

OBDX UI Extension & Configurations Guide

December 2019

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.

Introduction provides brief information on the overall functionality covered in the User Manual.

The subsequent chapters provide information on transactions covered in the User Manual.

Each transaction is explained in the following manner:

- Introduction to the transaction

- Screenshots of the transaction

- The images of screens used in this user manual are for illustrative purpose only, to provide improved understanding of the functionality; actual screens that appear in the application may vary based on selected browser, theme, and mobile devices.

- Procedure containing steps to complete the transaction- The mandatory and conditional fields of the transaction are explained in the procedure.

If a transaction contains multiple procedures, each procedure is explained. If some functionality is present in many transactions, this functionality is explained separately.

1.5 Related Information Sources

For more information on Oracle Banking Digital Experience Release 19.1.0.0.0, refer to the following documents:

- Oracle Banking Digital Experience Licensing Guide

- Oracle Banking Digital Experience Installation Manuals

2. **OBDX Component Extension**

This documentation will guide you on how to override existing OBDX components

Pre-requisites

- To override existing component you need following artifacts
    - ▪ ViewModel
    - ▪ Html
    - ▪ Model (optional)
    - ▪ Resource bundle
    - ▪ Partial (optional)
- Every extensible component must have module name and unique component name within its module.

Steps

- If you want to add new component place that component in **<CHANNEL_ROOT_PATH>/extensions/components**. It follow the same structure which is present in components folder. Same thing is applicable for the existing components. If you want to change anything then copy that component and place it in **extensions/components** folder with the same structure.

- If resource bundle needs to be changed for corresponding component then place related resource bundle in **<CHANNEL_ROOT_PATH>/extensions/resources** location. Structure remain same for **<CHANNEL_ROOT_PATH>/resources** and **<CHANNEL_ROOT_PATH>/extensions/resources** folder.

- You need to make entry of your component and partial in **<CHANNEL_ROOT_PATH>/extensions/extension.json file**

    This entry is in two parts.

    - ▪ Add entry of your component, in array, named as components "{moduleName}/{componentName}"
    - ▪ If your component requires partial then add its name, in array, named as partials. "folder-name/file-name"

    Sample extension.json

    ```
    {
            "components": ["loans/loan-calculations"],
            "partials": ["account-access/casa-account-access"]
    }
    ```

3.   Segment & JSON context Extension

For every application you can override following two properties

- User Type: Type of user

- Context: folder location from which json will be picked

To evaluate this two parameters there are respective functions in **<CHANNEL_ROOT_PATH>/extensions/override/extensions.js** file

evaluateUserType function,

>   Arguments (roles: [], defaultSegment : String)
>
>>   roles[] (array) will contain all the roles mapped to the user
>>
>>   defaultSegment (string) will contain system evaluated default segment
>
>   Return Value (String,),
>
>>   Should return string representing what is the segment for respective component (possible values: ANON | CORPADMIN | RETAIL | CORP | ADMIN)

Description,

If you want to change user type for your application, you can do so by implementing this function and return required user type for your application.

evaluateContext function,

>   *Arguments* (segment , roles)
>
>>   roles[] (array) will contain all the roles mapped to the user
>>
>>   segment (string) will contain evaluated user segment
>
>   *Return Value* (String)
>
>>   Should return string representing what is the context for respective component
>>
>>   (Possible values: index | retail | corporate | admin | corp-admin or any custom value)

Description,

This function will set context for your components to fetch json artifacts from correct folder. Return possible value and then json will be fetch from respective location. Using this you can specify menu for your context.

Note:

• Core functionality of Framework Elements like (header, dashboard, menu etc.) are not available for the modification. You can customize menu options.

• If any component is present in <CHANNEL_ROOT_PATH>/extensions/components will take precedence over the <CHANNEL_ROOT_PATH>/components.

• All components available under component folder are available for the extension

• If menu.json is to be changed and other are not changed, irrespective of the change, all files/folders within the base role, for example json/retail need to be copied for new role even if one file is changed. This is because when you change the context, using evaluateContext function above, the root for JSON lookup changes, that is why all the JSON files are then looked up from whatever value is being returned from the evaluateContext method. So, all the JSON files need to be present in new directory.

How to create/modify menu.json for new Context

Basic structure of your menu.json file should be as follows

```json
[
    {
      "name": "",
      "module": ""
    },
    {
      "name": "",
      "icon": "",
      "submenus": [
        {
          "name": "",
          "submenus": []
        }
      ]
    }

  ]
```

All your entries will go in array named as "default"

There are two types of entries

1. Single Menu Option

2. Nested Menu Option

Single Menu Option

Here you can specify following options

```
{
    "name": "",
    "module": "",
    "applicationType": "",
    "moduleURL": "",
    "type": ""
}
```

name	name of the component you want to load
module	module name of the component

Note: Also add component specific configurations wherever required. Please refer out of the box "menu.json" for each segment. For example below entries are required for some specific components only.

applicationType (optional)	it is component specific configuration
moduleURL (optional)	it is component specific configuration
type (optional)	it is component specific configuration

Nested Menu Option

This option you can use to group related menus together.

Following JSON denotes 1 menu group

```
{
  "name": "",
  "icon": "",
  "submenus": []
}
```

In above JSON

name is, key in resource bundle

icon is, name of icon from OBDX font. This icon will be the icon you want along with the name.

submenus [] will contain entries same as entry you will do for Single menu option

Sample menu.json

```json
[
  {
    "name": "PAYMENTS_TITLE",
    "icon": "icon-payments",
    "submenus": [
      {
        "name": "favorites",
        "module": "payments",
        "applicationType": "payments"
      },
      {
        "name": "SETUP",
        "submenus": [
          {
            "name": "manage-payees-billers",
            "module": "payee",
            "applicationType": "payments"
          }
        ]
      }
    ]
  },
  {
    "name": "about",
    "icon": "icon-information",
    "type": "MODAL"
  }
]
```

4. OBDX Validation Extension

All the validation available in the application are maintained in **<CHANNEL_ROOT_PATH>/framework/js/base-models/validations/obdx-locale.js.** Implementer can override and add new validations in the application without changing this file.

An extension hook is given at <CHANNEL_ROOT_PATH>extensions\override\obdx-locale.js

In this file Implementer can add or override validations.

For Example: If you need to change the pattern which validate Mobile Number. Add updated pattern in this file as below.

Figure 1 : Sample obdx-locale.js override

```
extensions > override > JS obdx-locale.js > ⬡ define() callback
 1   define([
 2       "ojL10n!resources/nls/obdx-locale"
 3   ], function (locale) {
 4       "use strict";
 5       return {
 6           MOBILE_NO: [{
 7               type: "regExp",
 8               options: {
 9                   pattern: "^(\\+\\d{1,3}[- ]?)?\\d{10}$",
10                   messageDetail: locale.messages.MOBILE_NO
11               }
12           }]
13       };
14   });
```

Apart from it all the data types used in UI side validation are maintained under <CHANNEL_ROOT_PATH>/resources/nls/data-types.js where all the regular expressions are defined.

```
define([], function() {
    "use strict";

    const DataTypeLocale = function() {
        return {
            root: {
                ALPHANUMERIC: "[a-zA-Z0-9]*",
                ALPHANUMERIC_WITH_SPACE: "[a-zA-Z0-9 ]*",
                NUMBERS: "[0-9]*",
                DECIMALS: "^[0-9]*\.?[0-9]+$",
                ALPHABETS: "[a-zA-Z]*",
                ALPHABETS_WITH_SPACE: "[a-zA-Z ]*",
                ALPHABETS_WITH_SOME_SPECIAL: "[a-zA-Z\-']*",
                LOWER_ALPHABETS: "[a-z]*",
                UPPER_ALPHABETS: "[A-Z]*",
                LOWER_ALPHABETS_WITH_SPACE: "[a-z ]*",
                UPPER_ALPHABETS_WITH_SPACE: "[A-Z ]*",
                ALPHANUMERIC_WITH_SPECIAL: "[a-zA-Z0-9 \%\&\:\,\)\(\.\_'\-\//;]*",
                ALPHANUMERIC_WITH_HYPHEN: "[a-zA-Z0-9\-]*",
                ALPHANUMERIC_WITH_SOME_SPECIAL: "[a-zA-Z0-9 \&\:\$\,\.\_\?]*",
                SWIFT: "[a-zA-Z0-9\- \+\:,\)\(\.'\?\/]*",
                SWIFT_X: "[A-Za-z0-9\/\\-\?\:\(\)\.\,\'\+\\s\r\n]*",
                SWIFT_Y: "[A-Za-z0-9\/\\-\?\:\(\)\.\,\'\+\\s\=\!\"\%\&\*\<\>\;]*",
                SWIFT_Z: "[A-Za-z0-9\/\\-\?\:\(\)\.\,\'\+\\s\_\=\!\"\%\&\*\<\>\;\@\#\{\r\n]*",
                ALPHANUMERIC_WITH_ALL_SPECIAL: "[a-zA-Z0-9\- \=\&\#\*\+\:,\)\(\.\!\$_\|'\`\?\[\\\]\/]*",
                SPACE_WITH_ALL_SPECIAL: "[!\"\#\$'\(\)\*\+\.\\\/\:\;\<\=\>\?\@\[\\\]\^\_\`\{\|\}\~\\\\-]*",
                FREE_TEXT: ".*",
                CUSTOM_TEXT: "[a-zA-Z0-9]*"
            },
            ar: true,
            fr: true,
            cs: false,
            sv: false,
            en: false,
            "en-us": false,
            el: false
        };
    };
```

During Implementation if implementer want to change data types regex, a similar file is present under <CHANNEL_ROOT_PATH>/extensions/resources/nls/data-types.js there they can modified the base values.

Along with this data-type.js also has language fall back so if implementer want different set of validation for other languages so they need to update data type regex for in that particular resource bundle. E.g. If implementer want different set of validation in Arabic (ar) so they have to
update regex either in <CHANNEL_ROOT_PATH>/resources/nls/data-types.js or <CHANNEL_ROOT_PATH>/extensions/resources/nls/ar/data-types.js

For guideline perspective they should put the entry in extension one and they have make sure language lookup is enabled for that particular language in <CHANNEL_ROOT_PATH>/extensions/resources/nls/data-types.js

Same behaviour is available for <CHANNEL_ROOT_PATH>/resources/nls/format.js where all the format are maintained.

For taxonomy validation Please refer **Oracle Banking Digital Experience Taxonomy Configuration Guide.**

5.   Calling Custom REST Services

In implementation if any new services are written by implementer it has been directed to change the context root for new REST to **digx/cz/v1**.

For supporting it from the UI, implementer has to pass **cz/v1** in the version field of the AJAX setting from his model.

For example see the snippet below:

```
const createApiGroup = function (payload, deferred) {
  const options = {
    url: "builder/apiGroup",
    version: "cz/v1",
    data: payload,
    success: function (data, status, jqXhr) {
      deferred.resolve(data, status, jqXhr);
    },
    error: function (data, status, jqXhr) {
      deferred.reject(data, status, jqXhr);
    }
  };

  baseService.add(options);
};
```

Figure 2 : Sample calling Custom REST Services

6. Internationalizing and Localizing Applications

Oracle Banking Digital Experience User Interface uses Oracle JET as it main library and it supports internationalization and globalization of web and hybrid mobile applications.

Refer following link for details

https://docs.oracle.com/en/middleware/jet/6/develop/internationalizing-and-localizing-applications.html

All resource bundle are available under **<CHANNEL_ROOT_PATH>/resources/nls** directory

In resource bundle file there are two parts i.e. root bundle and supported locale. In root bundle all the string are present which are required by the application. And in the locale part each locale entry is there with their lookup. If any particular locale entry is true and OBDX application is open in that locale then application is lookup for that locale resource bundle which should be present at **<CHANNEL_ROOT_PATH>/resources/nls/<locale>.**

For example if OBDX application's locale is fr then fr resource bundle must be present at **<CHANNEL_ROOT_PATH>/resources/nls/fr** location. But for locale specific lookup fr set as true in the supported locale section of main resource bundle.

Sample root resource bundle.

```
define([], function() {
    "use strict";

    return {
        root: {
            header: "About",
            productShortName: "OBDX",
            productName: "Oracle Banking Digital Experience",
            version: "Version",
            servicePack: "Service Pack",
            poweredBy: "Powered By",
            poweredByValue: "Oracle",
            copyright: "Copyright 1995-2017, Oracle and/or its affiliates. All right reserved.",
            build: "Build"
        },
        ar: true,
        fr: true,
        cs: true,
        sv: true,
        en: false,
        "en-us": false,
        el: true
    };

});
```

Sample Locale specific resource bundle:

```
define([], function() {
    "use strict";

    return {
        header: "About",
        productShortName: "OBDX",
        productName: "Oracle Banking Digital Experience",
        version: "Version",
        servicePack: "Service Pack",
        poweredBy: "Powered By",
        poweredByValue: "Oracle",
        copyright: "Copyright 1995-2017, Oracle and/or its affiliates. All right reserved.",
        build: "Build"
    };

});
```

Adding/Modify locale in OBDX resource bundle

Locale Configuration Tool is used to modify the locale setting of the resource bundles by a single command. So you have no need to go to each resource bundle and change locale setting.

By this tool user can perform

- Add New locale

- Enable or Disable any locale.

User need Node setup on their machine and run following command under **<CHANNEL_ROOT_PATH>/_build directory**

npm install

To use this utility you have to fire this command

 node locale-config.js <locale> <true | false>

For example **node locale-config fr true**

Above command will make **French** locale **true** in all resource bundles and if there is no locale code for French locale then it will add French locale with value as true in resource bundle.