OBDX UI Extensions Configurations Guide
Oracle Banking Digital Experience
Patchset Release 22.2.3.0.0

Part No. F72987-01

February 2024

ORACLE

OBDX UI Extensions Configurations Guide

February 2024

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 Patchset Release 22.2.3.0.0, refer to the following documents:

- 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 follows 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 framework component needs to be changed, place the new component code in **extensions/components** folder with same folder structure as in **framework/<api/core>** folder. For example, component present in framework/api should be available under extensions/components/api.
- If resource bundle needs to be changed for corresponding component, then place related resource bundle in **<CHANNEL_ROOT_PATH>/extensions/resources** location. Structure remains same for **<CHANNEL_ROOT_PATH>/resources** and **<CHANNEL_ROOT_PATH>/extensions/resources** folder.
- If you want to customize an existing flow, you need to make an entry of the flow in **<CHANNEL_ROOT_PATH>/extensions/extension.json** file against key “flows”. The customized flow should be present in the **<CHANNEL_ROOT_PATH>/extensions/flows** folder.

Please note all the components, flows, partials and resource bundles take priority to render. Means if file is present under the extension will be by default available for extension.

Home

3. Segment Extension

To evaluate segment there is respective function in **<CHANNEL_ROOT_PATH>/extensions/override/extensions.js** file

evaluateSegment 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.

init function,

Description,

This function takes no argument and returns nothing.

Implement this function if you want to perform any initialization.

getCurrencyFormattingOptions function,

Arguments (currency: String)

currency (string) will contain currency code

Return Value (Object),

Should return an object with the format given in the description.

Description,

If you want to override the default formatting for a currency, you can implement this function and return the object in the following format for that currency code passed.

```
{
    style: "currency",
    currency: currencyCode,
    maximumFractionDigits: maximumFractionDigits,
    minimumFractionDigits: minimumFractionDigits,
    useGrouping: true
}
```

You can refer the below link for more formatting options,

https://www.oracle.com/webfolder/technetwork/jet-420/jsdocs/oj.IntlNumberConverter.html#IntlNumberConverter

---

Note:

• 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 for new Context

Basic structure of your menu file should be as follows

```
[
    {
      "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 Redwood 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

```
[
    {
      "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"
  }
]
```

All menu in OBDX are user specific and available under <CHANNEL_PATH>framework/json/menu folder.

If implementor wants to override it so for it they have to make an entry in extension.json. Refer section 2.

Home

4. OBDX Validation Extension

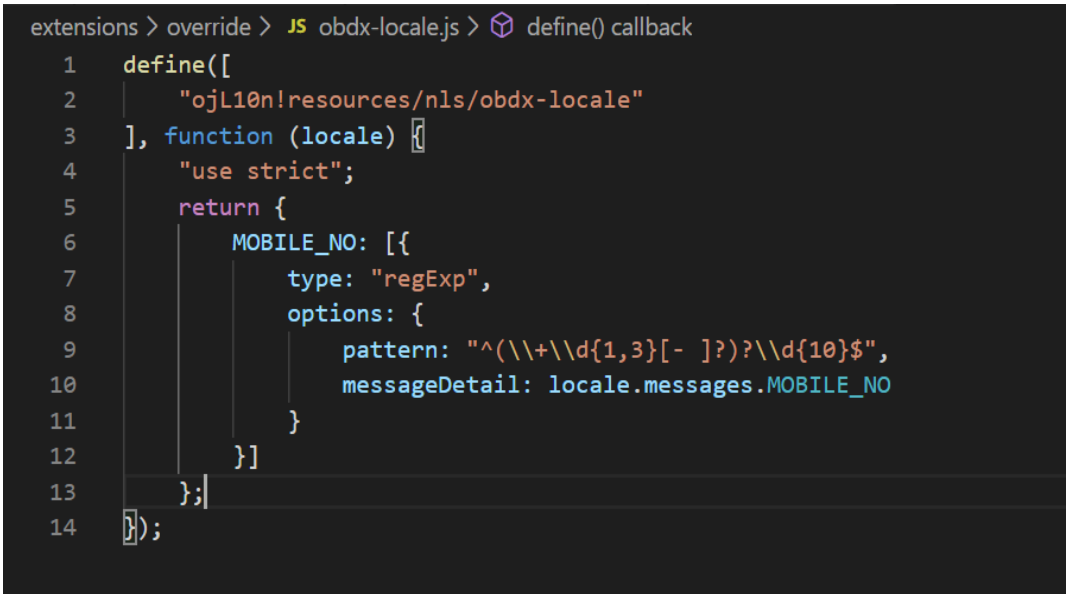
All the validation available in the application are maintained in **<CHANNEL_ROOT_PATH>/framework/js/base-models/validations/obdx-locale.js.** Implementer can override and add new validations in the application without changing this file.

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



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.**

Home

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:

Figure 2 : Sample calling Custom REST Services

```
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);
};
```

Home

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

We have Resource bundle loader which used during build for the resource bundles. All the supporting languages are mentioned there in supportedLang variable. This loader file is present under **<CHANNEL_ROOT_PATH>/**scripts/webpack/loaders/resource-bundle-loader.js

So if implementor wants to add or remove any language they can update that supportedLang variable which is an array.

After that implementor have to copy all resource bundle in **<CHANNEL_ROOT_PATH>reources/nls/<TARGET_LANG>**

After that run the GUI Build.

Home