

Oracle® Banking Digital Experience

OBDX UI Extensions Configurations Guide



Release 25.1.1.0.0
G43902-01
October 2025

ORACLE®

Copyright © 2015, 2025, Oracle and/or its affiliates.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software, software documentation, data (as defined in the Federal Acquisition Regulation), or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software," "commercial computer software documentation," or "limited rights data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government's use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle®, Java, MySQL, and NetSuite are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Inside are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Epyc, and the AMD logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

Contents

Preface

Purpose	i
Before you Begin	i
Pre-requisites	i
Audience	i
Documentation Accessibility	ii
Critical Patches	ii
Diversity and Inclusion	ii
Related Resources	ii
Conventions	ii
Screenshot Disclaimer	iii
Acronyms and Abbreviations	iii
Post-requisites	iii

1 OBDX Component Extension

2 Segment Extension

3 OBDX Validation Extension

4 Calling Custom REST Services

5 Internationalizing and Localizing Applications

Index

Preface

- [Purpose](#)
- [Before you Begin](#)
- [Pre-requisites](#)
- [Audience](#)
- [Documentation Accessibility](#)
- [Critical Patches](#)
- [Diversity and Inclusion](#)
- [Related Resources](#)
- [Conventions](#)
- [Screenshot Disclaimer](#)
- [Acronyms and Abbreviations](#)
- [Post-requisites](#)

Purpose

This guide is designed to help acquaint you with the Oracle Banking application. This guide provides answers to specific features and procedures that the user need to be aware of the module to function successfully.

Before you Begin

Kindly refer to our **Getting Started User Guide** for common elements, including Symbols and Icons, Conventions Definitions, and so forth.

Pre-requisites

Specify **User ID** and **Password**, and login to **Home** screen.

Audience

This document is intended for the following audience:

- Customers
- Partners

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>.

Access to Oracle Support

Oracle customer access to and use of Oracle support services will be pursuant to the terms and conditions specified in their Oracle order for the applicable services.

Critical Patches

Oracle advises customers to get all their security vulnerability information from the Oracle Critical Patch Update Advisory, which is available at [Critical Patches, Security Alerts and Bulletins](#). All critical patches should be applied in a timely manner to ensure effective security, as strongly recommended by [Oracle Software Security Assurance](#).

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

Related Resources

For more information on any related features, refer to the following documents:

- Oracle Banking Digital Experience Installation Manuals
- Oracle Banking Digital Experience Licensing Manuals

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
<i>italic</i>	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.

Convention	Meaning
monospace	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Screenshot Disclaimer

Personal information used in the interface or documents is dummy and does not exist in the real world. It is only for reference purposes; actual screens that appear in the application may vary based on selected browser, theme, and mobile devices.

Acronyms and Abbreviations

The list of the acronyms and abbreviations used in this guide are as follows:

Table 1 Acronyms and Abbreviations

Abbreviation	Description
OBDX	Oracle Banking Digital Experience

Post-requisites

After finishing all the requirements, please log out from the **Home** screen.

1

OBDX Component Extension

This topic provides information on **OBDX Component Extension**. This documentation will guide you on how to override existing OBDX components.

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.

2

Segment Extension

This topic provides information on **Segment Extension**. For every application you can override following two properties

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 **OBDX Component Extension**.

3

OBDX Validation Extension

This topic provides information on **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.

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])+$",
                ALPHABETS: "[a-zA-Z]*",
                ALPHABETS_WITH_SPACE: "[a-zA-Z ]*",
                ALPHABETS_WITH_SOME_SPECIAL: "[a-zA-Z\\-']*\"",
                LOWER_ALPHABETS: "[a-z]*",
                UPPER_ALPHABETS: "[A-Z]*",
                LOWER_ALPHABETS_WITH_SPACE: "[a-z ]*",
                UPPER_ALPHABETS_WITH_SPACE: "[A-Z ]*",
                ALPHANUMERIC_WITH_SPECIAL: "[a-zA-Z0-9 \\%&\\:;\\,\\)\\(\\,\\.\\_\\'\\-\\/\\/;]*\"",
                ALPHANUMERIC_WITH_HYPHEN: "[a-zA-Z0-9\\-]*",
                ALPHANUMERIC_WITH_SOME_SPECIAL: "[a-zA-Z0-9 \\&\\:;\\,\\.\\_\\?]*",
                SWIFT: "[a-zA-Z0-9\\- \\+\\:;\\,\\)\\(\\,\\.\\_\\'\\{\\}\\?\\V]*",
                SWIFT_X: "[A-Za-z0-9\\V\\V\\-\\?\\:;\\(\\)\\,\\.\\_\\'\\+\\|\\$\\r\\n]*",
                SWIFT_Y: "[A-Za-z0-9\\V\\V\\-\\?\\:;\\(\\)\\,\\.\\_\\'\\+\\|\\$|=\\!\\\"\\%\\&*\\<\\>\\;]*",
                SWIFT_Z: "[A-Za-z0-9\\V\\V\\-\\?\\:;\\(\\)\\,\\.\\_\\'\\+\\|\\$\\_|=\\!\\\"\\%\\&*\\<\\>\\;\\@\\#\\{\\}\\r\\n]*",
                ALPHANUMERIC_WITH_ALL_SPECIAL: "[a-zA-Z0-9\\- \\=\\&\\#\\*\\+\\:;\\,\\)\\(\\,\\.\\_\\'\\$\\_\\|\\'\\\"\\?\\[\\V\\V\\V\\V\\V]*",
                SPACE_WITH_ALL_SPECIAL: "[\\!\\\"\\#\\$\\'\\(\\)\\*\\+\\,\\.\\|\\V\\V\\:;\\<\\=\\>\\?\\@\\[\\V\\V\\V\\V\\V\\_\\'\\_\\'\\{\\}\\}\\~\\V\\V\\V\\V\\V]*",
                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**.

4

Calling Custom REST Services

This topic provides information on **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 4-1 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);  
};
```

5

Internationalizing and Localizing Applications

This topic provides information on **Internationalizing and Localizing Applications**.

Oracle Banking Digital Experience User Interface uses Oracle JET as its 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>resources/nls/<TARGET_LANG>**

After that run the GUI Build.

Index

C

Calling Custom REST Services, [1](#)

I

Internationalizing and Localizing Applications, [1](#)

O

OBDX Component Extension, [1](#)

OBDX Validation Extension, [1](#)

S

Segment Extension, [1](#)