

Oracle Banking Extensibility Workbench

User Manual

Release 14.7.5.0.0

October 2024





Oracle Banking Extensibility Workbench
User Manual
October 2024
Version 14.7.5.0.0

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 <https://www.oracle.com/industries/financial-services/index.html>

Copyright © 2024, Oracle and/or its affiliates. All rights reserved.

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

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are “commercial computer software” pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. 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 failsafe, 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.

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.

This software or hardware and documentation may provide access to or information on 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. 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.

Contents

1 Preface	6
1.1 Introduction	6
1.2 Audience	6
1.3 Document Accessibility	6
1.4 Related Documents	6
2 Welcome to Oracle Banking Extensibility Workbench	6
2.1 Introduction	6
2.2 OBX and Base artifacts compatibility	7
2.3 Setting up OBX for first time use	8
2.4 OBX Maintenance	9
2.5 OBX UI	10
2.5.1 Entity Details	11
2.5.2 Field Details	11
2.5.3 Child Entity Details	13
2.5.4 Relationship Details	13
3 Service Extensions	15
3.1 Simple Sub Domain Service	16
3.2 Maintenance sub domain service	19
3.3 Data/Resource Segment sub domain service	21
3.3.1 RSOV1	21
3.3.2 RSOV2 DS	23
3.3.3 Workflow DS	24
3.4 Simple Publisher/Subscriber Event Service	27
3.5 Batch Service	29
3.6 Custom Validation Service	31
3.7 Steps to adopt Multi Entity in existing service	32
3.8 Service Extensibility	34
4 UI Extensions – Web Component	37

4.1	Component Server.....	39
4.2	Simple Standalone.....	40
4.3	Virtual Page	43
4.4	Maintenance Detail and Summary	46
4.5	Data Segment.....	48
4.6	Dashboard Widget.....	50
4.7	Running Component after Generation.....	53
4.8	Creating final Extended Component war for Deployment	54
4.9	Creating final Extended Component war for Deployment.	55
4.10	Understanding DB Scripts for Web Components	57
5	Modification of Base Web Component.....	58
5.1	Steps for Modification of Base Component	58
5.2	Process Workbench.....	59
5.3	OBX Update Command	65
5.3.1	Service Update	65
5.3.2	UI Update.....	66
5.4	In-Scope DS	67
5.5	OBX Release Command	67
6	Extending Product Data Segments with Additional Fields	68
6.1	Additional Fields Maintenance.....	68
6.2	Populating Data in Corresponding Fields From UI	75
6.3	Fetching the Saved Values.....	77
7	Action URL and Static Tag Maintenance	79
7.1	Action URL Maintenance	79
7.2	Static Tag Maintenance	79
8	Extensibility Use Cases for OBARN Servicing.....	79
8.1	New Transaction Screen – 1499 (Exact Clone of 1401)	79
8.2	Exact Clone with Additional Fields Using Common Code.....	81
8.3	Exact Clone with Additional Fields Using Extensible Code.....	86
8.4	Jar Deployment in Weblogic:.....	87
9	Extensibility Use Cases for OBX	91

9.1	New Transaction screen – 1499 (Clone of 1401)	91
9.2	New Data Segment in Existing 1401 Screen.....	93
9.3	HTML Changes.....	94
9.4	JS Changes	95
9.5	JSON Changes.....	97
9.6	Model Changes.....	98
9.7	Database Changes	98
9.8	Service Component	98
9.9	New Field in Existing Base Data Segment	101
9.10	HTML Changes (Extended Components)	103
9.11	HTML Changes (Base Component)	103
9.12	JS Changes (Base Component)	104
9.13	JS Changes (Extended Component)	104
9.14	JSON Changes (Extended Component)	105
9.15	JSON Changes (Base Component)	105
9.16	DB Changes	106
9.17	Add New Columns in Base Component Table	107
9.18	Steps for adding extra column in task grid	109
9.19	Steps to use Additional Buttons provision in Task Screen	109
9.20	Steps to create common-extended folder for extending configJSON.js file	110
9.21	Customizing Existing LOV Fetch Result.....	111
9.22	Steps for adding Pre/post methods in extended components.....	111
9.23	ENDPOINT Overrides.....	112
9.24	Steps to create util-extended folder	113
9.25	Dynamic Data Configuration (DDC)	114
9.26	Task Screen Custom Config.....	117
10	Reference and Feedback.....	121
10.1	Reference	121
10.2	Documentation Accessibility	121
10.3	Feedback and Support	121

1 Preface

1.1 Introduction

This user guide would help you to understand the functioning of the Oracle Banking Extensibility Workbench – OBX and the types of extensions it provides. It provides the steps required to be followed for implementing the extensibility to the Base product. It is assumed that all the prior setup is already done related with Base product/ Kernel. In this document it is also assumed that installation will be done on Windows 10 operating system with minimum 8GB Ram and available/free space of 5GB.

1.2 Audience

This document is intended for the teams and developers who are responsible for creating extensions like services and web components for products which are developed using Oracle Banking Microservices Architecture.

1.3 Document 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_

1.4 Related Documents

For more information, refer to the following documents:

- Oracle Banking Extensibility Workbench Installation Guide
- Oracle Banking Extensibility Workbench Release Notes

2 Welcome to Oracle Banking Extensibility Workbench

Welcome to the Oracle Banking Extensibility Workbench (OBX) user manual. It provides the complete solution to create extensions for products based and developed on Oracle Banking Microservices architecture (OBMA). It helps in generating the services and UI web components artifacts. This guide is designed to help you create all these types of service and UI artifacts. It also has complete life cycle management incorporated for all the extensions generated from tool.

2.1 Introduction

Oracle Banking Extensibility Workbench (OBX) is a combination of GUI and command line tool, intended to create different type of extensions for Oracle Banking Micro services Architecture.

OBX support generation of following types of Extensions :

- Service Extensions
 - Simple sub domain service
 - Maintenance sub domain service
 - Data/Resource Segment sub domain service
 - Simple Publisher/Subscriber Event Service

- Custom Validation Service
- UI Extensions – Web Component
 - Simple Standalone
 - Virtual Page
 - Maintenance Detail and Summary
 - Data Segment
 - Dashboard Widget
- Modification of Base Web Component
 - Additions of Fields on Existing component
 - Hiding fields from screen
 - Defaulting values on screen ○ Disable field
 - Making Non-mandatory field

2.2 OBX and Base artifacts compatibility

OJET version compatibility:

Ensure that the OJET version of app shell used by the implementation team, aligns with the OJET version present in the OBX tool.

Note: As part of OJET upgrade some older libraries may not be supported. If consulting / implementation team is using any of the unsupported libraries for their customizations, compatibility issues may arise if the app-shell version they are using doesn't include those OJET libraries.

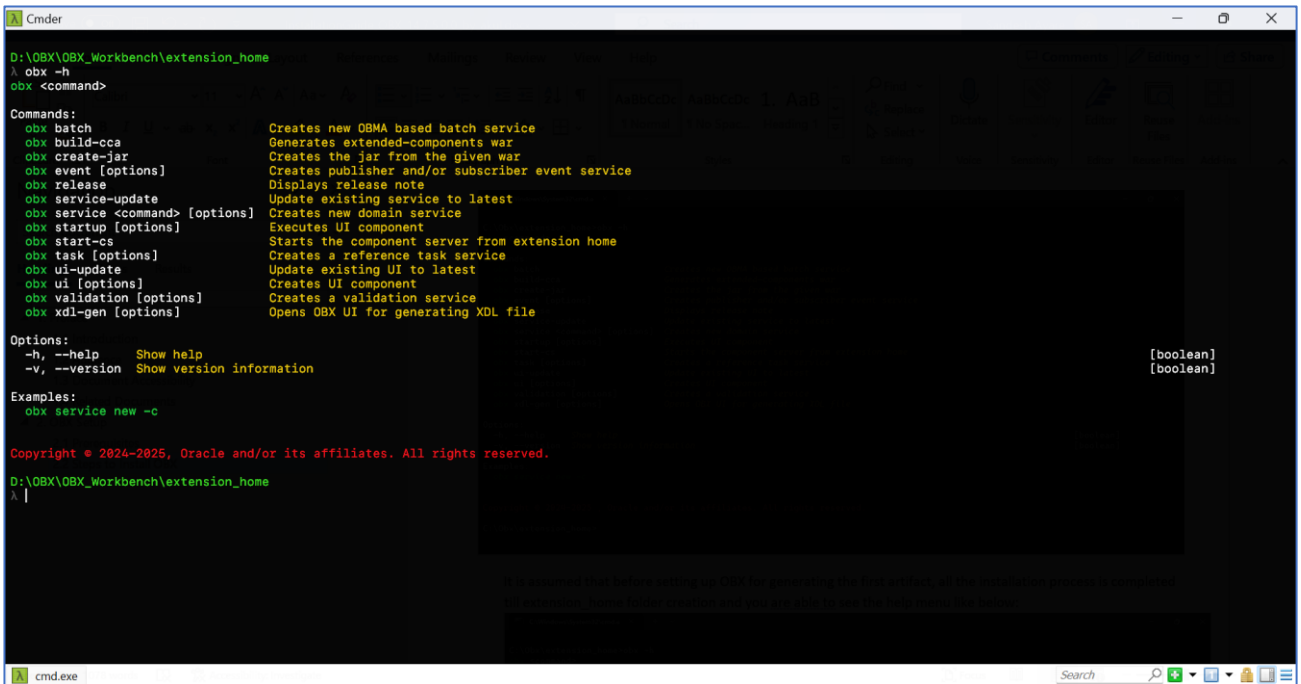
All the UI customizations / extensions are bundled into `extended-components` war which ultimately refer to the app-shell OJET libraries only.

Please find the compatibility matrix of app-shell OJET versions and OBX OJET versions below.

OBX version	OJET version
14.7.0.0.0	Appshell version xxxx (has 13.0.0 OJET version)
14.7.5.0.0	Appshell version 9.5.0 (has 15.1.8 OJET version)

2.3 Setting up OBX for first time use

It is assumed that before setting up OBX for generating the first artifact, all the installation process is completed till `extension_home` folder creation and you can see the help menu like below:



```
D:\OBX\OBX_Workbench\extension_home
λ obx -h
obx <command>

Commands:
obx batch                Creates new OBMA based batch service
obx build-cca             Generates extended-components war
obx create-jar            Creates the jar from the given war
obx event [options]       Creates publisher and/or subscriber event service
obx release              Displays release note
obx service-update        Update existing service to latest
obx service <command> [options] Creates new domain service
obx startup [options]     Executes UI component
obx start-cs              Starts the component server from extension home
obx task [options]        Creates a reference task service
obx ui-update             Update existing UI to latest
obx ui [options]          Creates UI component
obx validation [options]  Creates a validation service
obx xdl-gen [options]     Opens OBX UI for generating XDL file

Options:
-h, --help              Show help
-v, --version            Show version information

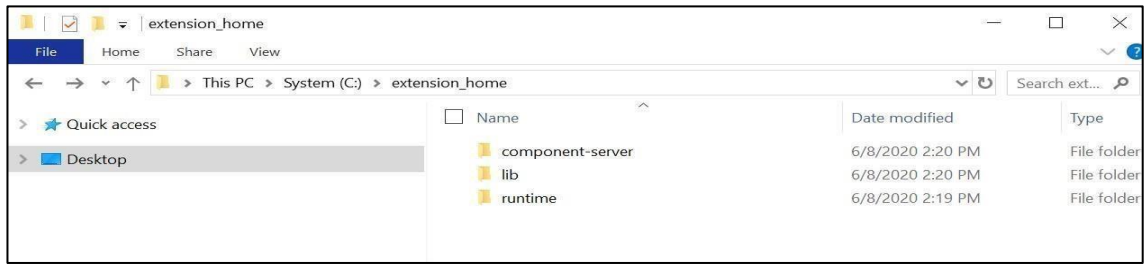
Examples:
obx service new -c

Copyright © 2024-2025, Oracle and/or its affiliates. All rights reserved.

D:\OBX\OBX_Workbench\extension_home
λ |
```

Once that is done, we will proceed to next step which is setting up libraries and components from base product. Please follow the below process to setup libraries and components:

- Create a folder **component-server** inside **extension_home** directory.
- Use 7zip or other similar tool to extract **app-shell-9.5.0.war** from base product to copy the **common** & **js** folders and put it inside the **component-server** folder.
- Navigate inside the **js** folder and copy the **components** folders and place it in the **component-server** folder.
- Create a folder **lib** inside **extension_home** directory.
- Again, using 7zip or other similar tool open any service war like **cmc-datasegment-services-9.5.0.war**, navigate inside **WEB-INF\lib** folder and copy all the jars and put it inside the **lib** folder of **extension_home**.
- Create a folder **runtime** inside **extension_home** directory.
- From the **gradle** folder which comes inside the **obx.zip**, navigate inside the **lib** folder and copy **extra_jars** folder which consist of compile time dependencies for services, and paste it inside **runtime** folder **extension_home**.
- After all the above process **extension_home** folder looks like below:



- Once all the above process is done, we can now generate the artifacts.

2.4 OBX Maintenance

Before generating the artifact, please verify the below items from the base installation.

- In the **plato-ui-config** schema, verify if the table '**PRODUCT_EXTENDED_LEDGER**' is present or not. If not available, please execute the below script:

```
-----
-- DDL for Table PRODUCT_EXTENDED_LEDGER
-----
```

```
CREATE TABLE "PRODUCT_EXTENDED_LEDGER" ("ID" VARCHAR2(20), "CCA_NAME"
VARCHAR2(100), "CCA_TYPE" VARCHAR2(20), "PARENT_CCA_NAME" VARCHAR2(100),
"PRODUCT_NAME" VARCHAR2(100))
-----
```

```
-- Constraints for Table PRODUCT_EXTENDED_LEDGER
-----
```

```
ALTER TABLE "PRODUCT_EXTENDED_LEDGER" ADD CONSTRAINT
"PRODUCT_EXTENDED_LEDGER_PK" PRIMARY KEY ("ID")
ALTER TABLE "PRODUCT_EXTENDED_LEDGER" MODIFY ("CCA_NAME" NOT NULL
ENABLE)
ALTER TABLE "PRODUCT_EXTENDED_LEDGER" MODIFY ("ID" NOT NULL ENABLE)
ALTER TABLE "PRODUCT_EXTENDED_LEDGER" ADD CONSTRAINT
"UNIQUES_CCA_NAME" UNIQUE ("CCA_NAME")
```

- Please maintain the product name '**OBX**' in the table '**SMS_TM_APPLICATION**' inside SMS schema.
- Please grant user '**OBX**' application access through '**SMS_TM_USER_APPLICATION**' or preferred use the UI.

Create User

User Role Branches

<input type="checkbox"/>	Branch Code	Role Code	Role Description
No data to display.			

Page 1 (0 of 0 items) |< 1 >|

Customer Access Groups

<input type="checkbox"/>	Customer Access Group	Customer Access Description
No data to display.		

Page 1 (0 of 0 items) |< 1 >|

Cancel Save

2.5 OBX UI

After setting up the OBX, we can now proceed to generate the XDL (OBX Domain Language) file which will be used by the OBX engine to further generate the service and UI artifacts. To start OBX UI we need to navigate to extension_home folder from console emulator (cmdr) and use the command **obx xdl-gen**.

This command will automatically open a new tab in cmdr with OBX UI running at local port 8080

(<https://localhost:8080>)

Note: If you have any running on port number 8080, you may need to stop that to make obx ui up and running.

```

Cmdr

OBX

ORACLE BANKING EXTENSIBILITY WORKBENCH October 2024 (14.7.5.0.0)
Copyright © 2024-2025, Oracle and/or its affiliates. All rights reserved.

> Starting OBX UI |
D:\OBX\OBX_Workbench\extension_home\obxui>npm link fs-extra cucumber-html-reporter cucumber selenium-webdriver express http-proxy-middleware gulp gulp-connect gulp-open gulp
-sass karma karma-chrome-launcher karma-coverage requirejs karma-jasmine karma-junit-reporter karma-requirejs karma-sonarqube-unit-reporter morgan portscanner serve-index 1
>nul
OBX UI is running at port:8080, Please generate xdl file before proceeding
? Did you generate the xdl file? (Y/n) |
  
```

Please open browser once OBX UI is up and running and navigate to <http://localhost:8080>.

The screenshot shows the Banking Extensibility Workbench interface. It has a dark header bar with navigation icons and the URL <https://localhost:8080>. The main content area is titled "Banking Extensibility Workbench" and contains four sections:

- Entity Details:** Includes an "Entity Name" input field with a placeholder "only Alphanumeric characters" and a "Required" label. To the right is a dashed box with the text "Please drop XDL file or Click here to upload".
- Field Details:** Features a table with columns "Field Name", "Field Type", "Default Value", and "Field Size". The table is currently empty, showing "No data to display." Below the table are "Add", "Modify", and "Delete" buttons.
- Child Entity Details:** Includes an "Add Child Entity" button and a message "No items to display."
- Relationship Details:** Contains a "Has Relationship" section with two radio buttons: "True" and "False". The "False" radio button is selected.

- Entity Details
- Field Details
- Child Entity Details
- Relationship Details

2.5.1 Entity Details

In this section you will capture the entity name. As the Domain Entity pattern "an object is primarily defined by its identity is called an Entity."

This is a close-up of the "Entity Details" section. It shows the "Entity Name" input field with the placeholder "only Alphanumeric characters" and a "Required" label. To the right is a dashed box with the text "Please drop XDL file or Click here to upload".

2.5.2 Field Details

For the main entity you need to define the fields in this section. For doing that click on the Add button and provide the field details.

This is a close-up of the "Field Details" section. It shows a table with columns "Field Name", "Field Type", "Default Value", and "Field Size". The table is currently empty, showing "No data to display." Below the table are "Add", "Modify", and "Delete" buttons.

Following are the different types of field types supported in OBX:

String	This is inbuild field type of OBX, it gets translated to varchar for sql scripts, string type in java files and normal text field in UI component
Integer	This is inbuild field type of OBX, it gets translated to number for sql scripts, integer type in java files and normal text field in UI component
Float	This is inbuild field type of OBX, it gets translated to number for sql scripts, float type in java files and normal text field in UI component
LOV	This field type is inherited from the base product and has its own configuration as below

Field Details

Field Name

Field Datatype

Default Value

Size

only letters

LOV

default value for field

1

Required

Mandatory

☒ True
☐ False

LOV Configuration

Id

Title

End Point

lov id

lov title

end point

RequiredRequiredRequired

Add

Here, ID is the specific id given to this LOV component, Title is displayed on the LOV dialog box and Endpoint is the service endpoint which this field connects to for fetching values.

- **Date:** This field is also inherited from the base product and add date component on the screen.
- **Amount:** This field is also inherited from the base product and add the amount field on the screen. This field also captures currency along with the amount.
- **Combo box:** This field is taken from Ojet Cookbook and OBX UI provides configurations to needed for this component like value and label.

Field Details

Field Name

only letters

Field Datatype

Combobox

Default Value

default value for field

Size

1

Required

Mandatory

☒ True
☐ False

Combobox Configuration

Value

No data to display.

Label

Value

Label

Add

- **Checkbox:** This field type is also taken from Ojet Cookbook and OBX UI provides configurations to needed for this component like value and label.
- **Toggle Button:** This field type is taken from Ojet Cookbook.
- **Text Area:** This field type is taken from Ojet Cookbook.

2.5.3 Child Entity Details

Use this block for adding the child entities. Once clicked on the Add Child Entity Button, it will open a dialog box where we can enter the child entity name. Once clicked ok it will add a child block below with its details:

Child Entity Details

Add Child Entity

No items to display.

Please add the child entity field details in a similar way like we added for main entity.

2.5.4 Relationship Details

Once all the entity details are added we can define relationship among them. Use this block to define the relationship. Currently OBX supports two types of relationships:

- One to Many

Relationship Details

Has Relationship

☒ True
☐ False

Relationship Type

One-Many

One-Many
One-Many-Many

Save XDL

Copyright © 2023, 2024 Oracle and/or its affiliates All rights reserved.

Once all of the above Entity, Field Details & Relationship is created click on the Save XDL button and it will save the xdl file on machine.

Note: Its recommended to put the xdl file under the same **extension_home** folder and give it proper name (generally main entity name).

The final XDL file looks like below:

```

1  entity customer {
2      customerId String required size(10) default(CUST100)
3      mobileNumber int size(10) default(1234567890)
4      rateOfInterest float size(15) default(8.1)
5      currency String size(3) default(USD) lov {id(currencyCode) title
(Currency LOV) endpoint(CORE.CURRENCY)}
6      accountOpenDate String size(15) default() date
7      balance String required size(20) default() amount
8      isExistingCustomer String size(3) default(no) dropdown [{no:No}
{yes:Yes}]
9      address String size(25) default(India) text-area
10 }
11 entity address {
12     address String required size(25) default() text-area
13     city String required size(15) default()
14     state String size(25) default()
15     country String required size(20) default()
16     pincode int required size(6) default()
17 }
18 relationship OneToMany {
19     customer to address
20 }

```

Once XDL file is generated you may come back to cmdr main tab where it is waiting for the input. You may proceed creating next set of artifacts which are described in next sections.

```
Cmder

OBX

ORACLE BANKING EXTENSIBILITY WORKBENCH October 2024 (14.7.5.0.0)

Copyright © 2024-2025, Oracle and/or its affiliates. All rights reserved.

> Starting OBX UI |
D:\OBX\OBX_Workbench\extension_home\obxui>npm link fs-extra cucumber-html-reporter cucumber selenium-webdriver express http-proxy-middleware gulp gulp-connect gulp-open gulp
-sass karma karma-chrome-launcher karma-coverage requirejs karma-jasmine karma-junit-reporter karma-requirejs karma-sonarqube-unit-reporter morgan portscanner serve-index 1
>nul
OBX UI is running at port:8080, Please generate xdl file before proceeding
? Did you generate the xdl file? Yes
? Do you want to create: (Use arrow keys)
> UI component
Domain service with optional UI component
Data-segment service with optional UI component
Maintenance domain service with optional detail & summary UI components
Obscf service with optional UI component      force C:\Users\saayaz\yo-rc-global.json
```

3 Service Extensions

Using OBX we can create multiple types of service extensions. This services extension has complete infrastructure needed to build to service. Also, the source folder generated out the box from OBX follows the package structure which is adopted and used by base/kernel teams to keep it in sync.

Note: There are 2 ways to generate the service artifact:

- Select the category immediately after generating the XDL file and proceed.

```
OBX UI is running at port:8080, Please generate xdl file before proceeding
? Did you generate the xdl file? Yes
? Do you want to create: (Use arrow keys)
? Do you want to create:
> UI component
Domain service with optional UI component
Data-segment service with optional UI component
Maintenance domain service with optional detail & summary UI components
Obscf service with optional UI component
```

- Use the service specific command to generate different types.

```

C:\Users\Kartik\Documents\OBX\OBX Final\OBX-14.7.0.0.0\extension_home
λ obx service -h
obx service <command> [options]

Creates new domain service

Commands:
  obx service ds [options]      Creates a new OBMA based data-segment service
  obx service mn [options]      Creates new OBMA based maintenance service
  obx service new [options]     Creates new OBMA based simple service
  obx service obscf [options]   Creates new OBMA based obscf service
  obx service rsov2 [options]   Creates a new RSOv2 based data-segment service
  obx service wfds [options]    Creates a new OBMA based Workflow data-segment service

Options:
  -h, --help      Show help
  -v, --version   Show version information

```

Both above ways will generate the same artifacts.

3.1 Simple Sub Domain Service

This is one of the primary use cases in OBX, to generate the simple sub-domain service. To generate it please follow the below steps:

- Navigate to same extension_home folder using cmd.
- Use the command **obx service new -c**.

```
Cmder

D:\OBX\OBX_Workbench\extension_home
λ obx service new -c

Field Details

OBX

ORACLE BANKING EXTENSIBILITY WORKBENCH October 2024 (14.7.5.0.0)

Copyright © 2024-2025, Oracle and/or its affiliates. All rights reserved.

? Select the product family:
> Oracle Banking Extensibility Workbench
  Oracle Banking Branch
  Oracle FLExcube Onboarding
  Oracle Banking Virtual Account Management
  Oracle Banking Trade Finance Process Management
  Oracle Banking Credit Facility Process Management
  Oracle Banking Corporate Lending Process Management
(Move up and down to reveal more choices)
```

- Once this command is fired, this will take you to next section where it will prompt other set of questions. Answer them accordingly to your setup and requirement.

```
Cmder

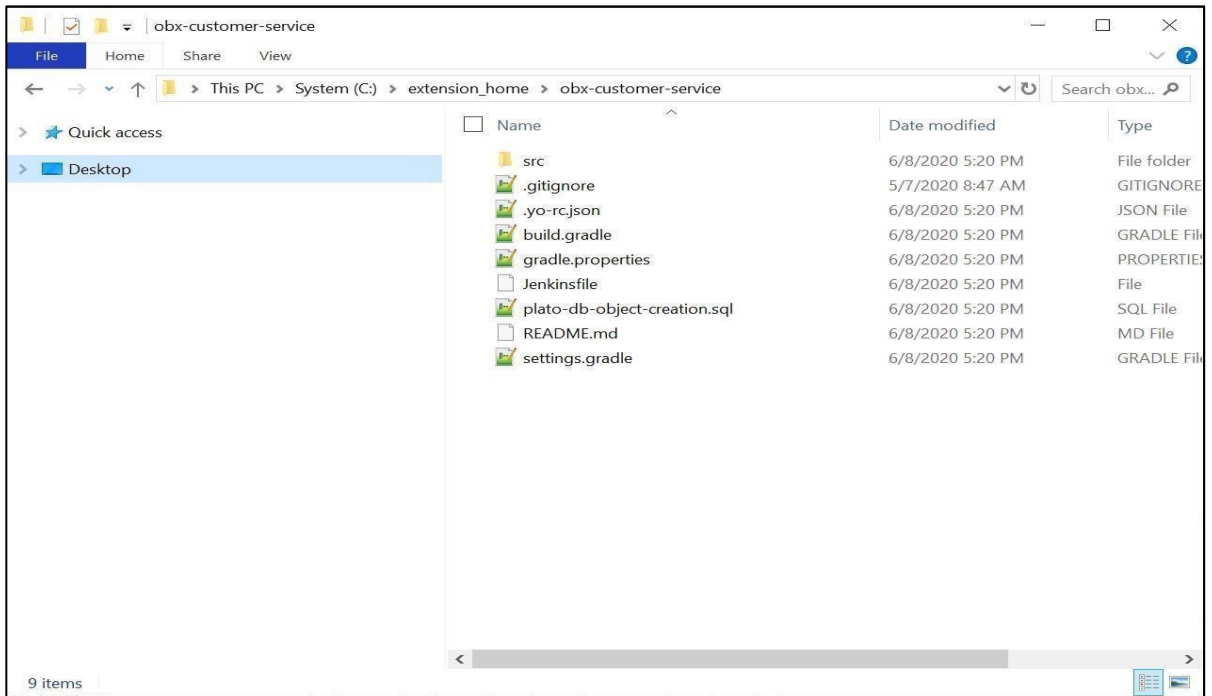
OBX

ORACLE BANKING EXTENSIBILITY WORKBENCH October 2024 (14.7.5.0.0)

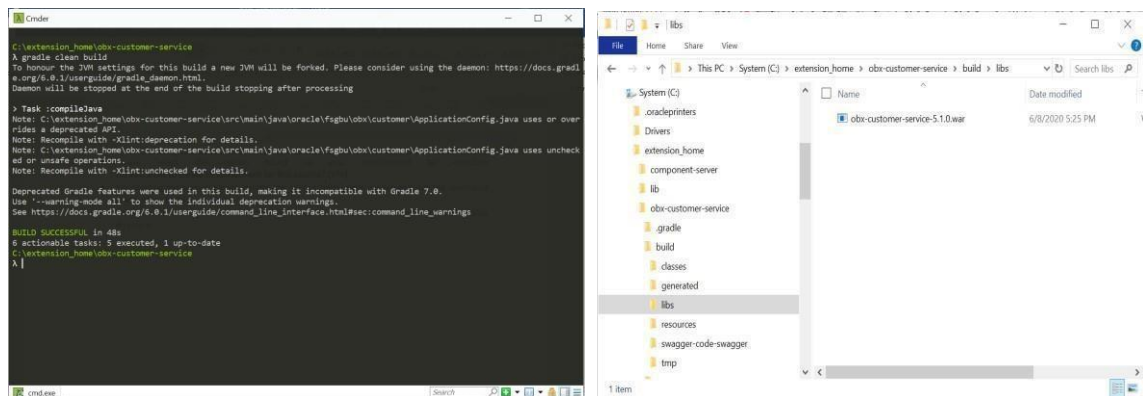
Copyright © 2024-2025, Oracle and/or its affiliates. All rights reserved.

? Select the product family: Oracle Banking Extensibility Workbench
? Enter name of service (I'll add -service to it): customer
? Select service tenant type: Single Tenant
? Enter name of Infra (OBMA) data source (I'll add prefix jdbc/ to it): OBMA
? Enter name of Security data source (I'll add prefix jdbc/ to it): PLATO_SECURITY
? Enter name of this service data source (I'll add prefix jdbc/ to it): ENTITY
? Enter product release version: 14.7.5.0.0
? Enter the absolute path of xdl file: D:\OBX\xdl-files\customer.xdl
```

- Once all the questions are answered and path of XDL is given, it will generate a folder inside the **extension_home** folder.



- Please select the option based on your requirement for question **Do you want to create UI component for this service? (Y/n).**
- For building the service please go into the service folder from cmd and run the command **gradle clean build.**
- This will build the service and we can find the war of the service getting created inside the build/lib's directory.



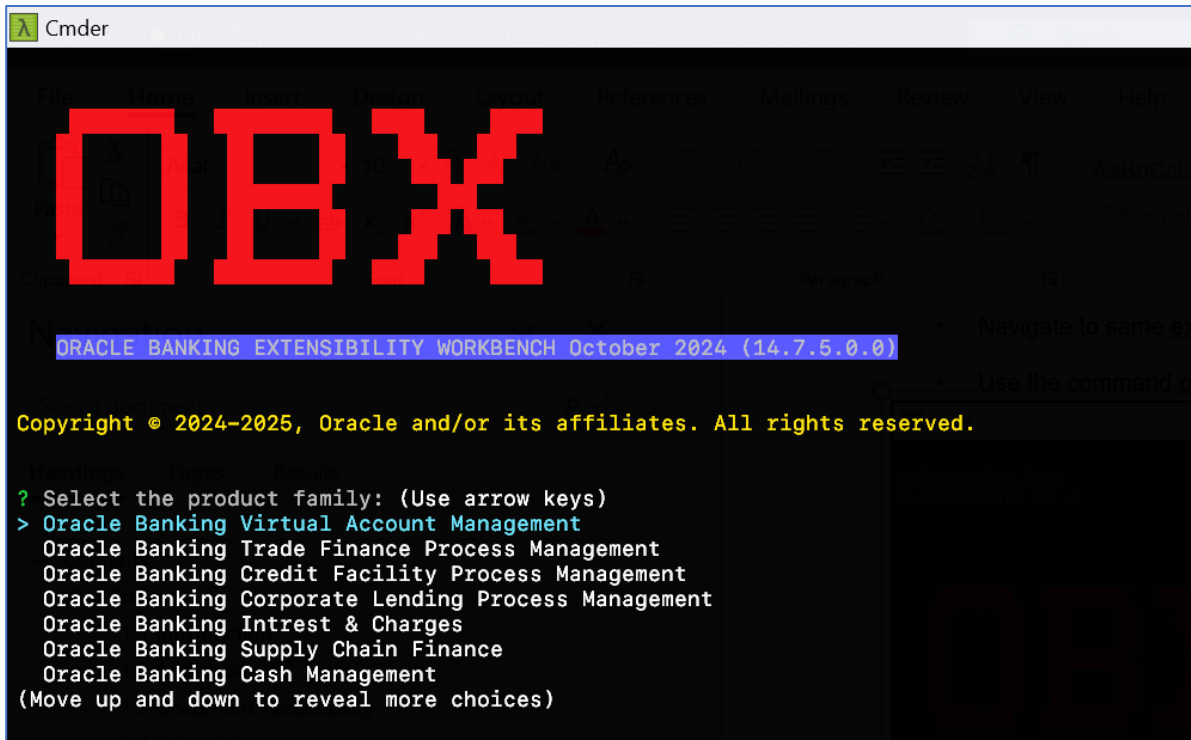
- Use this service and deploy it in your environment.
- Notes: DB scripts for the service will be generated inside the folder **\extension_home\obxcustomerservice\src\main\resources\ld**
- Please Compile the Entity script in the entity schema created for extensions only.
- Service created as part of extension should be deployed in separate domain and should not be mixed or co-deployed with any other product specific services.

- Before compiling **CONFIG_SCRIPT.sql** in verify the entries manually and change it according to your setup.
- Also, please verify **PLATO_TABLE_SCRIPT.sql** before executing it in the schema it may contain some dummy values.

3.2 Maintenance sub domain service

This section describes the process to generate the maintenance type of service. Maintenance service generally has concept of main and worktable. This allows enables functionality where all the Authorized records goes to main table and all the unauthorized records goes to worktable. Also, with this type of service we attach audit details to payload. To generate it please follow the below steps:

- Navigate to same **extension_home** folder using cmdr.
- Use the command **obx service mn -c**.



```

Cmder
File Edit View Options Database Window References Mailings Review View Help
OBX
ORACLE BANKING EXTENSIBILITY WORKBENCH October 2024 (14.7.5.0.0)
Copyright © 2024-2025, Oracle and/or its affiliates. All rights reserved.

? Select the product family: (Use arrow keys)
> Oracle Banking Virtual Account Management
  Oracle Banking Trade Finance Process Management
  Oracle Banking Credit Facility Process Management
  Oracle Banking Corporate Lending Process Management
  Oracle Banking Intrest & Charges
  Oracle Banking Supply Chain Finance
  Oracle Banking Cash Management
(Move up and down to reveal more choices)

```

- Once this command is fired, this will take you to next section where it will prompt other set of questions. Answer them accordingly to your setup and requirement.

```

Cmdr

OBX

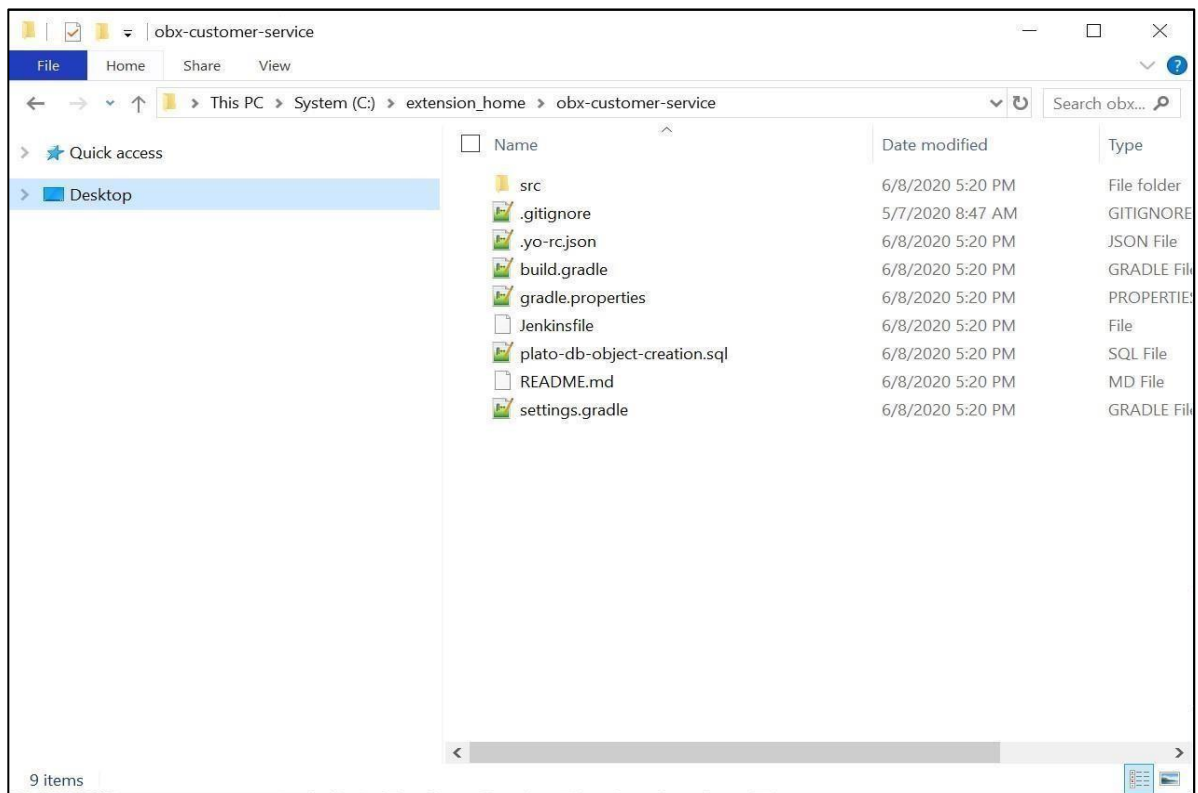
ORACLE BANKING EXTENSIBILITY WORKBENCH October 2024 (14.7.5.0.0)

Copyright © 2024-2025, Oracle and/or its affiliates. All rights reserved.

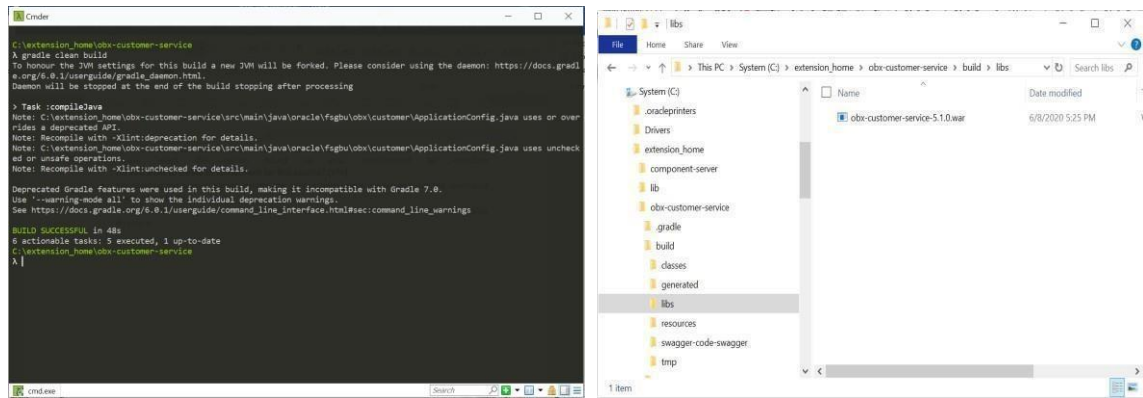
? Select the product family: Oracle Banking Extensibility Workbench
? Enter name of service (I'll add -service to it): customer
? Select service tenant type: Single Tenant
? Enter name of Infra (OBMA) data source (I'll add prefix jdbc/ to it): OBMA
? Enter name of Security data source (I'll add prefix jdbc/ to it): PLATO_SECURITY
? Enter name of this service data source (I'll add prefix jdbc/ to it): ENTITY
? Enter product release version: 14.7.5.0.0
? Enter the absolute path of xdl file: D:\OBX\xdl-files\customer.xdl

```

- Once all the questions are answered and path of XDL is given, it will generate a folder inside the **extension_home** folder.



- Please select the option based on your requirement for question: **Do you want to create a Maintenance and Summary Components for this service? (Y/n)**
- For building the service, please go into the service folder from cmdr and run the command **gradle clean build**.
- This will build the service and we can find the war of the service getting created inside the build/libs directory.



- Use this service and deploy it in your environment.

Notes:

- DB scripts for the service will be generated inside the folder:
\extension_home\obxcustomerservice\src\main\resources\db
- Please Compile the Entity script in the entity schema created for extensions only.
- Service created as part of extension should be deployed in separate domain and should not be mixed or co-deployed with any other product specific services.
- Here SMS (Security Management System) scripts are also generated.
\extension_home\obxcustomer-service\src\main\resources\db\sms
- Execute the SMS script in sms schema, here we only generate the functional activity of service. Assigning to proper role should be done according to the steps mentioned in base application.

3.3 Data/Resource Segment sub domain service

3.3.1 RSOV1

This section describes the process to generate the data/resource segment type of maintenance service. Here we can generate Master Type of data segment or child type of data segment.

Master Type: This case is used when user wants to generate the complete flow from scratch. It will generate the new screen class code for the data segments.

Child Type: This is primarily used when user wants to attach a single data-segment in the existing flow/process. Generally, this existing flow/process is available in the base product. We use the same screen class code from base and attach our data segment to it. To generate it please follow the below steps:

- Navigate to same extension_home folder using cmdr.
- Use the command **obx service ds -c**

```
Cmder

D:\OBX\OBX_Workbench\extension_home
λ obx service ds -c

OBX

ORACLE BANKING EXTENSIBILITY WORKBENCH October 2024 (14.7.5.0.0)

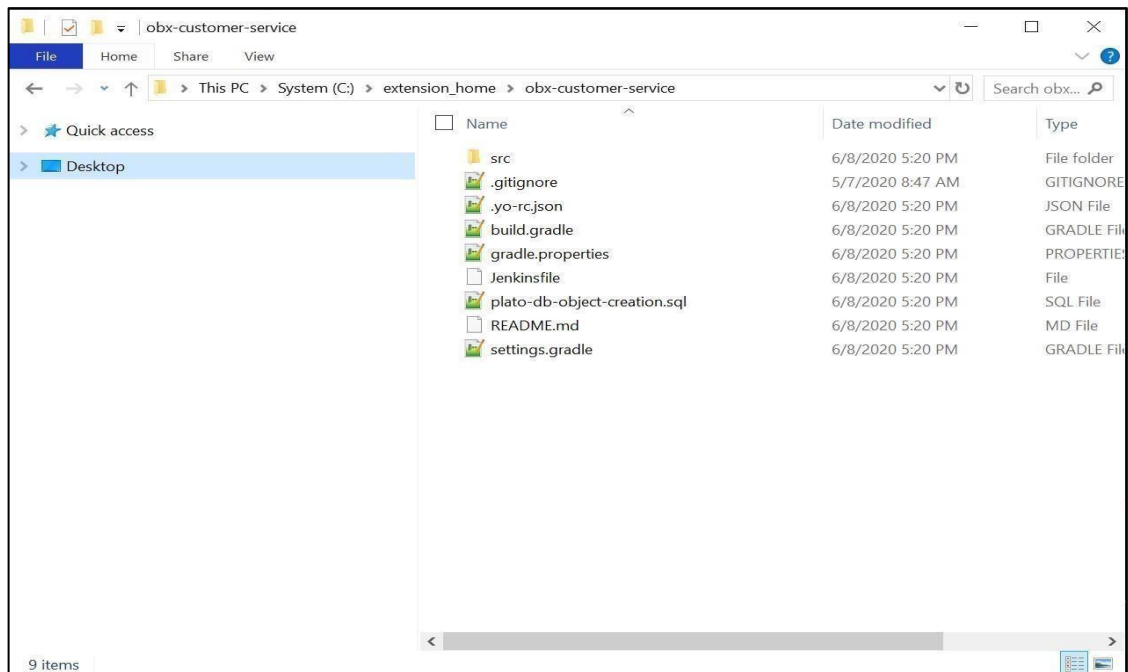
Copyright © 2024-2025, Oracle and/or its affiliates. All rights reserved.

? Select the product family: Oracle Banking Extensibility Workbench
? Enter name of service (I'll add -service to it): customer
? Is it a Master type component? Yes
? Select service tenant type: Single Tenant
? Enter name of Infra (OBMA) data source (I'll add prefix jdbc/ to it): OBMA
? Enter name of Security data source (I'll add prefix jdbc/ to it): PLATO_SECURITY
? Enter name of this service data source (I'll add prefix jdbc/ to it): ENTITY
? Enter product release version: 14.7.5.0.0
? Enter the absolute path of xdl file: D:\OBX\xdl-files\customer.xdl
```

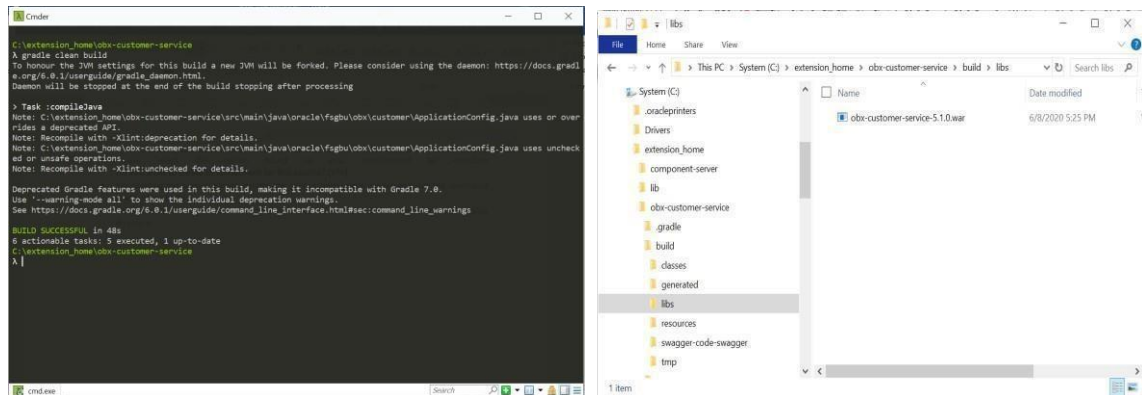
- Once this command is fired, this will take you to next section where it will prompt other set of questions. Answer them accordingly to your setup and requirement.
- Select the type of component according to your requirement.

```
? Is it a Master type component? (Y/n) |
```

Once all the questions are answered and path of XDL is given, it will generate a folder inside the extension_home folder.



- Please select the option based on your requirement for question: **Do you want to create a Data Segment for this service?(Y/n).**
- For building the service please go into the service folder from cmd and run the command: **gradle clean build.**
- This will build the service and we can find the war of the service getting created inside the build/libs directory.



- Use this service and deploy it in your environment.

Notes:

- DB scripts for the service will be generated inside the folder:
\extension_home\obxcustomer-service\src\main\resources\db.
- Please Compile the Entity script in the entity schema created for extensions only.
- Service created as part of extension should be deployed in separate domain and should not be mixed or co-deployed with any other product specific services.
- Here SMS (Security Management System) scripts are also generated:
\extension_home\obxcustomer-service\src\main\resources\db\sms
- Execute the SMS script in sms schema, here we only generate the functional activity of service. Assigning to proper role should be done according to the steps mentioned in base application.
- Here along with SMS and Entity, CMC scripts are also generated under folder:
\extension_home\obx-customer-service\src\main\resources\db\cmc
- Please execute them in the CMC schema.
- **Screen Class and Data Segment** has to be maintained from the UI which is present under common core.

3.3.2 RSOV2 DS

For Nov patchset innvation - RSOv1 is disscontinued and RSOv2 should be adopted for all customizations for maintenance services.

This section describes the process to generate the rsov2 data segment.

Here we can generate Master Type of data segment or child type of data segment.

```
Cmder
D:\OBX\OBX_Workbench\extension_home
λ obx service rsov2 -c

OBX

ORACLE BANKING EXTENSIBILITY WORKBENCH October 2024 (14.7.5.0.0)

Copyright © 2024–2025, Oracle and/or its affiliates. All rights reserved.

? Select the product family: Oracle Banking Extensibility Workbench
? Enter name of service (I'll add -service to it): customer
? Is it a Master type component? Yes
? Select service tenant type: Single Tenant
? Enter name of Infra (OBMA) data source (I'll add prefix jdbc/ to it): OBMA
? Enter name of Security data source (I'll add prefix jdbc/ to it): PLATO_SECURITY
? Enter name of this service data source (I'll add prefix jdbc/ to it): ENTITY
? Enter product release version: 14.7.5.0.0
? Enter the absolute path of xdl file: D:\OBX\xdl-files\customer.xdl
```

Master Type: This will create two components one would be core component of product services which will contain utility service, the other one would be the master type of component that needs to be included in the core services folder.

Child Type: This will create only one component that needs to be included in the core services (containing utility).

1. Navigate to same extension_home folder using cmd.
2. Use the command **obx service rsov2 -c**.
3. Once this command is fired, this will take you to next section where it will prompt other set of questions. Answer them accordingly to your setup and requirement.
4. Select the type of component according to your requirement.
5. Once all the questions are answered and path of XDL is given, it will generate the folders accordingly inside the extension_home.
6. Please select the option based on your requirement for question: **Do you want to create a Data Segment for this service?(Y/N)**
7. Include the folders created either master or child inside the (core-services), folder and make the modifications accordingly.
8. Use this service and deploy it in your environment.

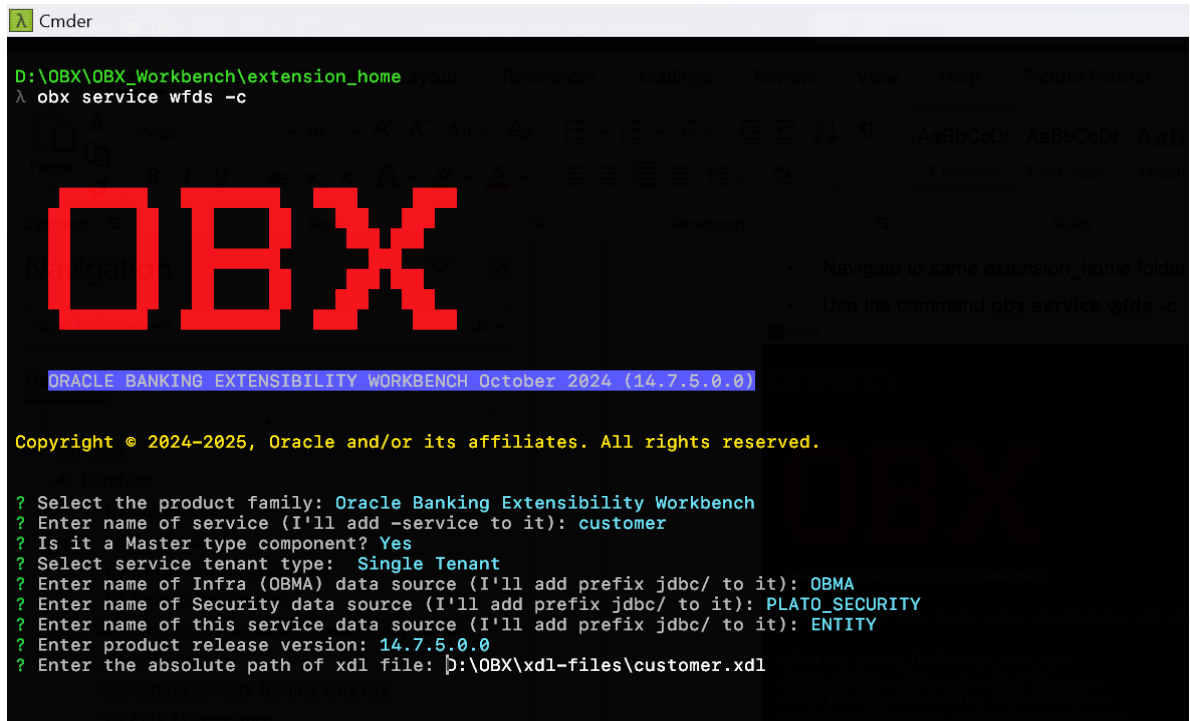
3.3.3 Workflow DS

This section describes the process to generate the workflow data segment. Here we can generate Master Type of data segment or child type of data segment.

Master Type: This case is used when user wants to generate the complete flow from scratch. It will generate the new screen class code for the data segments.

Child Type: This is primarily used when user wants to attach a single data-segment in the existing flow/process. Generally, this existing flow/process is available in the base product. We use the same screen class code from base and attach our data segment to it To generate it please follow the below steps:

- Navigate to same extension_home folder using cmdr.
- Use the command **obx service wfds -c**



```
Cmder
D:\OBX\OBX_Workbench\extension_home
λ obx service wfds -c

OBX

ORACLE BANKING EXTENSIBILITY WORKBENCH October 2024 (14.7.5.0.0)

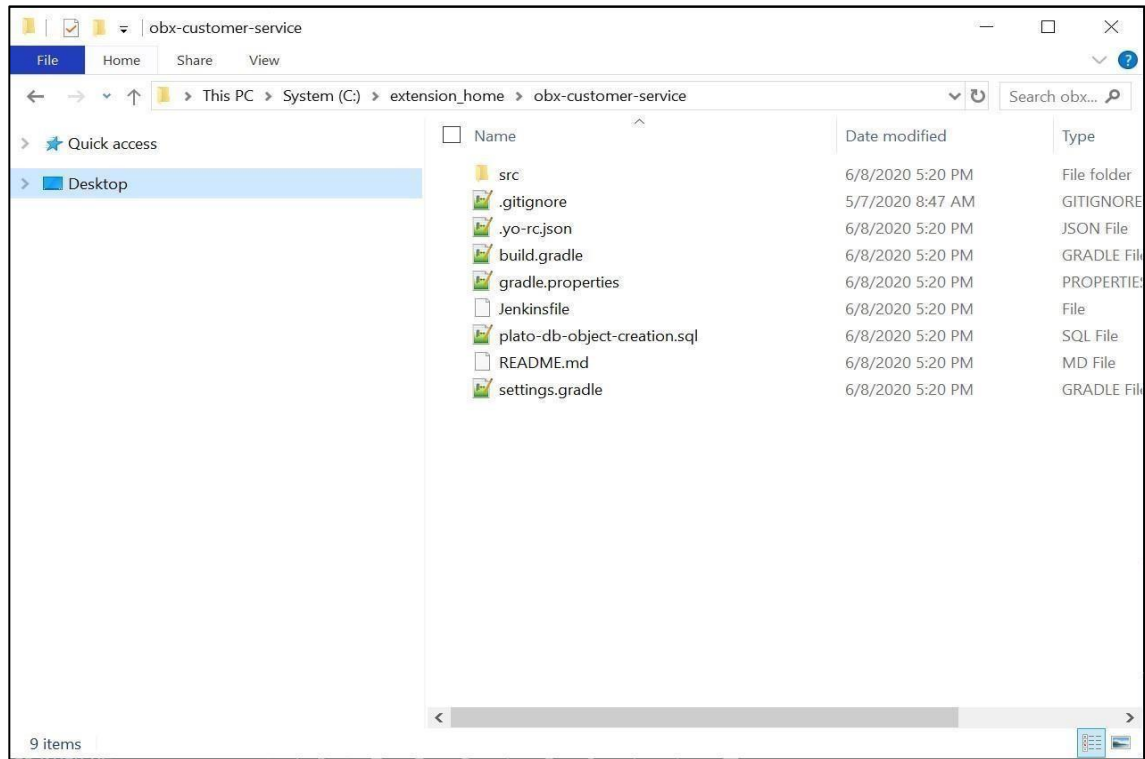
Copyright © 2024-2025, Oracle and/or its affiliates. All rights reserved.

? Select the product family: Oracle Banking Extensibility Workbench
? Enter name of service (I'll add -service to it): customer
? Is it a Master type component? Yes
? Select service tenant type: Single Tenant
? Enter name of Infra (OBMA) data source (I'll add prefix jdbc/ to it): OBMA
? Enter name of Security data source (I'll add prefix jdbc/ to it): PLATO_SECURITY
? Enter name of this service data source (I'll add prefix jdbc/ to it): ENTITY
? Enter product release version: 14.7.5.0.0
? Enter the absolute path of xdl file: D:\OBX\xdl-files\customer.xdl
```

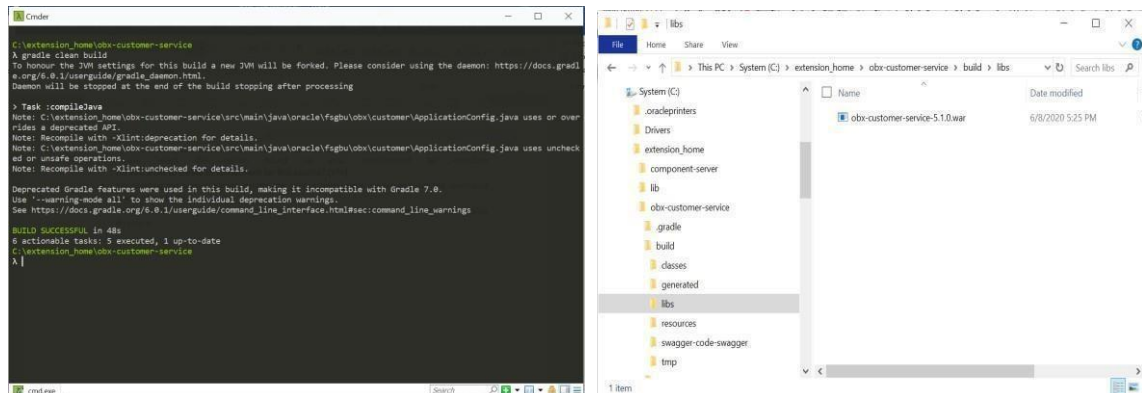
- Once this command is fired, this will take you to next section where it will prompt other set of questions. Answer them accordingly to your setup and requirement.
- Select the type of component according to your requirement.

```
? Is it a Master type component? (Y/n) |
```

- Once all the questions are answered and path of XDL is given, it will generate a folder inside the **extension_home** folder.



- Please select the option based on your requirement for question: **Do you want to create a Data Segment for this service? (Y/n).**
- For building the service please go into the service folder from cmdr and run the command **gradle clean build**.
- This will build the service and we can find the war of the service getting created inside the build/libs directory.



- Use this service and deploy it in your environment.

Notes:

- DB scripts for the service will be generated inside the folder.
\extension_home\obxcustomer-service\src\main\resources\db
- Please Compile the Entity script in the entity schema created for extensions only.
- Service created as part of extension should be deployed in separate domain and should not be mixed or co-deployed with any other product specific services.

- Here SMS (Security Management System) scripts are also generated.
`\extension_home\obxcustomer-service\src\main\resources\db\sms`
- Execute the SMS script in sms schema, here we only generate the functional activity of service. Assigning to proper role should be done according to the steps mentioned in base application.
- Here along with SMS and Entity, CMC scripts are also generated under folder.
`\extension_home\obx-customer-service\src\main\resources\db\cmc`
- Please execute them in the CMC schema.
- Screen Class and Data Segment has to be maintained from the UI which is present under common core.

3.4 Simple Publisher/Subscriber Event Service

This section describes the process to generate simple publisher/subscriber event service. To generate it please follow the below steps:

- Navigate to same extension_home folder using cmdr.
- Use the command **obx event -c**
- Once this command is fired, this will take you to next section where it will prompt other set of questions. Answer them accordingly to your setup and requirement.

```

Cmder
D:\OBX\OBX_Workbench\extension_home
λ obx event -c

OBX

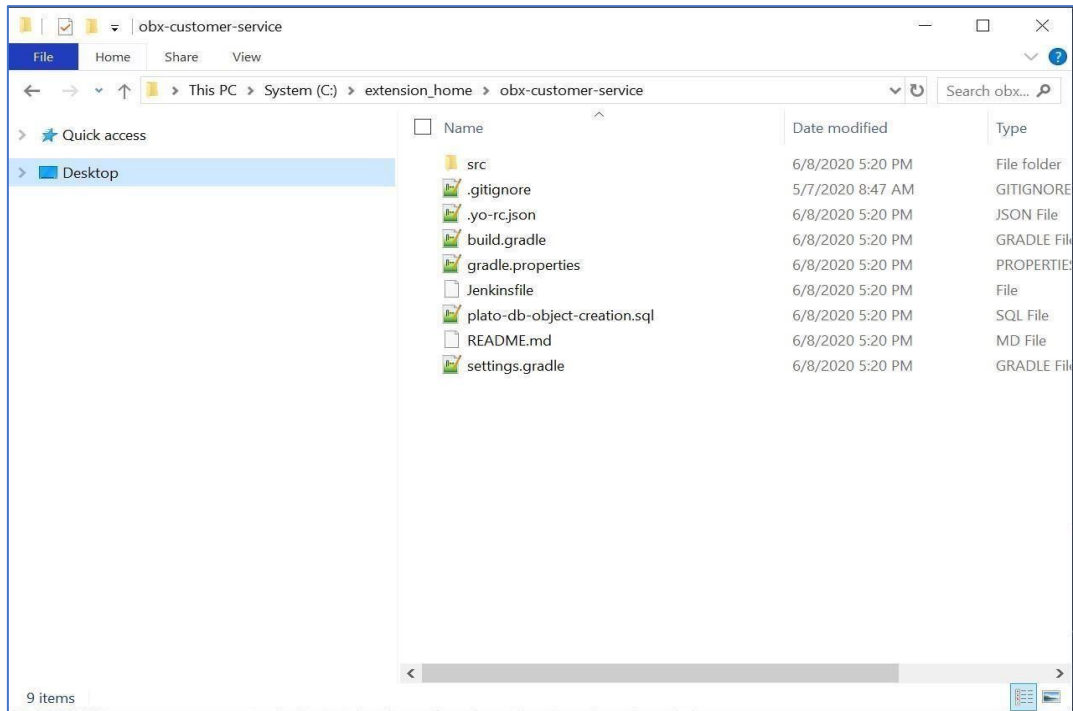
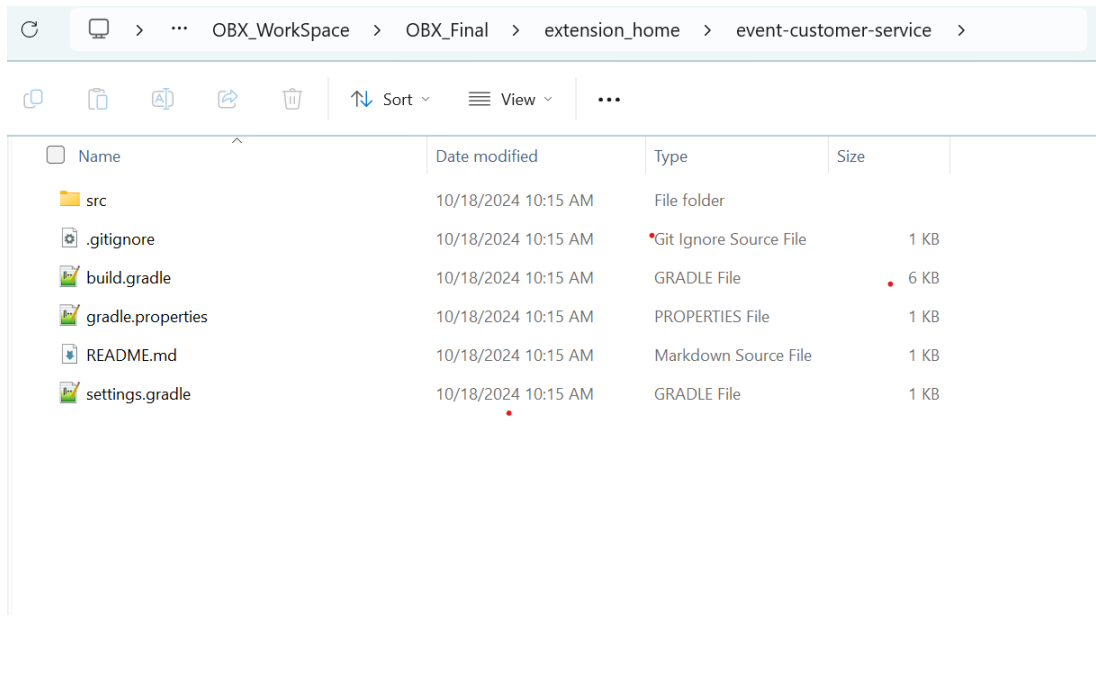
ORACLE BANKING EXTENSIBILITY WORKBENCH October 2024 (14.7.5.0.0)

Copyright © 2024-2025, Oracle and/or its affiliates. All rights reserved.

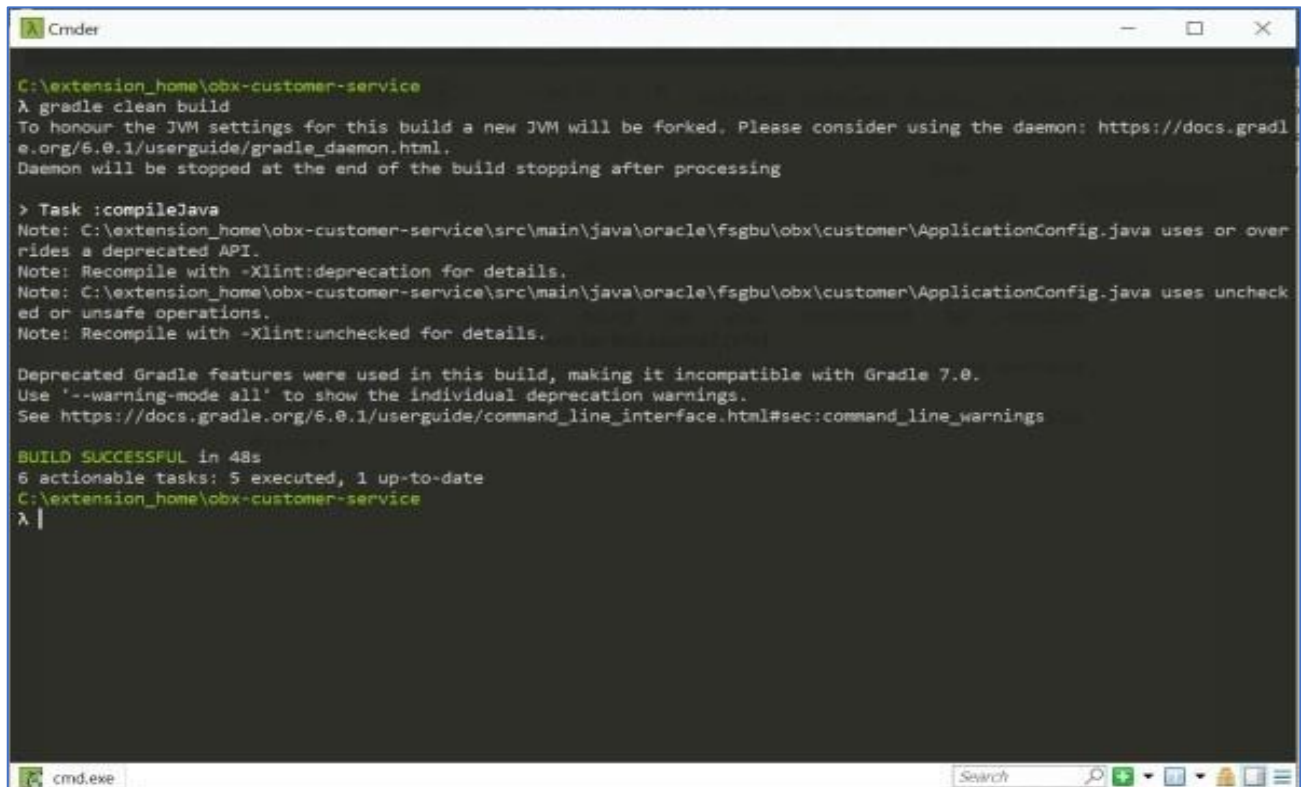
? Enter name of service (I'll add -service to it): customer
? Enter the hostname for kafka server: localhost
? Enter the port for kafka server: 9092
? Enter the hostname for zookeeper server: localhost
? Enter the port for zookeeper server: 2181
? Enter number of events: 1
? Please Select the Type of event/stream you wish to create publisher
? Enter the name of event/stream 1:
? Enter topic name for the selected event/stream: customer
? Enter avro schema name for the selected event: Avrosch

```

- Once all the questions are answered and path of XDL is given, it will generate a folder inside the extension_home folder.



- For building the service please go into the service folder from cmd and run the command **gradle clean build**.
- This will build the service and we can find the war of the service getting created inside the build/libs directory.



```
C:\extension_home\obx-customer-service
λ gradle clean build
To honour the JVM settings for this build a new JVM will be forked. Please consider using the daemon: https://docs.gradle.org/6.0.1/userguide/gradle_daemon.html.
Daemon will be stopped at the end of the build stopping after processing

> Task :compileJava
Note: C:\extension_home\obx-customer-service\src\main\java\oracle\fsghu\obx\customer\ApplicationConfig.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
Note: C:\extension_home\obx-customer-service\src\main\java\oracle\fsghu\obx\customer\ApplicationConfig.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

Deprecated Gradle features were used in this build, making it incompatible with Gradle 7.0.
Use '--warning-mode all' to show the individual deprecation warnings.
See https://docs.gradle.org/6.0.1/userguide/command_line_interface.html#sec:command_line_warnings

BUILD SUCCESSFUL in 48s
6 actionable tasks: 5 executed, 1 up-to-date
C:\extension_home\obx-customer-service
λ |
```

- Use this service and deploy it in your environment.

3.5 Batch Service

This section describes the process to generate OBMA based Batch service. The purpose of this service is to create reader, writer and processor in which methods will be written according to business use case. To generate it please follow the below steps:

- Navigate to same extension_home folder using cmd.
- Use the command **obx batch -c**
- Inputs to be given after the command.
- Select the product family.
- Enter name of the service(I'll construct it as <productFamilyName>-batch<serviceName>-extended-services):
- Enter product release version.
- Upon successful creation of batch service, user will find a folder generated with <productFamilyName>-batch-<serviceName>-extended-services having the sample service code generated.
- The generated code has two types of batch job template inside.
 - o Simple job creation using spring batch features. The method name for this type of job creation is jobName(). The reader, writer, processor etc are taken from spring's itemReader, itemWriter, itemProcessor.

```

141
142     @Bean(name = "jobName")
143     public Job jobName(JobBuilderFactory jobBuilderFactory, StepBuilderFactory stepBuilderFactory,
144         Reader itemReader, Processor itemProcessor, Writer itemWriter) {
145
146         Step step = stepBuilderFactory.get("step1").chunk(10).reader(itemReader).processor(itemProcessor)
147             .writer(itemWriter).build();
148
149         return jobBuilderFactory.get("jobName").start(step).build();
150     }
151
152

```

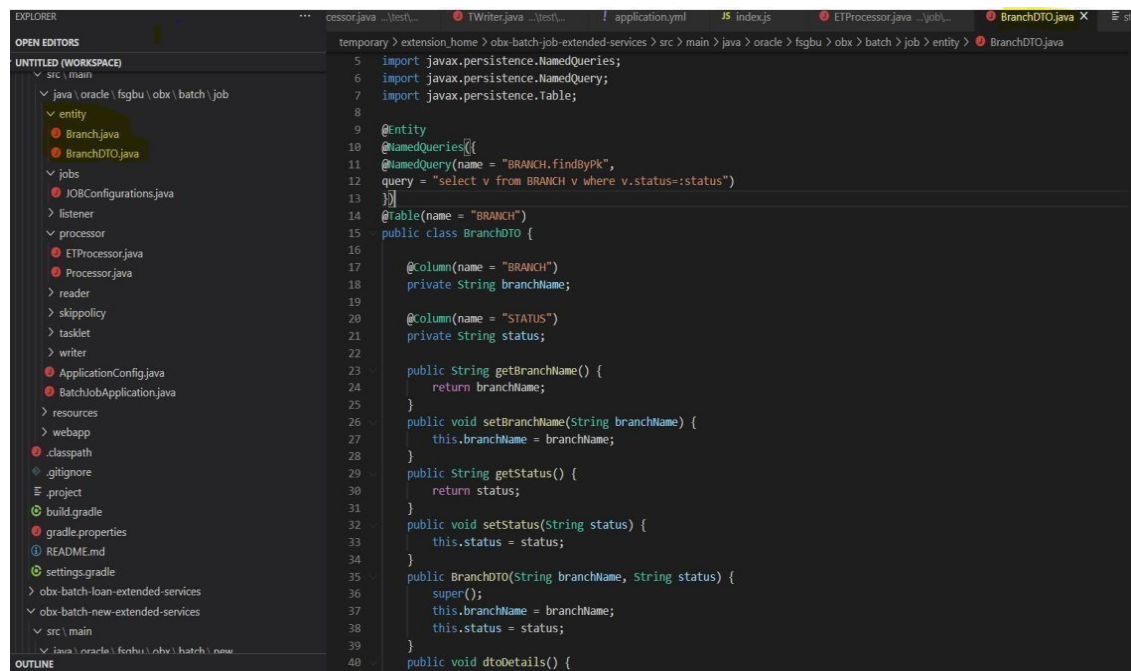
- Plato batch type job creation by keeping plato batch into consideration. The method name for this type of job creation is batchProcessJob(). In this case reader is specified as EReader, writer as TWriter and processor as ETProcessor. E means the entity to be read for this job; T means the transformed object to be persisted in the database. Hence the names are given in that manner.

```

@Bean
public Job batchProcessJob() throws Exception {
    return jobBuilderFactory.get("batchProcessJob").start(taskletStep()).next(chunkStep()).build();
}

```

- For plato batch type job, user needs to write his/her entity classes in which the business logic will be kept. For example, this is the structure of the entity class highlighted in the left.

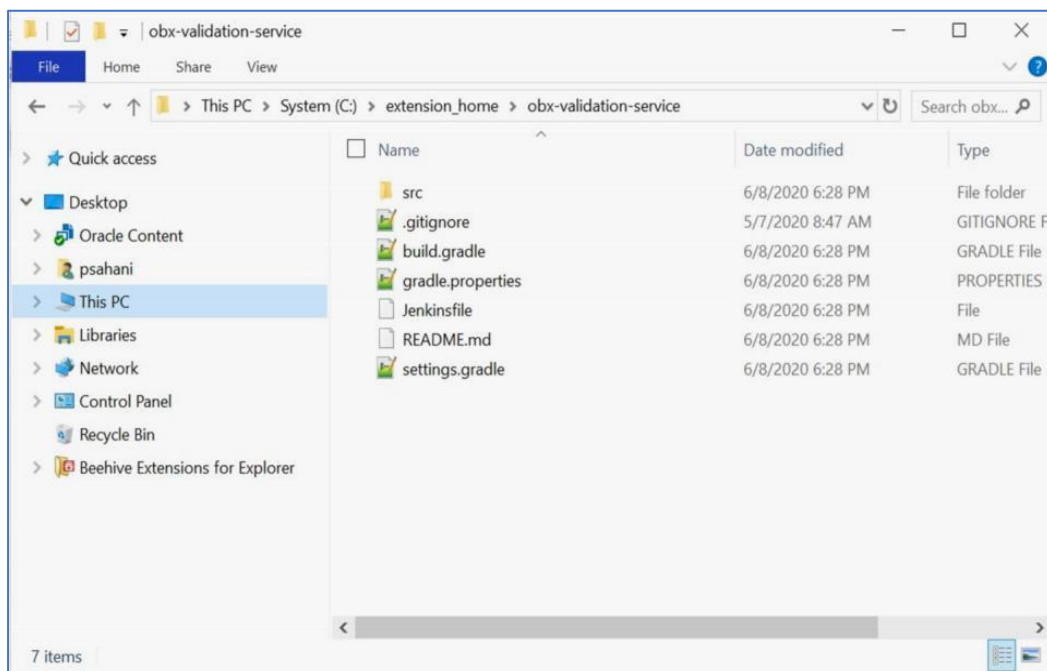


- One needs to write methods for reader, writer, and processor accordingly.
- To build the service
 - Navigate to the service.
 - Fire the command gradle clean build.
 - This will create the war file of the service in the folder structure build/libs/
productFamilyName>-batch-<serviceName>-extended-services.war.

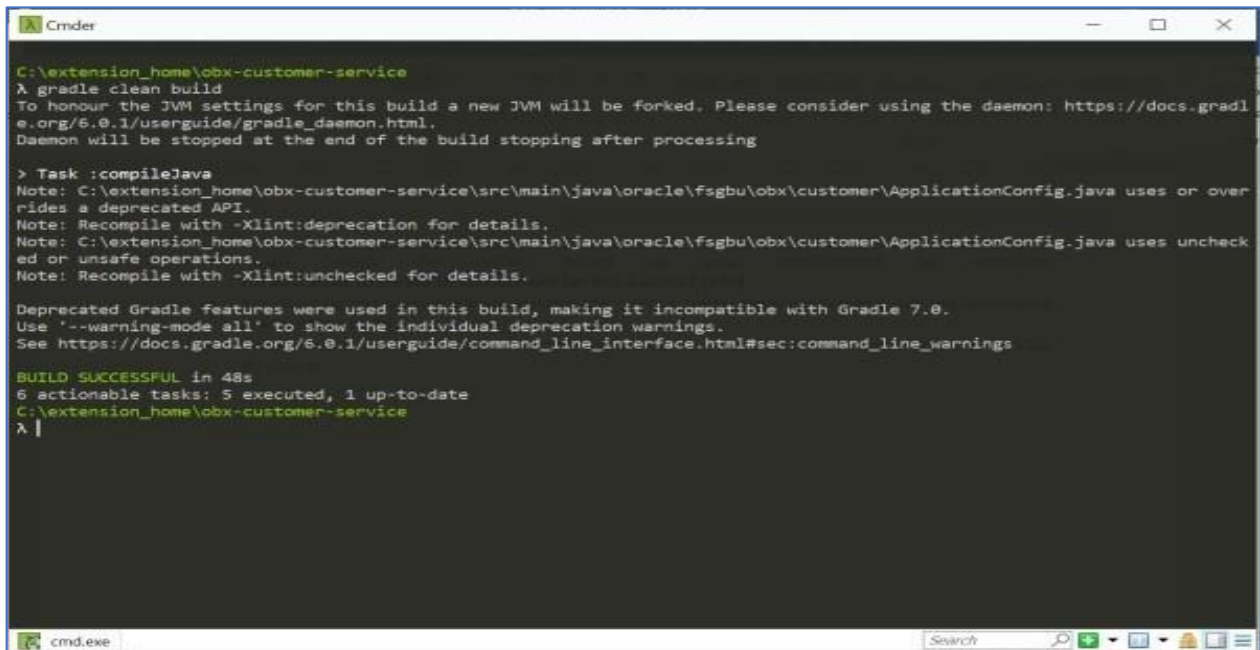
3.6 Custom Validation Service

This section describes the process to generate validation service. The purpose of this service is to perform custom validations on the base service. It is important to remember that we will be only able to perform the validation and never modify the payload to change the value. To generate it please follow the below steps:

- Navigate to same extension_home folder using cmdr.
- Use the command obx validation -c.
- It will generate a folder inside the extension_home folder with obx-validation-service.



- For building the service, please go into the service folder from cmdr and run the command **gradle clean build**
- This will build the service and we can find the war of the service getting created inside the build/libs directory.



```
C:\extension_home\obx-customer-service
λ gradle clean build
To honour the JVM settings for this build a new JVM will be forked. Please consider using the daemon: https://docs.gradle.org/6.0.1/userguide/gradle_daemon.html.
Daemon will be stopped at the end of the build stopping after processing

> Task :compileJava
Note: C:\extension_home\obx-customer-service\src\main\java\oracle\fsghu\obx\customer\ApplicationConfig.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
Note: C:\extension_home\obx-customer-service\src\main\java\oracle\fsghu\obx\customer\ApplicationConfig.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

Deprecated Gradle features were used in this build, making it incompatible with Gradle 7.0.
Use '--warning-mode all' to show the individual deprecation warnings.
See https://docs.gradle.org/6.0.1/userguide/command_line_interface.html#sec:command_line_warnings

BUILD SUCCESSFUL in 48s
6 actionable tasks: 5 executed, 1 up-to-date
C:\extension_home\obx-customer-service
λ |
```

- Use this service and deploy it in your environment.

3.7 Steps to adopt Multi Entity in existing service

Plato Micro Service Dependencies Changes

```
compile("release.obma.plato.21_0_0.services:plato-microservice-dependencies:6.0.0")
```

Eventhub dependency changes

```
compile("release.obma.plato.21_0_0.services:plato-eventhub-dependencies:6.0.0")
```

PlatoInterceptor Changes

```
@Bean public MappedInterceptor gemInterceptor(PlatoInterceptor
platoInterceptor){
LOG.info("Added interceptor for fetching the application headers"); return new
MappedInterceptor(new String[] { "/*" }, platoInterceptor); }
```

Logging (Please include only ,%X{entityId}, change. Rest of them remain as per the old logback.xml)

Please include only %X{entityId} in the existing value of the LOG_PATTERN of your logback.xml

One sample format is below,

```
<property name="LOG_PATTERN" value="%clr(%d{yyyy-MM-dd
HH:mm:ss.SSS}){faint} %clr(%5p [%${applicationName},%X{entityId},%X{X-B3-
TraceId:},%X{X-B3-SpanId:-},%X{X-Span-Export:-}]) %clr([%mdc{env:-null}] [%mdc{tenant:- null}]
[%mdc{user:-null}] [%mdc{branch:-null}])%faint} %clr(${PID:- }){magenta} %clr(---
-){faint} %clr([%15.15t]){faint} %clr(%-
40.40logger{39}){cyan} %clr(:){faint} %m%n${LOG_EXCEPTION_CONVERSION_WORD:-%wEx}" />
```

Feed Services

Folder structure should be */parentFolder/<<entityId>>/{fileName}

```
compile("release.obma.plato.21_0_0.services:plato-feed-core:6.0.0")
```

Caching Strategy

```
@Cacheable(value = "customers", key = "{ <<functionalKeys>> T(oracle.fsgbu.plato.core.persistance.provider.PlatoHolder).getCurrentEntityId() }")
```

Introduce appld in application.yml of individual micro services

If the service is a eventhub based service they should use

```
spring:
  application:
    appID:
```

If the service is a non-eventhub based service they can use either

```
spring:
  application:
    appID:
```

or

```
appId: <<appId>>
```

3.8 Service Extensibility

Structure of Service Extensions can be seen in below table.

Component Name	Component Description
<< micro - service - name >>extn.jar	Extension jar
<< <i>micro - service - name</i> >>.war	WAR File which refers to << <i>micro - service - name</i> >>extn.jar during runtime.

Step # 1)

Add all the required classes from << *micro - service - name* >>.war to the classpath of<< *micro - service - name* >>extn.jar project and then build it.

For creation of war, we can use the command “obx create-jar”

- Go to extension home.
- Run the command **obx create-jar**.
- It will prompt you with the location of the extended war file. (After giving the location give enter two times).
- On providing the war file, it will create a jar for the same in the same location.

Step # 2)

The **build.gradle** of the extension project should include the statement

```
compileOnly files("classes")
```

Step # 3)

For shared libraries we follow the optional packages approach. The following entries are expected in the **MANIFEST.MF** of respective war file.

Extension-List: << micro - service - name >>-extn,
<< micro - service - name >>-extn-Extension-Name: << micro - service - name >>-extn

For this, we need to modify the **build.gradle** of war files to include the below statements.

```
war {  
    ...  
    manifest {  
attributes(  
        "Extension-List": "<< micro - service - name >>-extn",  
        "<< micro - service - name >>-extn-Extension-Name": "<< micro - service - name >>-extn"  
    )  
}  
    ...  
}
```

Step # 4)

In the extension jar create a new service class that extends the original service class and annotate the class with "**@Primary**" annotation to give the service class in the extension jar higher precedence.

CustomerServiceImplExt

```
@Primary@Service  
public class CustomerServiceImplExt extends CustomerServiceImpl  
implements CustomerService {.....}
```

If the extension jar is provided the methods in the extension jar will be invoked or else the methods in the original war will be invoked.

Step # 5)

Weblogic deployment

Deploy the extension jar first in the weblogic then in the same server deploy the war.

Tomcat deployment

Modification in server.xml

<Context ...>

<Resources>

<PreResources className="org.apache.catalina.webresources.DirResourceSet" base="<<directory containing the extension jars" webAppMount="/WEB-INF/lib"/>

</Resources>

</Context>

Step # 6)

The class names inside the **<< micro - service - name >>-extn.jar**, should have the naming convention as below,

<<basePackageNameOf<< micro - service - name >>.war>>.<<service/controller/model>>

4 UI Extensions – Web Component

This section describes the OBX capability to generate to different types of web components. Each Web component can run itself locally. There are various types of these web components each serving different functionality.

Standalone Component: A standalone component can be thought of as a smallest reusable.

UI component. They are generally the building blocks of main screens. Components like amount, text fields, lov etc. are part of standalone components.

Virtual Page: A virtual page can be thought of as a screen or a web page in single page applications. They are loaded inside the content area next to the left navigation menu. Important point to remember when designing virtual page is, it appends and changes the router (app URL) when navigation is done.

Container Component: These Components are a special type of components which are loaded inside a container called as Wizard. It gives functionality like minimizing the component and open multiple screens simultaneously on the screen. Important point to remove here is that these components never change to router state, so bookmarking is not possible for these screens.

The screenshot shows the Oracle Bank Virtual Page. The header includes the Oracle logo, the page title "Bank Virtual Page", and user information: "Plato (000)", "Apr 13, 2018", and "PAWAN pawan@oracle.com". The main section is titled "Bank Details" and contains a "New" button. Below this is a form with the following fields: "Bank Code", "Bank Name", "Number Of Branches", "Default Currency" (with a search icon), and "Holiday" (with a dropdown arrow). At the bottom, there is a table with columns: "Address", "City", "State", "Country", and "Pincode". The table is currently empty, displaying "No data to display." and has "+" and "-" buttons for adding or removing rows.

Data/Resource Segment: A component designed using data segment approach are similar to that of virtual page but are always part of flow or process and loaded like container components. It is helpful in use cases where data to be captured is huge or is captured in various stages of applications.

The screenshot shows the Oracle Dashboard. The header includes the Oracle logo, the page title "Dashboard", and user information: "Plato (000)", "Apr 13, 2018", and "PAWAN pawan@oracle.com". The main section is titled "Customer DS Details" and contains a sidebar with two segments: "Customer" (selected) and "Income Details". The "Customer" segment displays a form with the following fields: "Customer Id" (with a dropdown arrow, value "CUST100"), "First Name" (value "firstname"), "Last Name", "Dob" (with a calendar icon), "Address", and "Mobile Number" (value "987654321"). At the bottom right, there are four buttons: "Back", "Next", "Save & Close", and "Cancel". The text "Screen (1 / 2)" is visible in the top right corner of the main content area.

In above screenshot Customer and Income Details on left are two data segments which is part of Customer DS Details Application.

Widgets: Widgets are special components meant for dashboard. These are generally created in the form of tiles and are attached to the dashboard.



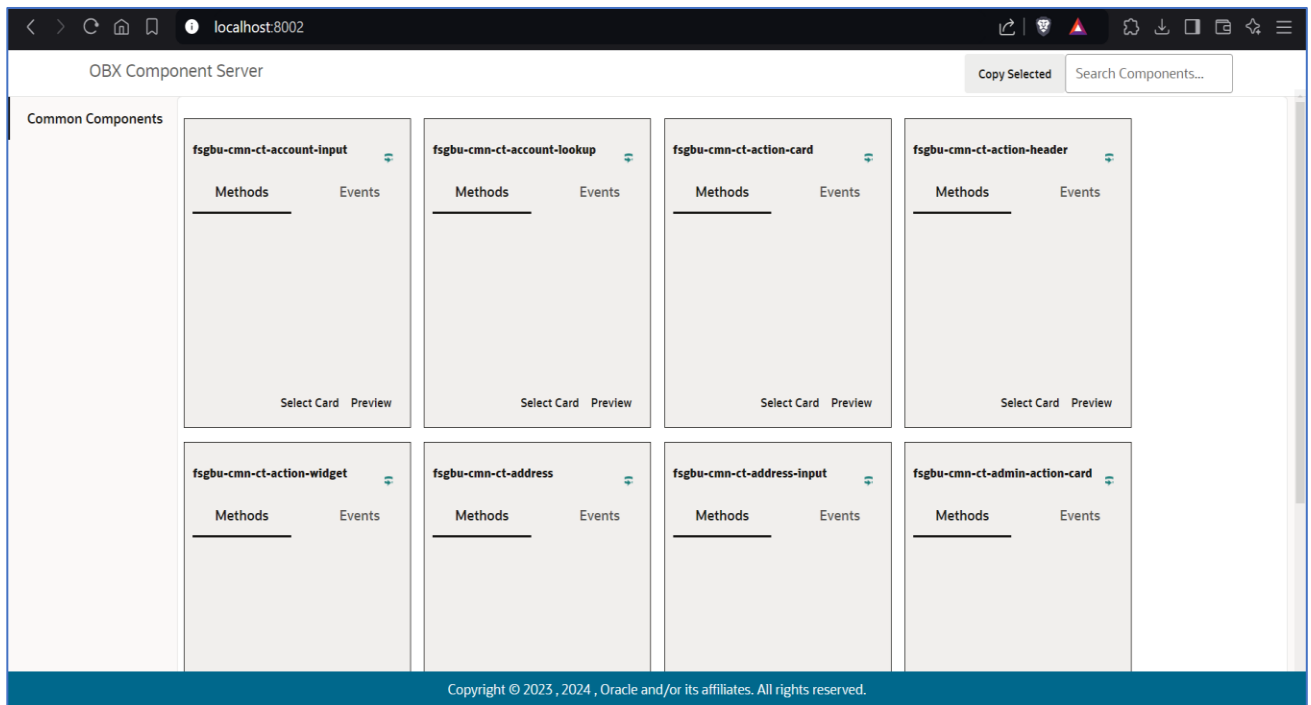
Note:

- All the above components except standalone components have SMS applied on it.
- We have to assign functional activity of web components to the role and then only they are integrated with the main application shell.
- Also, it always recommended to try and run the component locally before merging them into main application.
- All web components come bundled with testing framework including unit test cases and functional test. Therefore, it's a good practice to write them along with the development.

4.1 Component Server

It is one of highlight feature from OBX. A component server is hub of components which are available from the base/kernel application. As each component is developed individually and reusable, we can use this functionality to reuse even the components from base application. It saves time as we don't have to code same thing again and again. We can reuse as many components as possible from base application into extensions.

Component server is started automatically when you generate the web component. It runs on <http://localhost:8002>. One can simply go to browser and copy components and put them in a metadata.js file which is created inside the component and by doing so it indicated OBX that we have to reuse the component and it generates the code automatically.



4.2 Simple Standalone

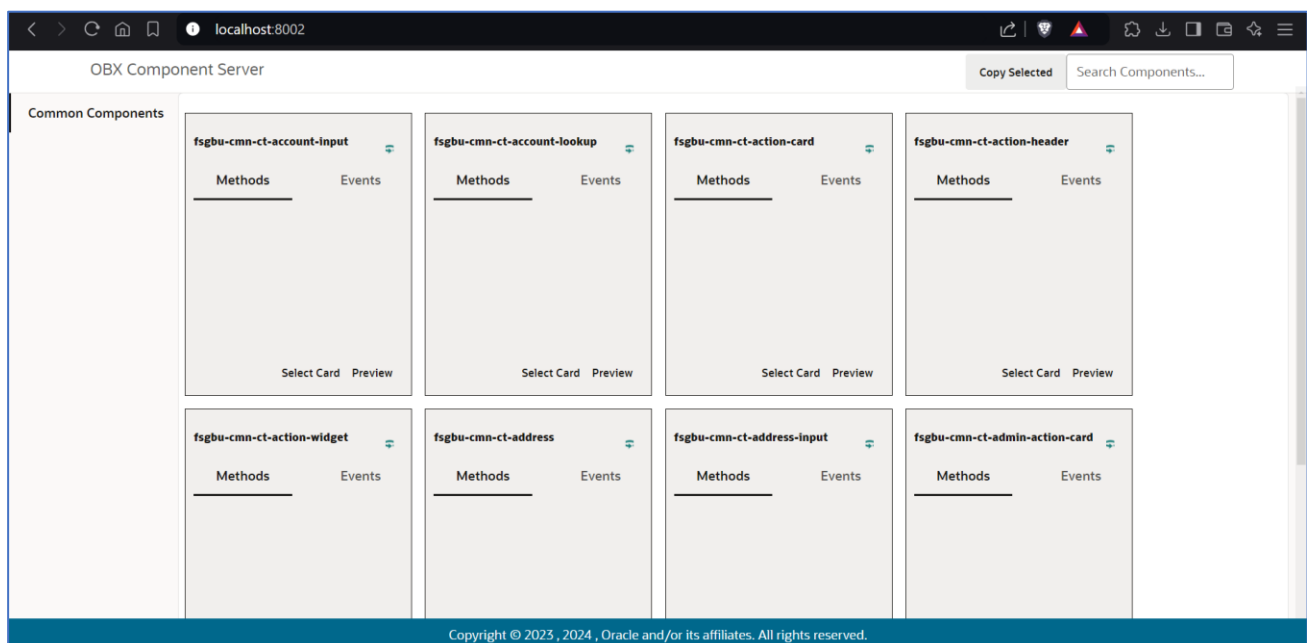
This section describes the process of creating the simple standalone component using OBX.

Following are the steps needed to be followed:

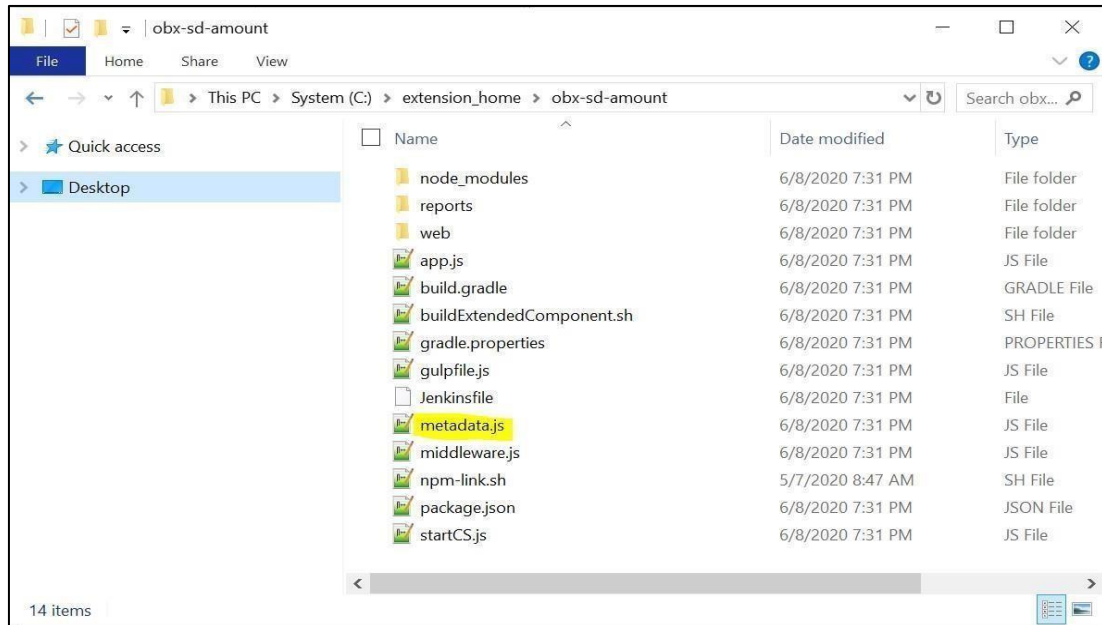
- Navigate to **extension_home** folder from cmdr.
- Use the command: **obx ui —sd** .
- Once this command is executed, this will take you to next section where it will prompt other set of questions. Answer them accordingly to your setup and requirement.

```
Cmder
λ obx ui --sd
? select the product family: Oracle Banking Extensibility Workbench
? Select the name of the standalone component (I'll prepend obx-sd- to it): customer
force ..\yo-rc.json
force C:\Users\saayare\yo-rc-global.json
create buildExtendedComponent.sh
create buildExtendedComponent.bat
create build.gradle
create gradle.properties
create package.json
create Jenkinsfile
create app.js
create gulpfile.js
create startCS.js
create metadata.js
create middleware.js
> Generating Libraries-
```

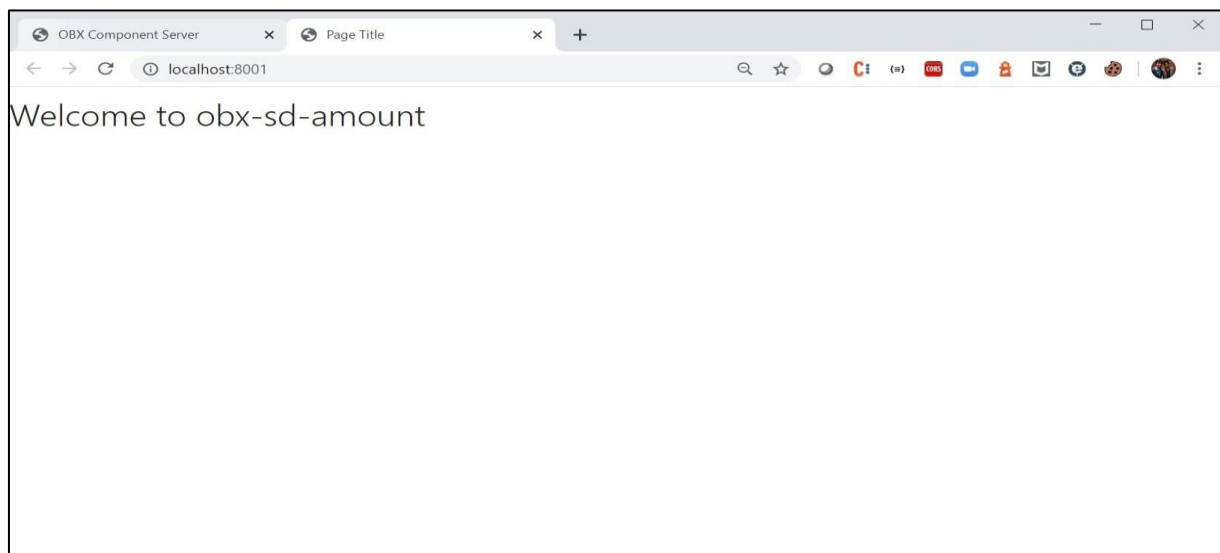
- It will automatically generate the libraries for the component to run locally and you will be also able to see new cmder tab opened where component server is running.
- At this point of time go to browser and navigate to <http://localhost:8002>. You will be able to see component server home page like:



- Select the component which you want to reuse in your extension and paste it in **module.exports = []** inside the **metadata.js** file



- Once done come back to main tab in cmdr where is waiting with question, **Please modify the Metadata.js file before proceeding. Once done press 'y' to proceed?**
- On completing the above process, it will automatically generate the source folder now and open a new tab on cmdr where component will be running. Along with cmdr tab it will automatically open a tab on default browser as well with component rendered on the screen.

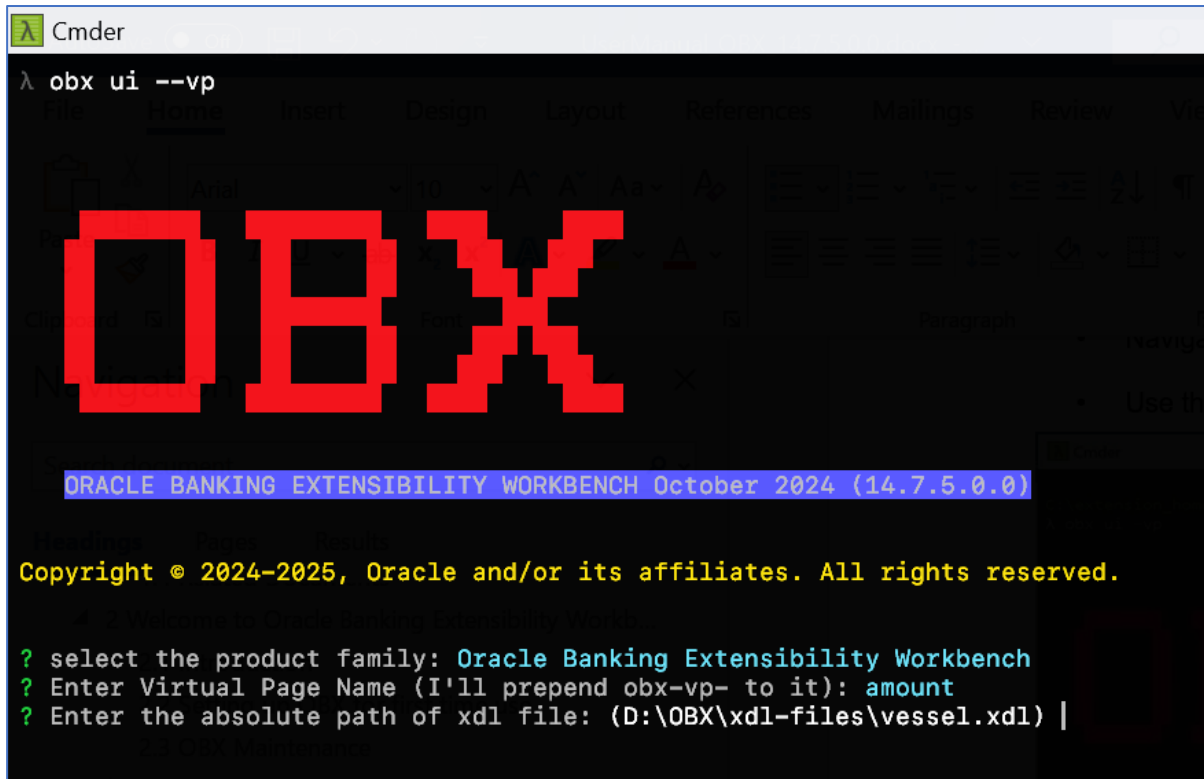


4.3 Virtual Page

This section describes the process of creating the virtual page component using OBX.

Following are the steps needed to be followed:

- Navigate to **extension_home** folder from cmdr
- Use the command **obx ui --vp**



```
λ obx ui --vp
File Home Insert Design Layout References Mailings Review Vie
OBX
ORACLE BANKING EXTENSIBILITY WORKBENCH October 2024 (14.7.5.0.0)
Copyright © 2024-2025, Oracle and/or its affiliates. All rights reserved.
Welcome to Oracle Banking Extensibility Workbench
? select the product family: Oracle Banking Extensibility Workbench
? Enter Virtual Page Name (I'll prepend obx-vp- to it): amount
? Enter the absolute path of xdl file: (D:\OBX\xdl-files\vessel.xdl) |
```

- Once this command is executed, this will take you to next section where it will prompt other set of questions. Answer them accordingly to your setup and requirement.

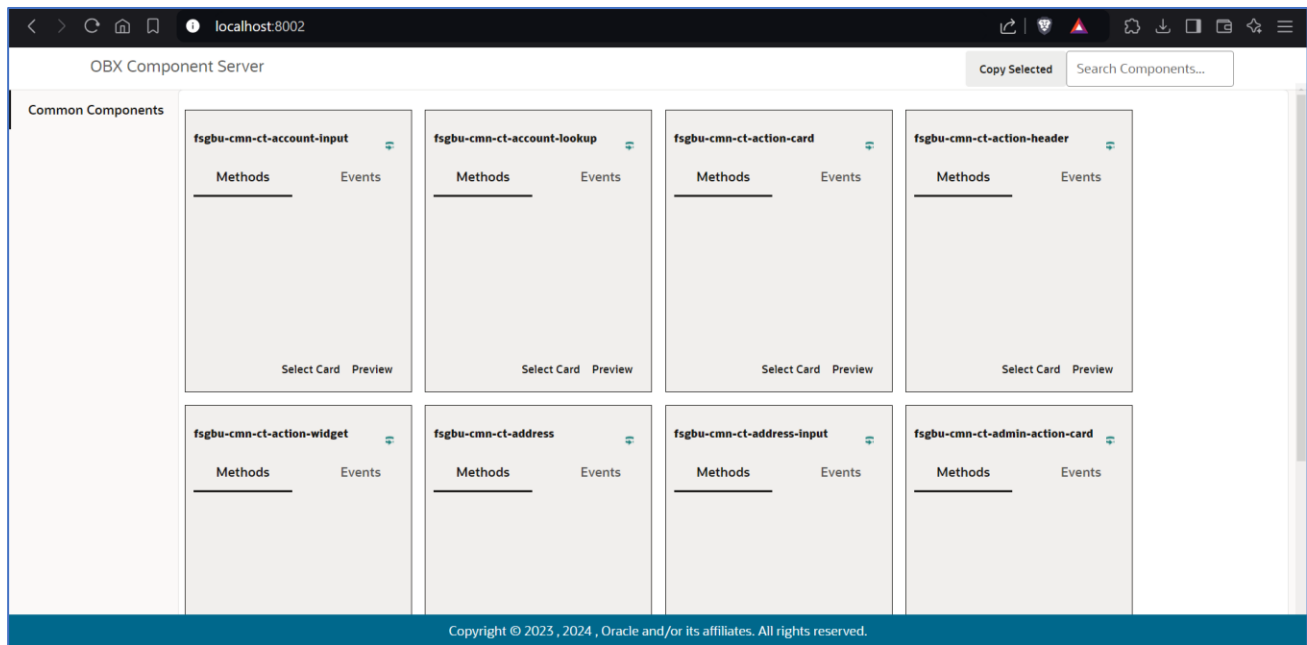
```
Cmder
λ obx ui --vp

ORACLE BANKING EXTENSIBILITY WORKBENCH October 2024 (14.7.6.0.0)

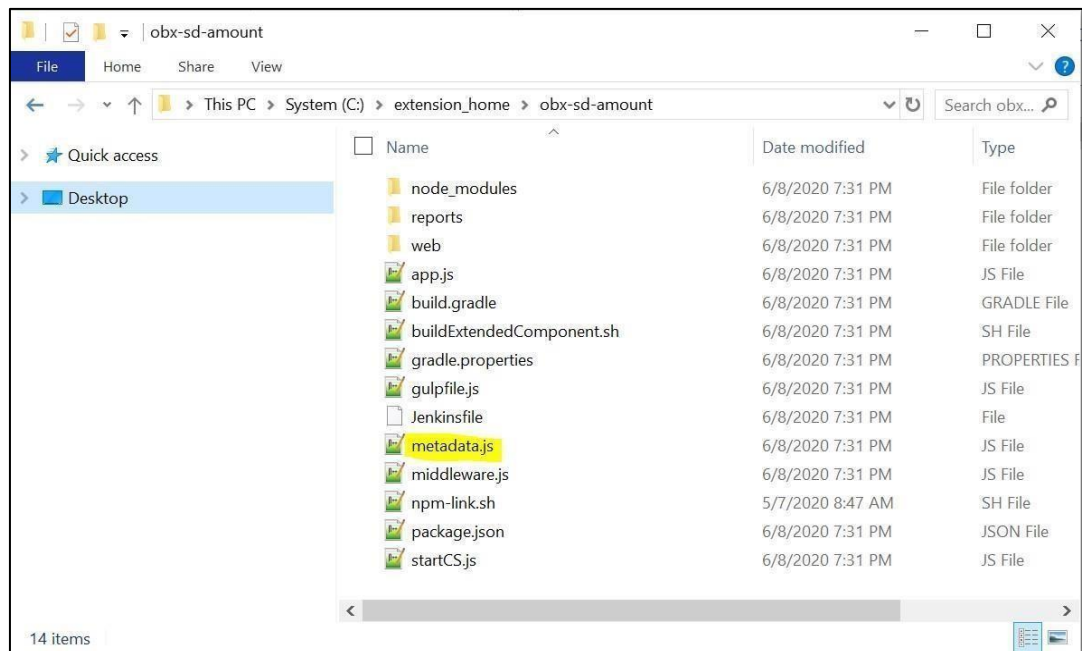
Copyright © 2024-2025, Oracle and/or its affiliates. All rights reserved.

? select the product family: Oracle Banking Extensibility Workbench
? Enter Virtual Page Name (I'll prepend obx-vp- to it): amount
? Enter the absolute path of xdl file: D:\OBX\xdl-files\amount.xdl
  force ..\.yo-rc.json
  force C:\Users\saayare\.yo-rc-global.json
  create buildExtendedComponent.sh
  create buildExtendedComponent.bat
  create build.gradle
  create gradle.properties
  create package.json
  create Jenkinsfile
  create app.js
  create gulpfile.js
  create startCS.js
  create metadata.js
  create middleware.js
> Generating Libraries/|
```

- It will automatically generate the libraries for the component to run locally and you will be also able to see new cmder tab opened where component server is running.
- At this point of time go to browser and navigate to <http://localhost:8002>. You will be able to component server home page like:



- Select the component which you want to reuse in your extension and paste it in `module.exports = []`; inside the `metadata.js` file



- Once done come back to main tab in cmdr where is waiting with question: **Please modify the Metadata.js file before proceeding. Once done press 'y' to proceed?**
- On completing the above process, it will automatically generate the source folder now and open a new tab on cmdr where component will be running.
- Along with cmdr tab it will automatically open a tab on default browser as well with component rendered on the screen.

Reset Save Get All

Vesselname

Ownership

	Companyname	Relationship
No data to display.		

Page 1 (0 of 0 items) |< < 1 > >|

4.4 Maintenance Detail and Summary

This section describes the process of creating the Maintenance Detail and Summary component using OBX. Here we must remember that we will be generating two web components one will be detail component and another one for summary component. Following are the steps needed to be followed:

- Navigate to **extension_home** folder from cmdr
- Use the command **obx ui -mns**
- Once this command is executed, this will take you to next section where it will prompt other set of questions. Answer them accordingly to your setup and requirement.

```
Cmder

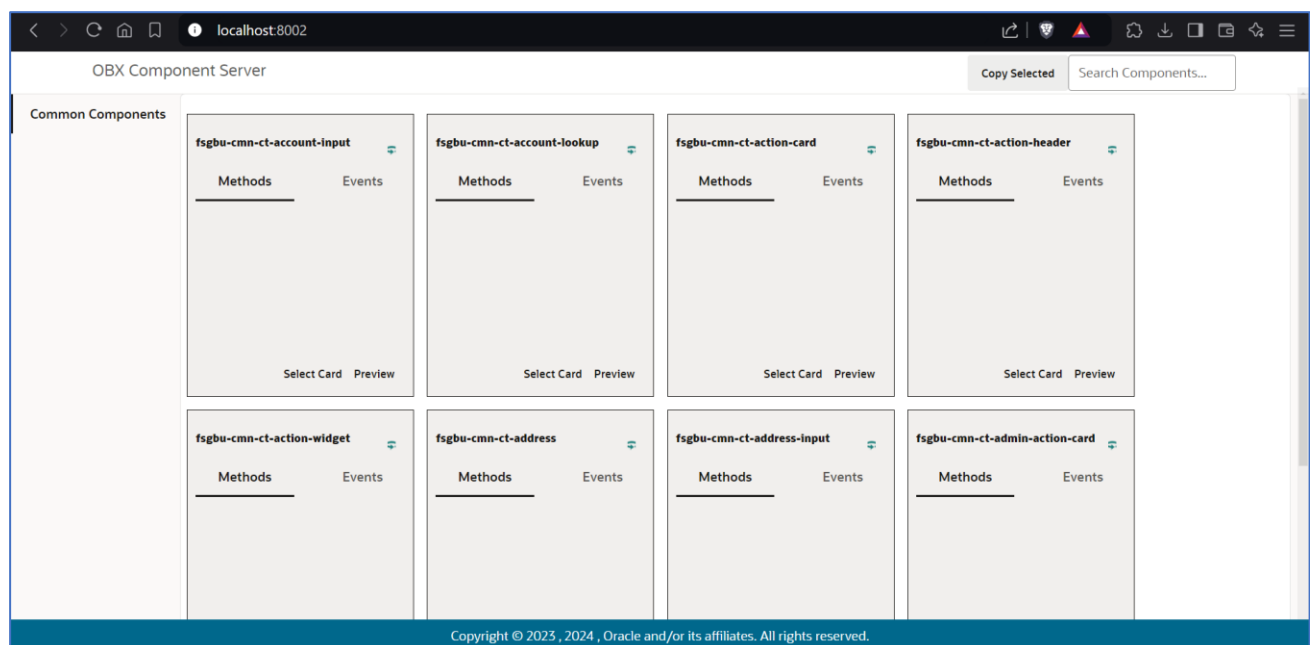
OBX

ORACLE BANKING EXTENSIBILITY WORKBENCH October 2024 (14.7.5.0.0)

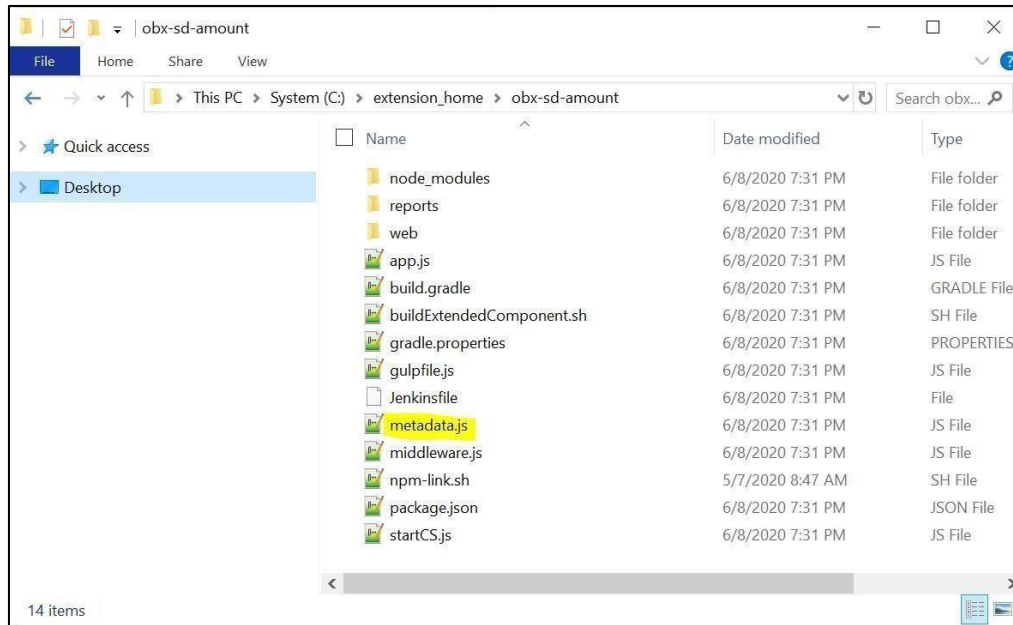
Copyright © 2024-2025, Oracle and/or its affiliates. All rights reserved.

? select the product family: Oracle Banking Extensibility Workbench
? Enter your Web Component Name (I'll prepend obx-mn- and obx-sm- to the components): testmaint
? Enter the absolute path of xdl file: D:\OBX\xdl-files\vessel.xdl
Please wait your Component obx-mn-testmaint is being Created here !!!!
force ..\yo-rc.json
force C:\Users\saayare\yo-rc-global.json
create buildExtendedComponent.sh
create buildExtendedComponent.bat
create build.gradle
create gradle.properties
create package.json
create Jenkinsfile
create app.js
create gulpfile.js
create startCS.js
create metadata.js
create middleware.js
> Generating Libraries/|
```

- It will automatically generate the libraries for the components.
- At this point of time go to browser and navigate to <http://localhost:8002>. You will be able to component server home page like:



- Select the component which you want to reuse in your extension and paste it in **module.exports = []**; inside the **metadata.js** file



- Once done come back to main tab in cmdr where is waiting with question **Please modify the Metadata.js file before proceeding. Once done press 'y' to proceed?**
- On completing the above process, it will automatically generate the source folder for maintenance details screen and same process will be followed for summary screen as well.
- For this case we will be not able to see the component running locally as we must 2 components generated.
- To start the component, one needs to go inside the component and run it manually.

4.5 Data Segment

This section describes the process of creating the virtual page component using OBX.

Following are the steps needed to be followed:

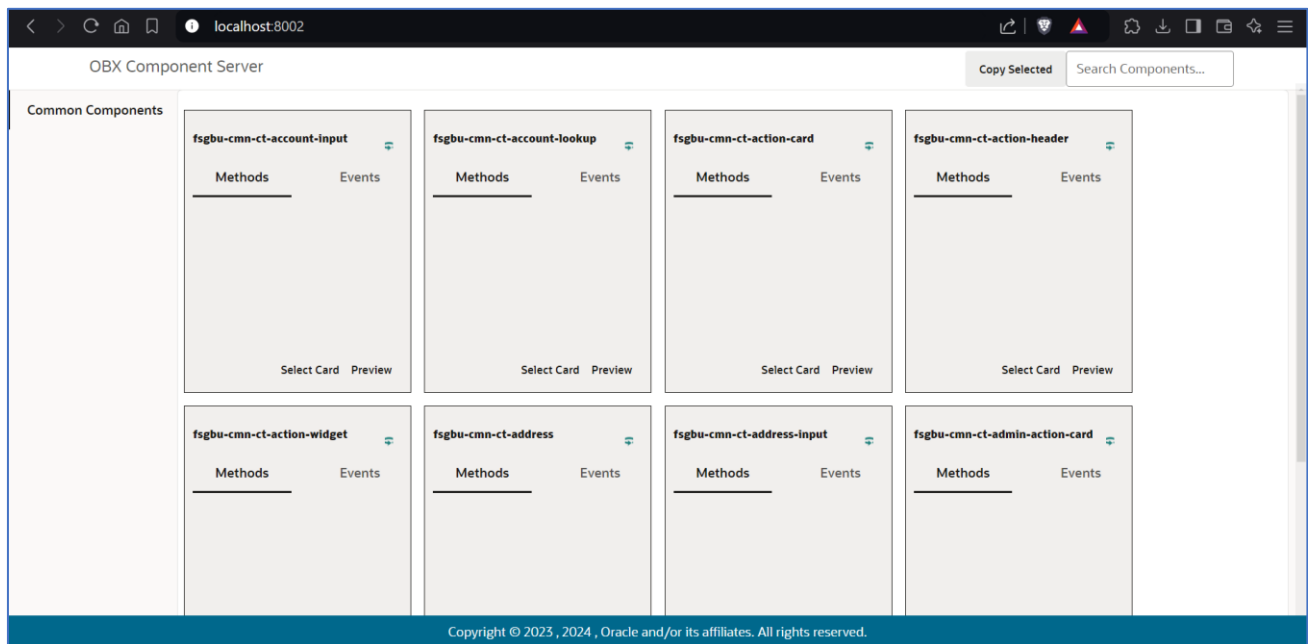
- Navigate to **extension_home** folder from cmdr.
- Use the command **obx ui -ds**.
- Once this command is executed, this will take you to next section where it will prompt other set of questions. Answer them accordingly to your setup and requirement.

```
Cmder
λ obx ui --ds
ORACLE BANKING EXTENSIBILITY WORKBENCH October 2024 (14.7.5.0.0)

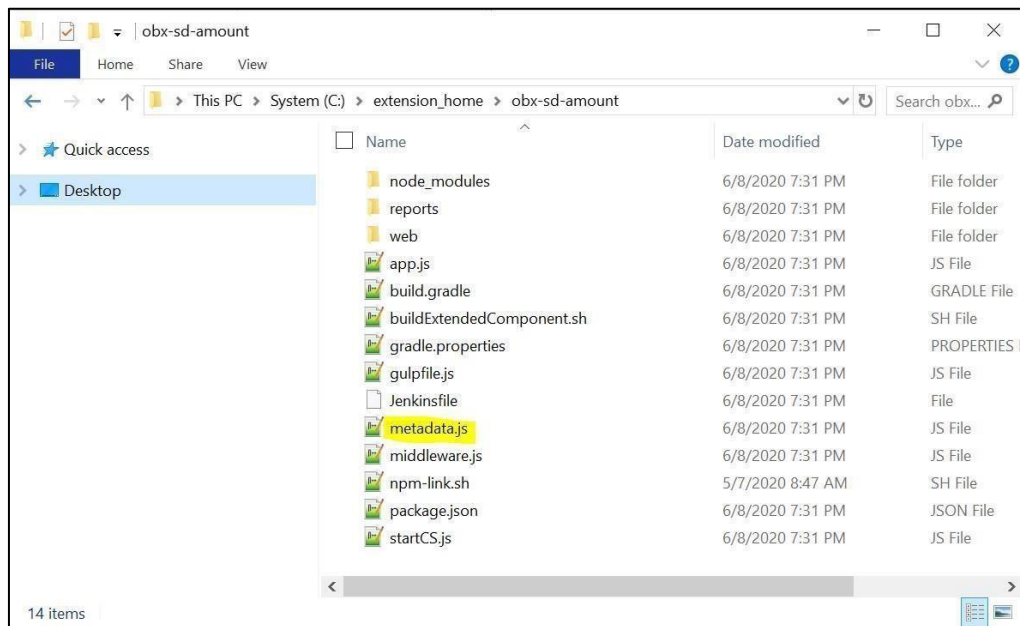
Copyright © 2024-2025, Oracle and/or its affiliates. All rights reserved.

? select the product family: Oracle Banking Extensibility Workbench
? Enter your Web Component Name (I'll prepend obx-ds- to it): dstest
? Enter the absolute path of xdl file: (D:\OBX\xdl-files\vessel.xdl) |
```

- It will automatically generate the libraries for the component to run locally and you will be also able to see new cmder tab opened where component server is running.
- At this point of time go to browser and navigate to <http://localhost:8002>. You will be able to component server home page like:



- Select the component which you want to reuse in your extension and paste it in **module.exports = []**; inside the **metadata.js** file.



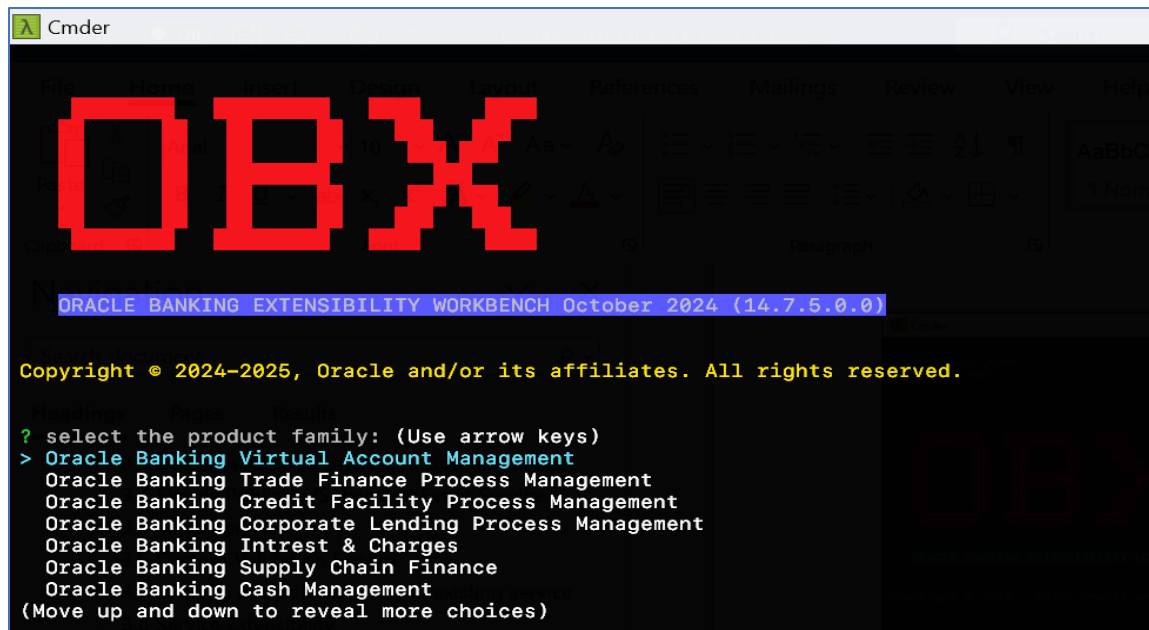
- Once done come back to main tab in cmdr where is waiting with question **Please modify the Metadata.js file before proceeding. Once done press 'y' to proceed?**
- On completing the above process, it will automatically generate the source folder now and open a new tab on cmdr where component will be running.
- Along with cmdr tab it will automatically open a tab on default browser as well with component rendered on the screen.

4.6 Dashboard Widget

This section describes the process of creating the simple standalone component using OBX.

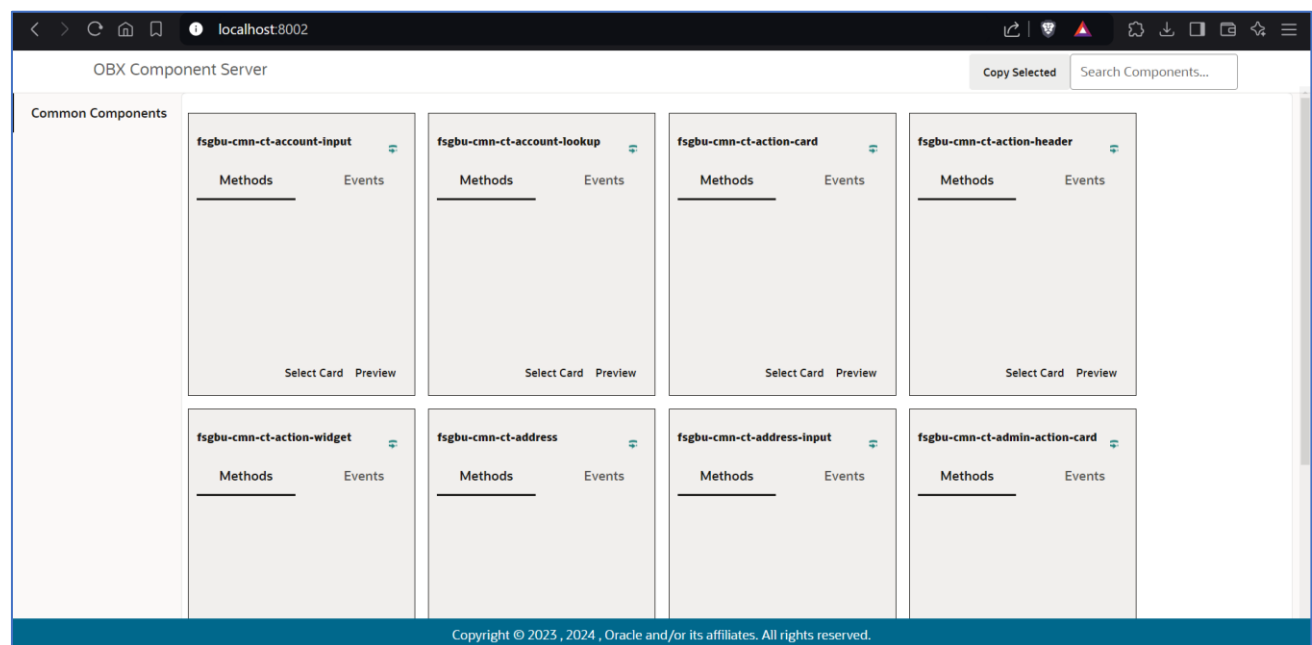
Following are the steps needed to be followed:

- Navigate to **extension_home** folder from cmdr
- Use the command **obx ui --wd**

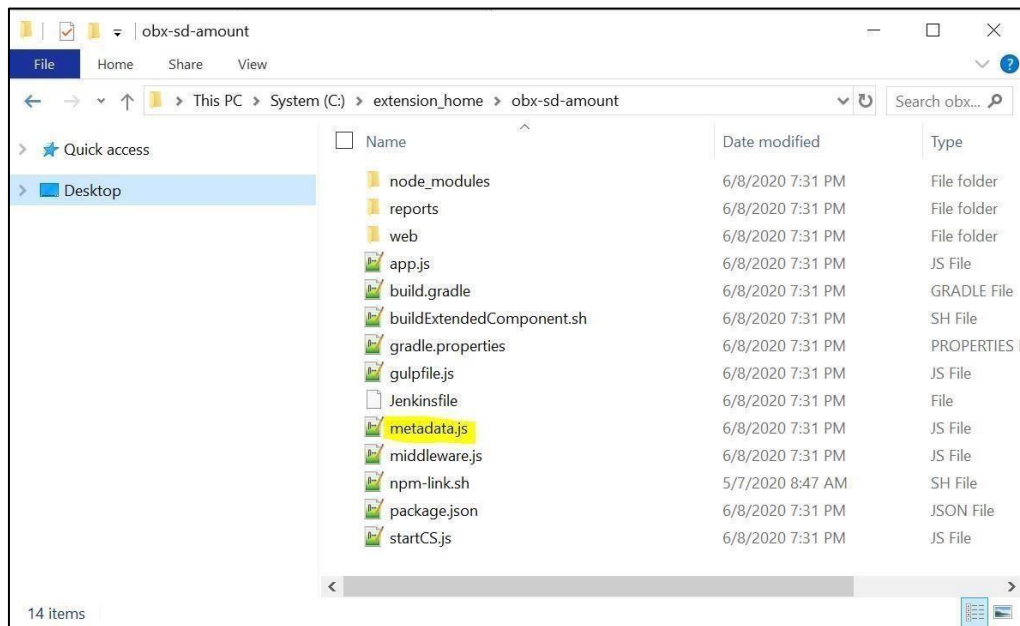


```
Coder
OBX
ORACLE BANKING EXTENSIBILITY WORKBENCH October 2024 (14.7.5.0.0)
Copyright © 2024-2025, Oracle and/or its affiliates. All rights reserved.
? select the product family: (Use arrow keys)
> Oracle Banking Virtual Account Management
Oracle Banking Trade Finance Process Management
Oracle Banking Credit Facility Process Management
Oracle Banking Corporate Lending Process Management
Oracle Banking Interest & Charges
Oracle Banking Supply Chain Finance
Oracle Banking Cash Management
(Move up and down to reveal more choices)
```

- Once this command is executed, this will take you to next section where it will prompt other set of questions. Answer them accordingly to your setup and requirement.
- It will automatically generate the libraries for the component to run locally and you will be also able to see new coder tab opened where component server is running.
- At this point of time go to browser and navigate to <http://localhost:8002>. You will be able to see component server home page like:



- Select the component which you want to reuse in your extension and paste it in `module.exports = []`; inside the `metadata.js` file.



- Once done come back to main tab in cmdr where is waiting with question **Please modify the Metadata.js file before proceeding. Once done press 'y' to proceed?**.
- On completing the above process, it will automatically generate the source folder now and open a new tab on cmdr where component will be running.
- Along with cmdr tab it will automatically open a tab on default browser as well with component rendered on the screen.

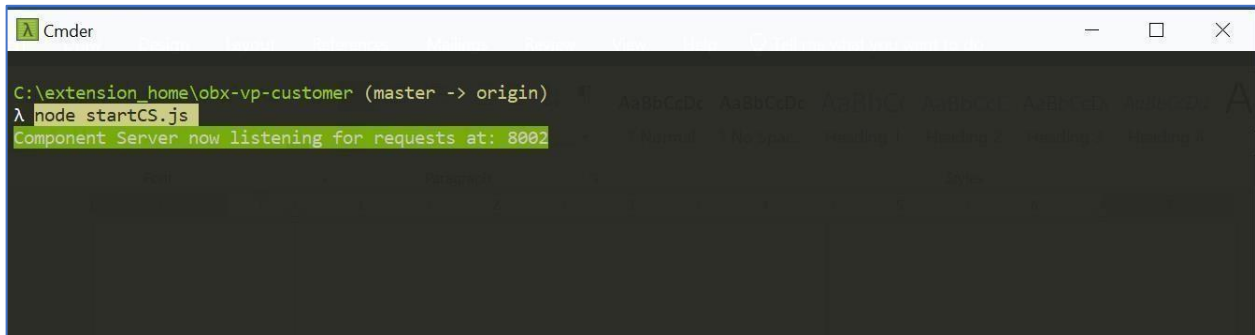


4.7 Running Component after Generation

This section describes the steps you need to follow to re-run the component created or generated earlier.

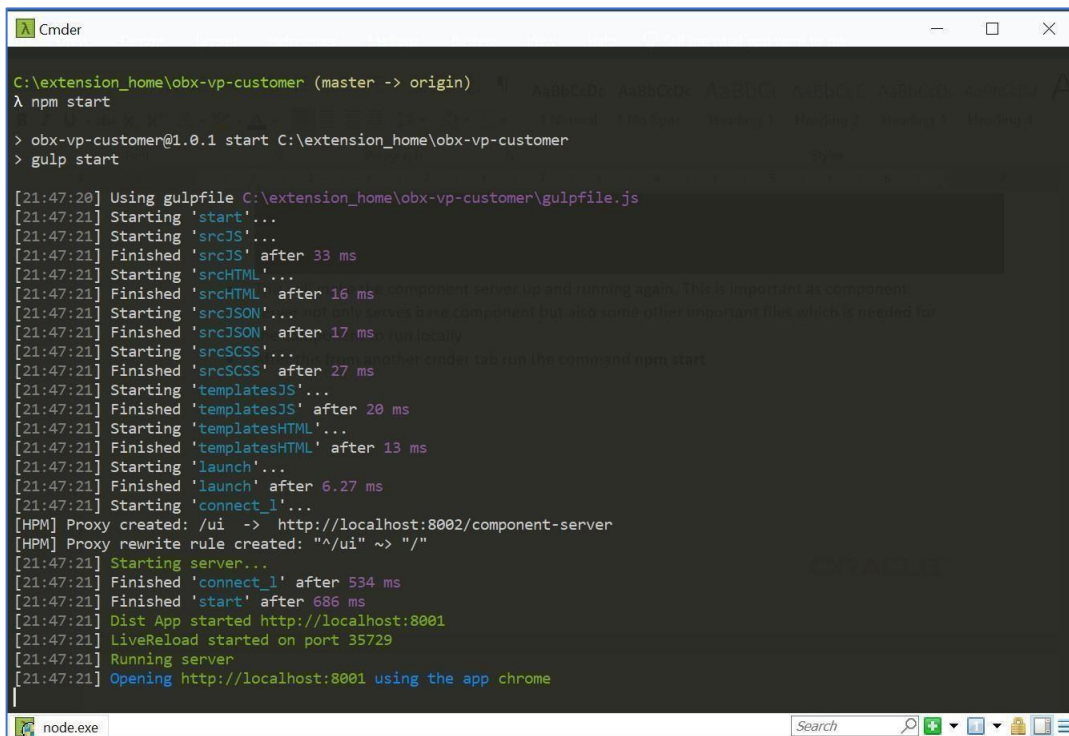
Please follow the below steps to do the same:

- Make sure you always have the component server rightly created.
- Open two tabs in the cmdr tool and navigate to component folder in both the tabs for example `\extension_home\obx-vp-customer`.
- From the first tab run the command **node startCS.js**



```
C:\extension_home\obx-vp-customer (master -> origin)
λ node startCS.js
Component Server now listening for requests at: 8002
```

- This will make the component server up and running again. This is important as component server not only serves base component but also some other important files which is needed for the component to run locally.
- After this from another cmdr tab run the command **npm start**



```
C:\extension_home\obx-vp-customer (master -> origin)
λ npm start
> obx-vp-customer@1.0.1 start C:\extension_home\obx-vp-customer
> gulp start

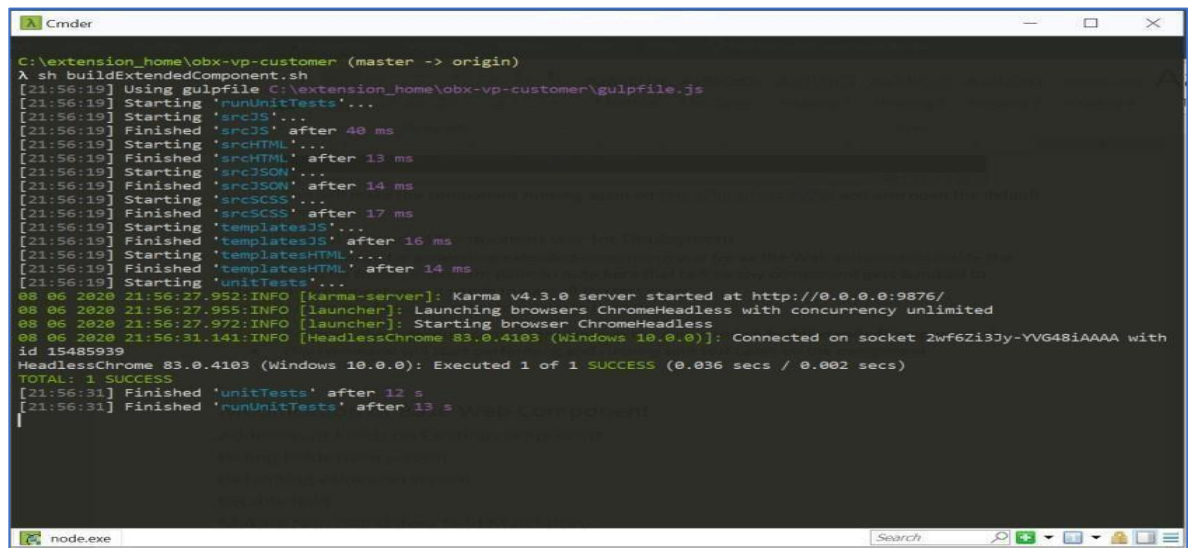
[21:47:20] Using gulpfile C:\extension_home\obx-vp-customer\gulpfile.js
[21:47:21] Starting 'start'...
[21:47:21] Starting 'srcJS'...
[21:47:21] Finished 'srcJS' after 33 ms
[21:47:21] Starting 'srcHTML'...
[21:47:21] Finished 'srcHTML' after 16 ms
[21:47:21] Starting 'srcJSON'...
[21:47:21] Finished 'srcJSON' after 17 ms
[21:47:21] Starting 'srcSCSS'...
[21:47:21] Finished 'srcSCSS' after 27 ms
[21:47:21] Starting 'templatesJS'...
[21:47:21] Finished 'templatesJS' after 20 ms
[21:47:21] Starting 'templatesHTML'...
[21:47:21] Finished 'templatesHTML' after 13 ms
[21:47:21] Starting 'launch'...
[21:47:21] Finished 'launch' after 6.27 ms
[21:47:21] Starting 'connect_1'...
[HPM] Proxy created: /ui -> http://localhost:8002/component-server
[HPM] Proxy rewrite rule created: "/ui" ~> "/"
[21:47:21] Starting server...
[21:47:21] Finished 'connect_1' after 534 ms
[21:47:21] Finished 'start' after 686 ms
[21:47:21] Dist App started http://localhost:8001
[21:47:21] LiveReload started on port 35729
[21:47:21] Running server
[21:47:21] Opening http://localhost:8001 using the app chrome
```

- This will make the component running again on <http://localhost:8001/> and also open the default browser.

4.8 Creating final Extended Component war for Deployment

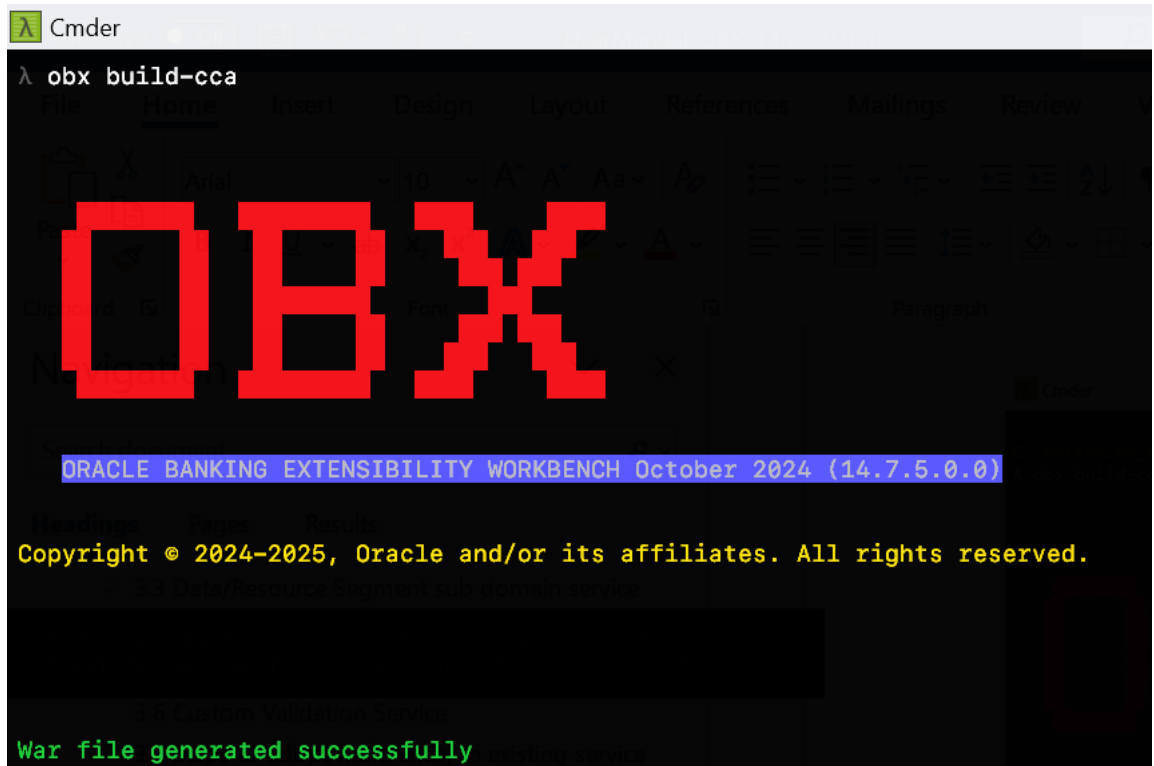
This is the final stage for generating extended-component war for all the Web components inside the `extension_home` folder. Important point to note here that before any component gets bundled to extended-component.war, it needs to pass all the test cases. Please perform the following steps to generate the war:

- Go inside the individual component and run the command `sh buildExtendedComponent.sh`. This command will start performing and running unit test cases on the component.



```
C:\extension_home\obx-vp-customer (master -> origin)
λ sh buildExtendedComponent.sh
[21:56:19] Using gulpfile C:\extension_home\obx-vp-customer\gulpfile.js
[21:56:19] Starting 'runUnitTests'...
[21:56:19] Starting 'srcJS'...
[21:56:19] Finished 'srcJS' after 40 ms
[21:56:19] Starting 'srcHTML'...
[21:56:19] Finished 'srcHTML' after 13 ms
[21:56:19] Starting 'srcJSON'...
[21:56:19] Finished 'srcJSON' after 14 ms
[21:56:19] Starting 'srcCSS'...
[21:56:19] Finished 'srcCSS' after 17 ms
[21:56:19] Starting 'templatesJS'...
[21:56:19] Finished 'templatesJS' after 16 ms
[21:56:19] Starting 'templatesHTML'...
[21:56:19] Finished 'templatesHTML' after 14 ms
[21:56:19] Starting 'unitTests'...
08 06 2020 21:56:27.952:INFO [karma-server]: Karma v4.3.0 server started at http://0.0.0.0:9876/
08 06 2020 21:56:27.955:INFO [launcher]: Launching browsers ChromeHeadless with concurrency unlimited
08 06 2020 21:56:31.141:INFO [HeadlessChrome 83.0.4103 (Windows 10.0.0)]: Connected on socket 2wf6Zi3Jy-YVG48iAAAA with
id 15485939
HeadlessChrome 83.0.4103 (Windows 10.0.0): Executed 1 of 1 SUCCESS (0.036 secs / 0.002 secs)
TOTAL: 1 SUCCESS
[21:56:31] Finished 'unitTests' after 12 s
[21:56:31] Finished 'runUnitTests' after 13 s
```

- Once the test cases are executed successfully it will create a folder inside the `extension_home` folder named `extended components`.
- Now we have to navigate back to `extension_home` folder and run the command **obx build-cca**.



```
λ obx build-cca

ORACLE BANKING EXTENSIBILITY WORKBENCH October 2024 (14.7.5.0.0)

Copyright © 2024-2025, Oracle and/or its affiliates. All rights reserved.

War file generated successfully
```

- This extended-component.war should be deployed in the same domain where application shell is deployed.

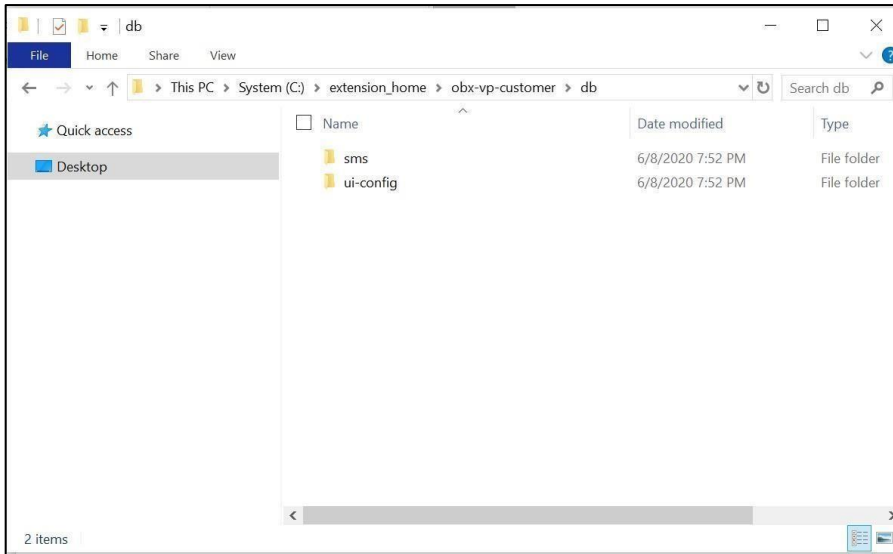
4.9 Creating final Extended Component war for Deployment.

This is the final stage for generating extended-component war for all the Web components inside the extension_home folder. Important point to note here that before any component gets bundled to extended-component.war, it needs to pass all the test cases. Please perform the following steps to generate the war:

- Go inside the individual component and run the command sh buildExtendedComponent.sh
- This command will start performing and running unit test cases on the component.

4.10 Understanding DB Scripts for Web Components

This section describes the significance of DB folder generate inside the web component folder. This is important as without executing these scripts extension web components will not be loaded inside application shell and even these components menu will be not listed in left navigation menu.



DB folder inside the web component consists of two folders sms and ui-config:

- **SMS:** The sms scripts consists of all the service activity, functional activity generated all out of the box from OBX.

```
INSERT INTO SMS_TM_UI_ACTIVITY (UI_ACTIVITY_CODE,DESCRIPTION,ICON,CCA_NAME,APPLICATION_ID,UI_ACTIVITY_TYPE)
VALUES ('OBX_UA_CUSTOMER',' OBX Customer',null,'obx-vp-customer','OBX','Virtual Page');

INSERT INTO SMS_TM_SERVICE_ACTIVITY (SERVICE_ACTIVITY_CODE,DESCRIPTION,CLASS_NAME,METHOD_NAME,APPLICATION_ID,SERVICE_TYPE,UI_ACTIVITY_CODE)
VALUES ('OBX_SA_CUSTOMER','OBX Customer','oracle.fsgbu.obx.customer.web.CustomerWebController','getCustomerById','OBX','Web API','OBX_UA_CUSTOMER');

INSERT INTO SMS_TM_UI_ACTIVITY_ACTIONS (ID,UI_ACTIVITY_CODE,SERVICE_ACTIVITY_CODE,LABEL)
VALUES ('OBX_AA_CUSTOMER','OBX_UA_CUSTOMER','OBX_SA_CUSTOMER','view');

INSERT INTO SMS_TM_MENU (ID,DESCRIPTION,SERVICE_ACTIVITY_CODE,APPLICATION_ID,PARENT_ID,SEQUENCE)
VALUES ('OBX_CUSTOMER','Customer',null,'OBX',null,1);
INSERT INTO SMS_TM_MENU (ID,DESCRIPTION,SERVICE_ACTIVITY_CODE,APPLICATION_ID,PARENT_ID,SEQUENCE)
VALUES ('OBX_CUSTOMER_DETAIL','Customer Detail','OBX_SA_CUSTOMER','OBX','OBX_CUSTOMER',null);

INSERT INTO SMS_TM_MENU_DESCRIPTION (ID,MENU_ID,LANGUAGE,DESCRIPTION)
VALUES ('OBX_CUSTOMER_ENG','OBX_CUSTOMER','ENG','Customer');
INSERT INTO SMS_TM_MENU_DESCRIPTION (ID,MENU_ID,LANGUAGE,DESCRIPTION)
VALUES ('OBX_CUSTOMER_DETAIL_ENG','OBX_CUSTOMER_DETAIL','ENG','Customer Details');

INSERT INTO SMS_TM_FUNCTIONAL_ACTIVITY (FUNCTIONAL_ACTIVITY_CODE,APPLICATION_ID,TYPE)
VALUES ('OBX_FA_CUSTOMER','OBX','O');

INSERT INTO SMS_TM_FUNC_ACTIVITY_DETAIL (ID,FUNCTIONAL_ACTIVITY_CODE,SERVICE_ACTIVITY_CODE)
VALUES ('OBX_FD_CUSTOMER','OBX_FA_CUSTOMER','OBX_SA_CUSTOMER');

COMMIT
```

- **UI Config:** This script should be compiled in ui-config schema. It maintains the ledger of all the extended components. App-shell uses this configuration to identify which components should be referred from extended-component war.

```
Insert into PRODUCT_EXTENDED_LEDGER (ID,CCA_NAME,CCA_TYPE,PARENT_CCA_NAME,PRODUCT_NAME)
select max(ID)+1,'obx-vp-customer','vp',null,'EXTENDED_COMPONENTS' from PRODUCT_EXTENDED_LEDGER;

Insert into PRODUCT_SERVICES_LEDGER (ID,PRODUCT_NAME,ENDPOINT_KEY,ENDPOINT_VALUE,REQUEST_TYPE,SERVICE_NAME)
select max(ID)+1,'OBX','CUSTOMER','/api/v1/customers','GET','obx-customer-service' from PRODUCT_SERVICES_LEDGER;

Insert into PRODUCT_SERVICES_CTX_LEDGER (ID,PRODUCT_NAME,SERVICE_NAME,SERVICE_CONTEXT_PATH,HEADER_APPID,CONTENT_TYPE,ACCEPT,USERID,BRANCH,SOURCE)
select max(ID)+1,'OBX','obx-customer-service','/','PKDSSRV001','application/json','application/json',null,null,null from PRODUCT_SERVICES_CTX_LEDGER;

COMMIT
```

5 Modification of Base Web Component

This feature of OBX enables users to create extensions which helps to modify the behaviour of existing component. It serves the one of the most common use cases from extensibility perspective. There are few important points which should be remembered before modifying the behaviour of existing components.

Important Points:

- Addition of fields can be done on various locations of base screen, but this breaks the CSS if not handled properly (Responsive Behaviour). In such cases it is always recommended to put additional fields at the bottom of other fields
- Wherever possible, use Data-segments to add additional field.
- In use case where you want to hide the fields from existing screen, always check whether the field is mandatory or not. If it is mandatory, then it should set before making it hidden on the screen. If not done so service calls make break
- Above point is also valid in case where you want to disable a field on the screen

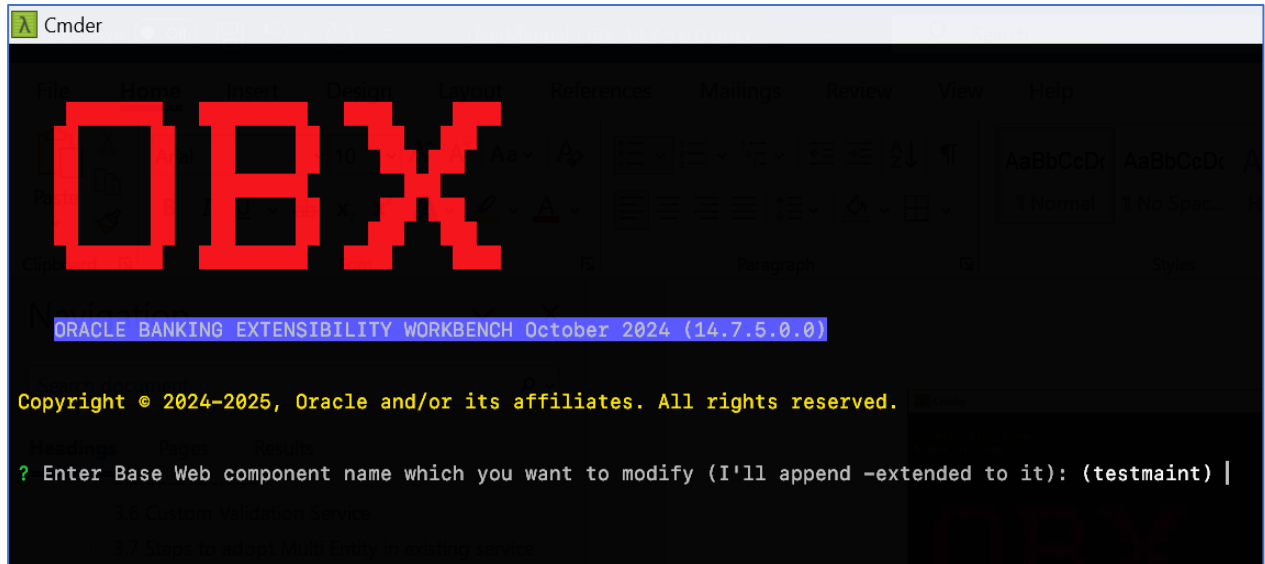
Following are the uses cases which can be achieved using modification of existing component

- Addition of Fields
- Hiding fields from screen
- Defaulting values on screen
- Disable field.
- Making Non-mandatory field Mandatory

5.1 Steps for Modification of Base Component

This section describes the steps to follow in case of adding fields on the existing screen. It is assumed that before using this command a developer knows the name of the base component in which he will be adding the additional fields. Following are the steps needed to be followed:

- Navigate to the extension_home folder from the cmdr.
- Execute the command **obx ui --mb**



```
Cmder

OBX

ORACLE BANKING EXTENSIBILITY WORKBENCH October 2024 (14.7.5.0.0)

Copyright © 2024-2025, Oracle and/or its affiliates. All rights reserved.

? Enter Base Web component name which you want to modify (I'll append -extended to it): (testmaint) |
```

- After above command is executed, it will prompt for the name of base component. Once given it will create a folder with base component name appending -extended at the end of it.
- Here also like above all the libraries are generated at runtime.
- Component generated contains the boiler plate or reference code, which helps to achieve the use case.

Again, db folder contains all the relevant scripts which is needed to be executed prior to see the component live and running in main application shell.

5.2 Process Workbench

- This screen is used for creation or modification of a process.
- We can add a new stage or modify an existing stage of an existing process.
- We can also upload a json-based dsl into the system using this screen.
- This screen will also help for any customization to do in workflow definition.
- This also provides to download JSON- based dsl based on whatever modifications done in UI.
- We can preview the flow-diagram of a modified or new added process.
- Any process, if modified, will be automatically incremented by 1 from the latest version.

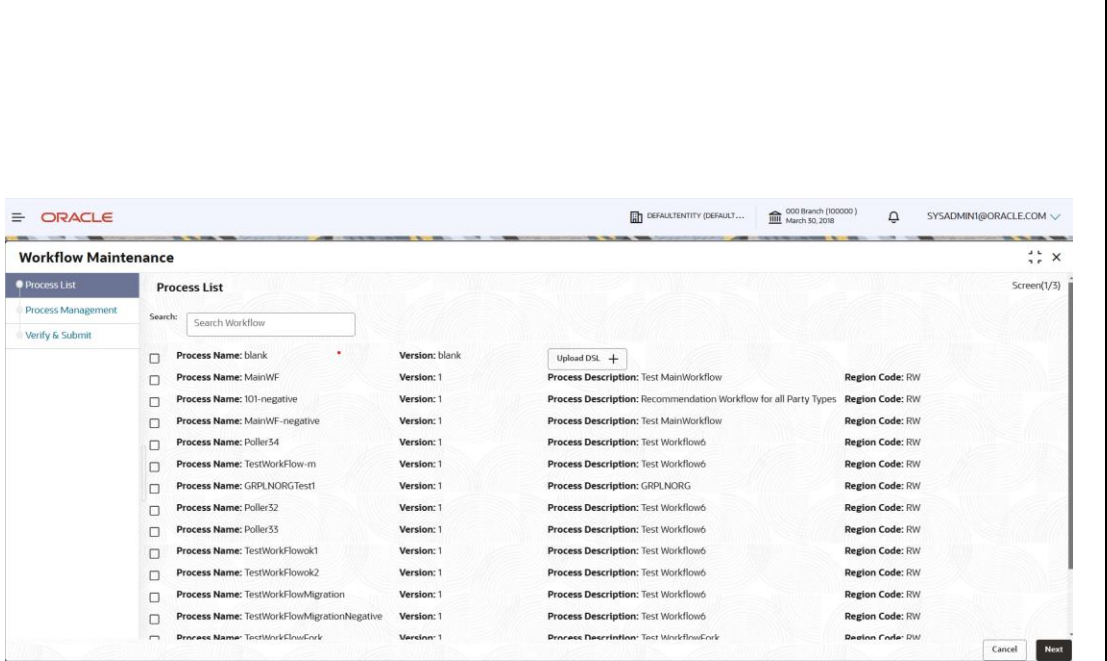
Screen 1:

Shows list of the processes

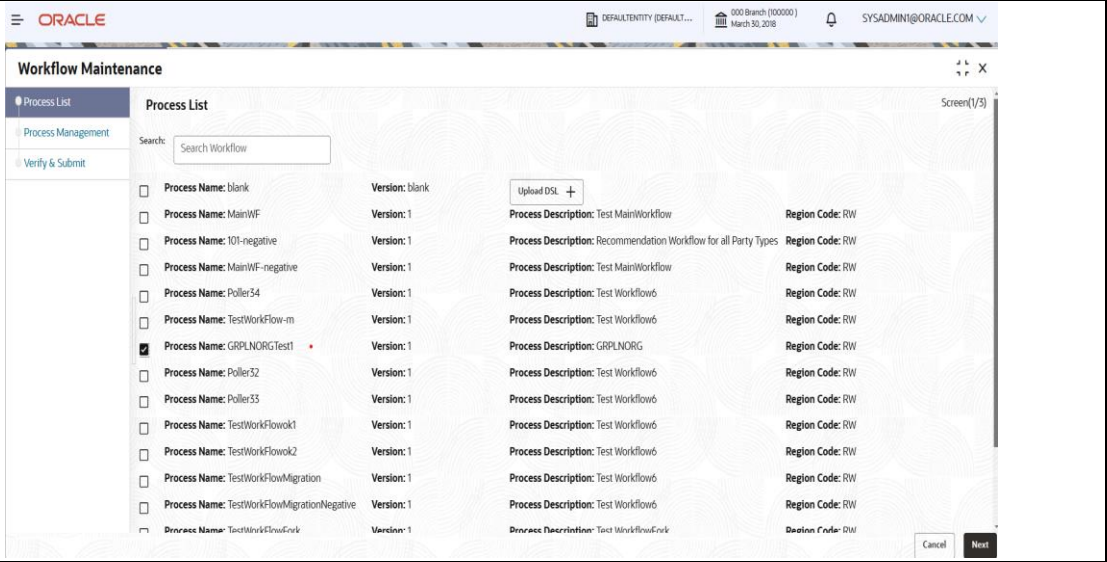
Upload DSL button:

Can be used to upload workflow in JSON format.

blank option - 1st row can be used for creating new process.



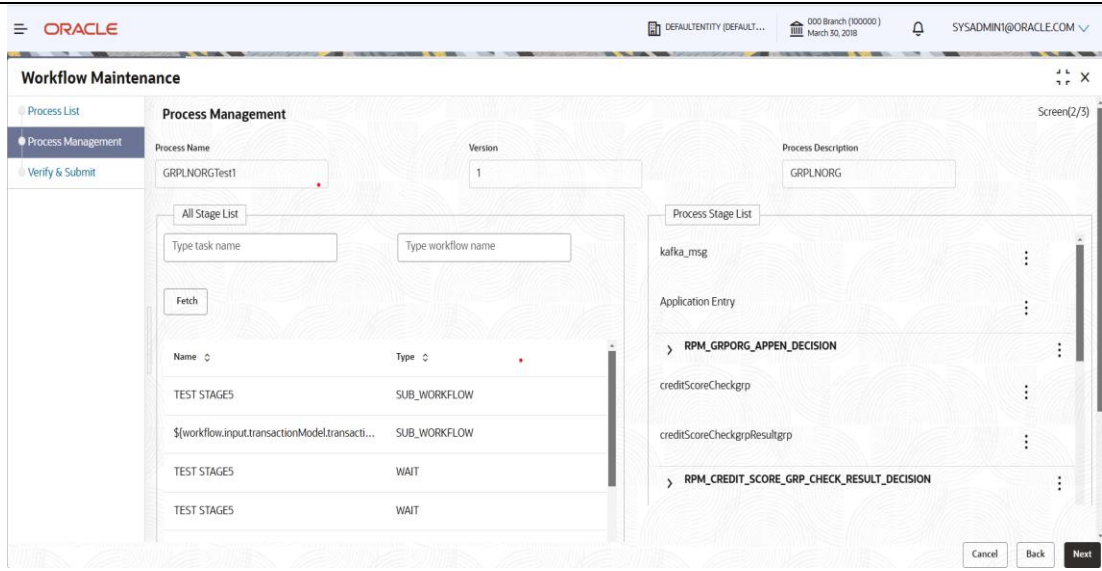
Selecting one Process



Screen 2:

Shows stages.

under the process which was selected on screen 1

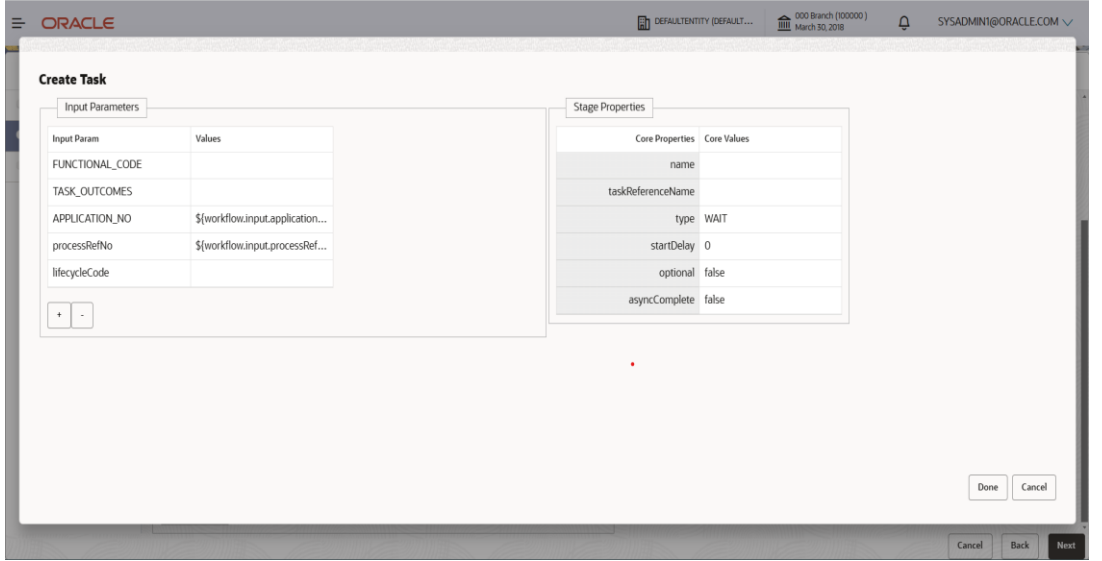


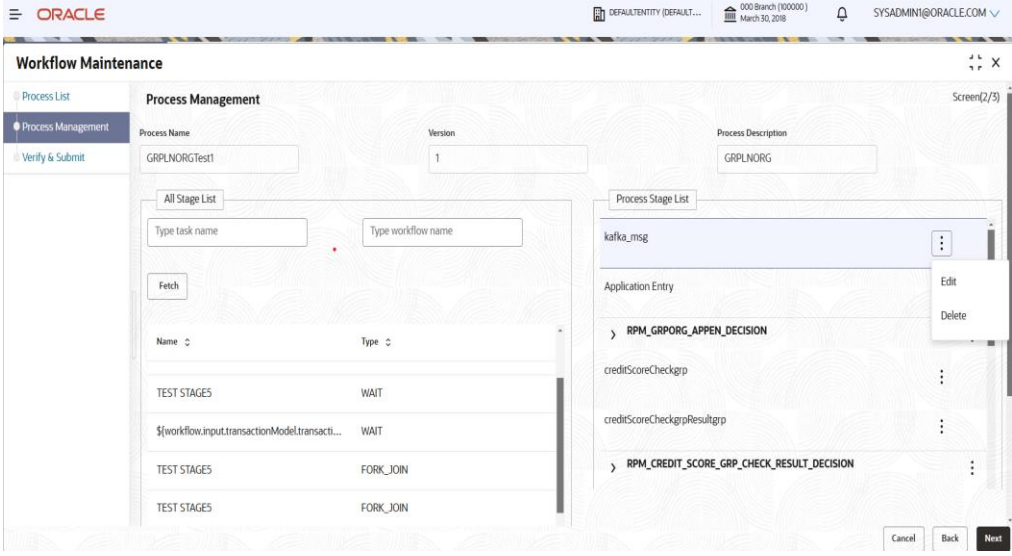
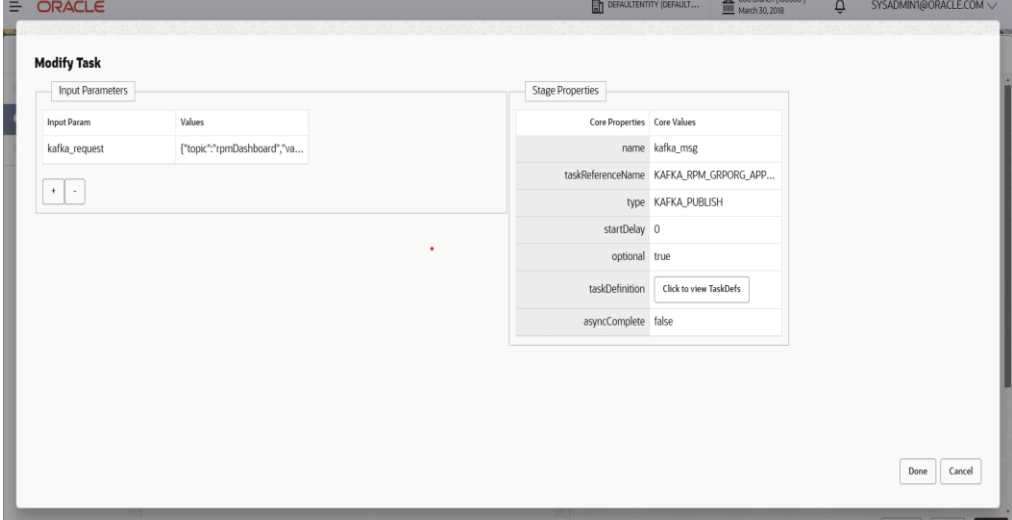
Create

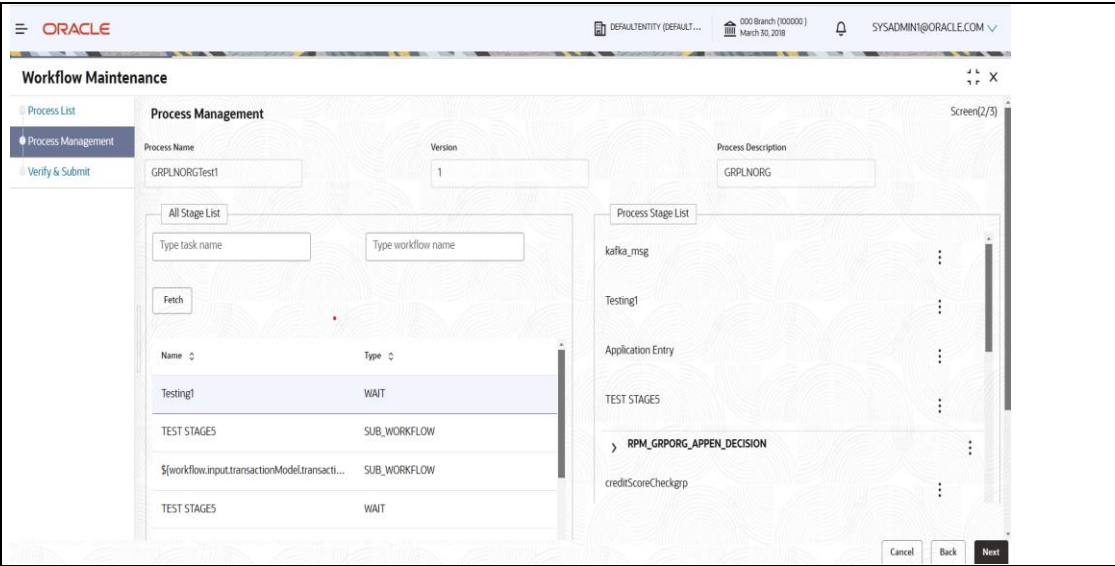
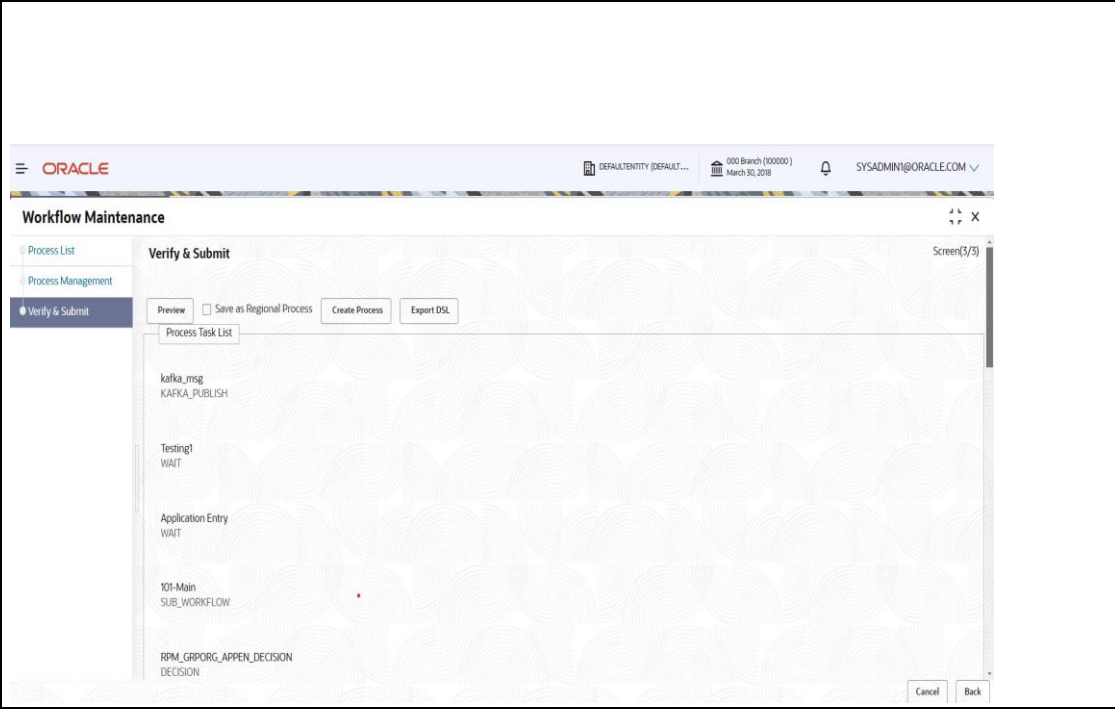
Stage button:

Used to create a new stage.

Dialog box for creating a new stage

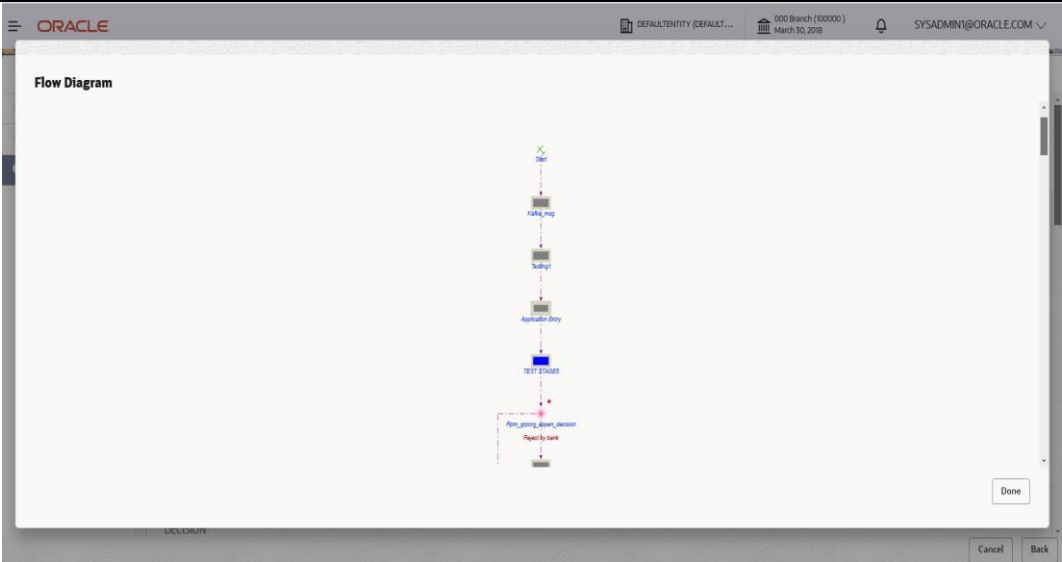


<p>5</p>	<p>We can edit/delete a particular stage in Process Stage list</p>	 <p>The screenshot shows the 'Workflow Maintenance' interface. Under 'Process Management', the 'Process Name' is 'GRPLNORGTST1' and 'Version' is '1'. The 'Process Description' is 'GRPLNORGT'. The 'All Stage List' table shows stages like 'TEST STAGES' with types 'WAIT', 'FORK_JOIN'. The 'Process Stage List' on the right lists stages including 'kafka_msg', 'RPM_GRPORG_APPEN_DECISION', 'creditScoreCheckgrp', and 'RPM_CREDIT_SCORE_GRP_CHECK_RESULT_DECISION'. A context menu is open over 'kafka_msg' with 'Edit' and 'Delete' options.</p>
<p>6</p>	<p>Dialogue box which opens. when we edit a particular stage</p>	 <p>The screenshot shows the 'Modify Task' dialog box. It has two tabs: 'Input Parameters' and 'Stage Properties'. The 'Input Parameters' tab shows a table with 'Input Param' and 'Values'. The 'Stage Properties' tab shows a table with 'Core Properties' and 'Core Values'. The 'Stage Properties' table includes fields like 'name', 'taskReferenceName', 'type', 'startDelay', 'optional', 'taskDefinition', and 'asyncComplete'.</p>

<div>7</div> <div>Drag and Drop Functionality</div> <div>Stage named "Testing1" from all stage list was dragged and dropped on the process stage list as shown here</div>	
<div>8</div> <div>Screen 3</div> <div>Preview: To preview flow diagram of the process selected</div> <div>Create Process:</div> <div>For creating a new process</div> <div>Export DSL:</div> <div>To Export DSL into a file in JSON format</div>	

9

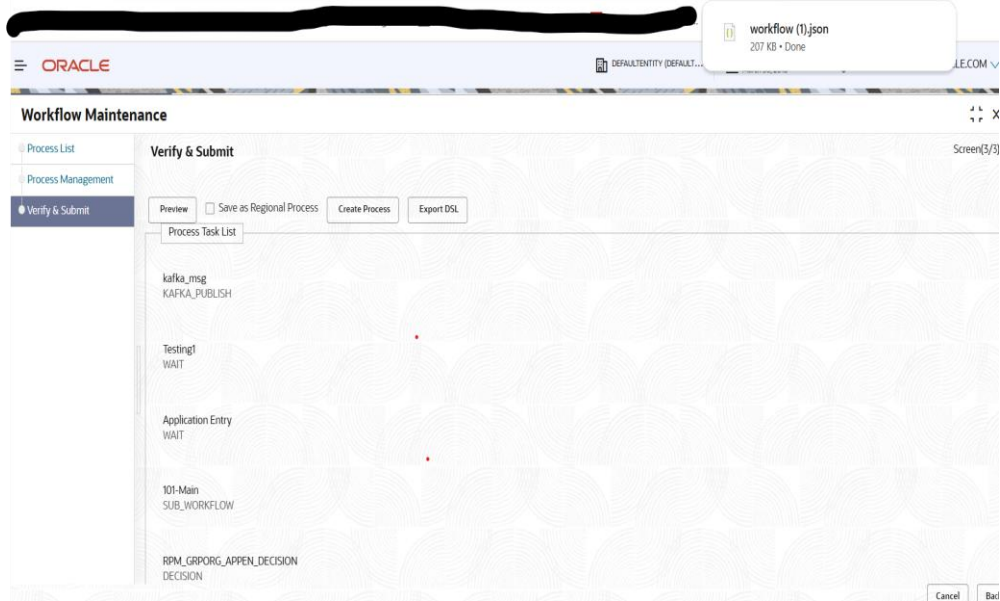
Flow Diagram of the process

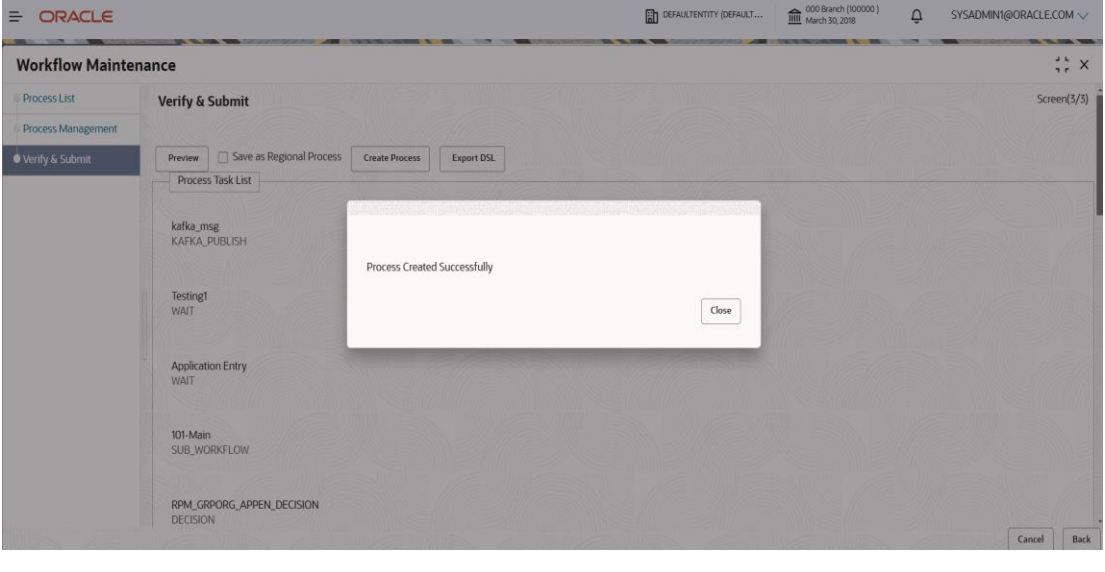
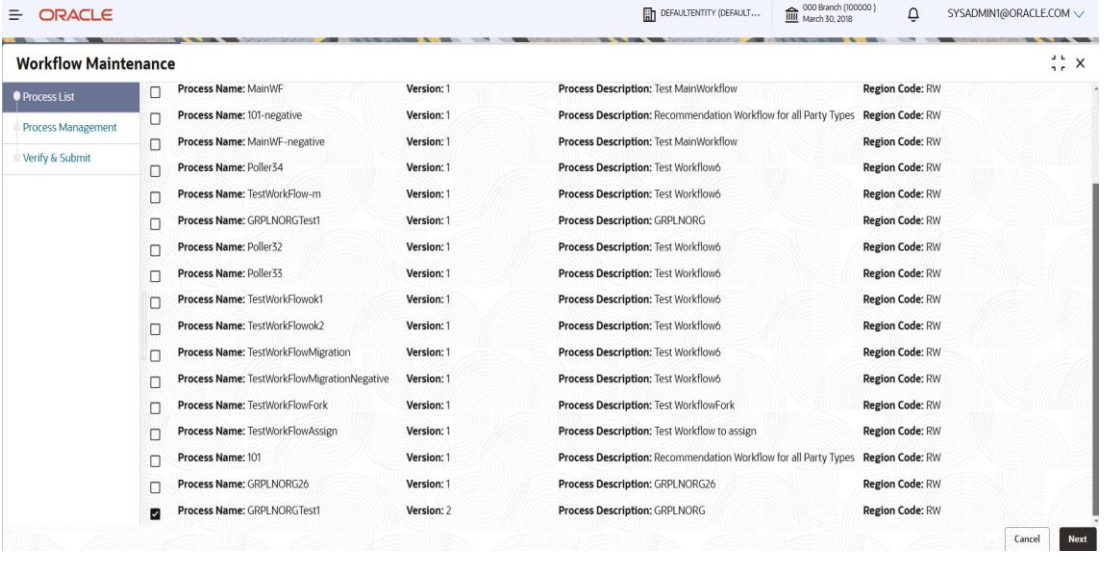


10

When "Export DSL" button is clicked.

The DSL gets downloaded in workflow(1).json file as shown



11	<p>When "Create Process" button is clicked.</p> <p>Process is Created.</p>	 <p>The screenshot shows the Oracle Workflow Maintenance interface. The 'Verify & Submit' tab is active. A modal dialog box in the center displays the message 'Process Created Successfully' with a 'Close' button. The background interface includes a sidebar with 'Process List', 'Process Management', and 'Verify & Submit' options. The main area shows a 'Process Task List' with tasks like 'kafka_msg', 'KAFKA_PUBLISH', 'Testing1', 'WAIT', 'Application Entry', 'WAIT', '101-Main', 'SUB_WORKFLOW', and 'RPM_GRPORG_APPEN_DECISION DECISION'.</p>
12	<p>Version is updated.</p> <p>when the process is created successfully</p>	 <p>The screenshot shows the Oracle Workflow Maintenance 'Process List' screen. It displays a table of processes with columns for checkboxes, Process Name, Version, Process Description, and Region Code. The process 'GRPLNORGTest1' is highlighted with a checked checkbox and shows 'Version: 2'. Other processes listed include 'MainWF', '101-negative', 'MainWF-negative', 'Poller34', 'TestWorkflow-m', 'GRPLNORGTest1', 'Poller32', 'Poller33', 'TestWorkFlowok1', 'TestWorkFlowok2', 'TestWorkFlowMigration', 'TestWorkFlowMigrationNegative', 'TestWorkFlowFork', 'TestWorkFlowAssign', '101', 'GRPLNORG26', and 'GRPLNORGTest1'.</p>

5.3 OBX Update Command

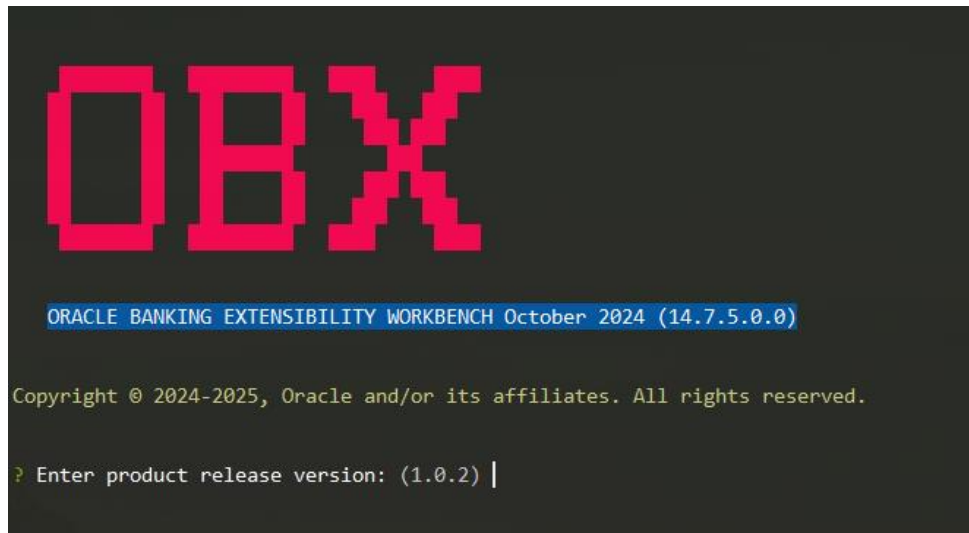
This section helps in migrating the artifacts from previous version of OBX to latest. This is applied to both services and web components. Following sections describes the steps to be followed to upgrade the existing artifacts:

5.3.1 Service Update

To migrate services developed in previous versions of OBX to latest please follow the below steps:

- Navigate to service specific folder inside the extension_home directory.
- Execute the command obx service-update.

- Provide the relevant product release version number.
- Once provided it will automatically change the build.gradle file and service is ready to be built with latest dependencies.



```

OBX

ORACLE BANKING EXTENSIBILITY WORKBENCH October 2024 (14.7.5.0.0)

Copyright © 2024-2025, Oracle and/or its affiliates. All rights reserved.

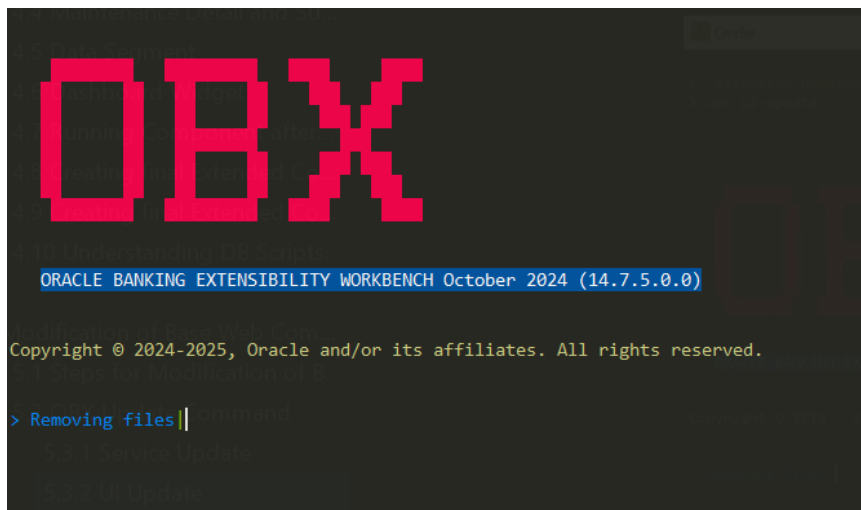
? Enter product release version: (1.0.2) |

```

5.3.2 UI Update

To migrate services developed in previous versions of OBX to latest please follow the below steps:

- Navigate to UI (Web Component) specific folder inside the extension_home directory
 - o Execute the command `obx ui-update`.



```

OBX

ORACLE BANKING EXTENSIBILITY WORKBENCH October 2024 (14.7.5.0.0)

Copyright © 2024-2025, Oracle and/or its affiliates. All rights reserved.

> Removing files||

```

- This command will automatically start removing old libraries without changing the source folder. This help will help you retaining the business logic already written in web component.
- Once done and executed successfully you will see the below message.


```
create web\js\util\resources\trade\nls\ar\bundle.js
create web\js\util\resources\trade\nls\fr\bundle.js
-----Component updated successfully-----
```

- Now to run the command with new libraries run below command sequentially:
- **sh npm-link.sh** – it will create new node module folder inside the component with latest modules and dependencies.
- **node startCS.js** - Open a new tab in cmdr and navigate to same web component directory and run command node startCS.js.
- **npm start** – From the main tab, where we executed npm-link command run the command npm start, it will automatically run the web component with latest libraries and launch it on the browser as well.

5.4 In-Scope DS

- Additional of fields at any desired location in an existing data-segment is supported now.
- Data will be stored in separate custom schema.
- In-scope Data segment can be used for addition of new fields. (using jquery, at any position, we can add the field).

Example of In-Scope DS (Additional fields):

- Include the hooks required in js and html of base components accordingly.
- Run the command “obx ui --af” for adding fields in extended components.
- Include the additional field in “self.data”

```
self.data = {

    "newField": ko.observable("")

};
```

- Subscribe it to change handler.
self.data.newField.subscribe(self.changeHandler);
- Use jquery to insert it in the location you want to add the fields.

```
var element = context.properties.data.payload.homeBranch;
$('#homeBranch').parent().parent().parent().append($('#ui-ex-divnewField').parent());
```

5.5 OBX Release Command

This command is used to check all the available features bundled with OBX version installed on the machine. To run this command, navigate to extension_home folder and run the command: **obx release**.

```
λ obx release

ORACLE BANKING EXTENSIBILITY WORKBENCH October 2024 (14.7.5.0.0)

Copyright © 2024-2025, Oracle and/or its affiliates. All rights reserved.

Release Notes

This Release offers a comprehensive standalone solution for creating extension for OBMA based products. Following are the major features

-> Workflow Data Segment has been included (--wfds)
-> Rsov2 has been introduced (--rsov2)
-> Endpoint Maintenance
-> Pre/post hooks added for some common CCAs
-> Additional buttons provision in task screens
-> Provision to extend configJSON.js file

[1] Workflow Data Segment has been included (--wfds)

-> Creation of Workflow Data Segment service and UI component is supported now.

[2] Rsov2 has been introduced (--rsov2)

-> For Nov patchset innovation - RSOv1 is discontinued and RSOv2 should be adopted for all customizations for maintenance services. OBX will now support RSO v2 b
e and UI component generation

[3] Endpoint Maintenance

-> As part of this Extensibility, we are introducing a new table PRODUCT_SERVICE_EXT_LEDGER. This table helps to override the existing endpoint
```

6 Extending Product Data Segments with Additional Fields

6.1 Additional Fields Maintenance

This screen is used to maintain the additional fields for a transaction screen. To process this screen, type Additional Fields Maintenance in the Menu Item Search located at the left corner of the application toolbar and select the appropriate screen (or) do the following steps:

- From Home screen, click Core Maintenance. Under Core Maintenance, click Additional Fields Maintenance.
- The Additional Fields Maintenance screen is displayed.

Additional Fields Maintenance

Additional Fields Details

Component Name	Unique Identifier/Product Code	Description	Application ID
fighu-ob-crm-ds-additional-ty	01	Additional Fields	OSO

UI Key: fighu-ob-crm-ds-additional-fields@01

Construct Additional Fields MetaData

Se	Field ID	Field Label	Category	Field Type	Edit Properties	Mandatory	Is Unique?
<input type="checkbox"/>	UOF_TEST	TEST	UDF	TEXT	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Construct Validation MetaData

Se	Validation Name	Validation Template To Use	Custom Error Message	Edit Arguments
No data to display				

Buttons: Audit, Cancel, Save

Figure 1: Additional Fields Maintenance Screen

- Specify the details in the Additional Fields Maintenance screen. For more information on fields, refer to table Field Description – Additional Field Maintenance

Field Description – Additional Field Maintenance

Field	Description
Component Name	Specify the data segment name as component name. NOTE: By default, the value fsgbu-ob-cmndsadditional-fields is displayed , which is the Common Core Data Segment that displays the maintained additional fields. It will fetch the corresponding maintained record for Additional Fields by querying with uiKey = DataSegmentName @ ProductCode.
Product Code	Specify the function code as product code.
Product Name	Displays the product name of the specified product code.
Description	Displays the description as Additional Fields .
Application ID	Displays the Application ID.
+ icon	Click this icon to add a new row.
– icon	Click this icon to delete a row, which is already added.
Construct Additional Fields MetaData	Specify the fields.

Select	Check this box to select a row.
Field ID	Specify the Field ID.
Field Label	Specify the field label.

Category	Specify the category.
Field Type	Specify the field type.
Edit	Select if a value needs to be inputted in the additional field.
Mandatory	Select if the input value is mandatory in the additional field.
Construct Validation MetaData	Specify the fields.
Select	Check this box to select a row.
Validation Name	Specify the validation name.
Validation Template to Use	Specify the template to be used for validation.
Custom Error Message	Specify the custom error message to be displayed.
Edit Arguments	Select if arguments needs to be edited in the additional field.

- Click Save to add the additional field in the maintenance work table.
(CMC_TW_ADDDT_ATTR_MASTER).
- NOTE: Once it is approved, the data will persist in the master table. Currently, Mobile Number and Date are added as additional fields. In addition, the validation is added for Date.
- Sign in with different user ID since maker will not be able to approve the records with the same user ID.

Figure 2: Additional Fields Maintenance Records

- Map the new data segment for the function code. Make sure that the data is present in **CMC_TM_SCREEN_DS_MAPPING**.

- NOTE: Once the additional fields are added for a particular function code, a separate data segment will be enabled in the transaction screen for Additional Fields.
- Click Submit, to save the transaction data of additional fields to the CMC_TB_ADDDT_ATTR_DATA. In addition, the following actions have been performed from service side.
- Fetch record through inter-service call to additional attributes service in common transaction with record ID.
- Append the field data to the main payload for the ejlogging.

```
{
  "data": {
    "addDtIs": {
      "signatureVerifyIndicator": "Y",
      "hostStatus": null,
      "hostMultiTripId": null,
      "txnBranchCcy": "GBP",
      "txnBranchDate": "2020-03-25T18:30:00.000+0000",
      "txnType": "C",
      "cashInOutIndicator": "I",
      "ejLoggingRequired": null,
      "ejTxnAmtMapping": "TO",
      "ejTxnCcyMapping": "TO",
      "adviceName": null,
      "orchestratorId": null,
      "rsp": null,
      "isReversal": "N",
      "isAdvice": "N",
      "reversalButton": "N",
      "ignoreApproval": false,
      "ignoreWarning": false,
      "isExternal": false
    },
    "txnDtIs": {
      "functionCode": "1401",
```

```

        "txnBranchCode": null,
        "txnBranchCcy": null,
        "txnBranchDate": null,
        "requestStatus": "COMPLETED",
        "assignmentMode": null,
        "txnId": "f6b36a91-889d-4505-aac0-d7b98484d098",
        "txnRefNumber": "989124345493245",
        "tellerSeqNumber": null,
        "overrideConfirmFlag": null,
        "supervisorId": null,
        "onlineOfflineTxn": null,
        "userComments": null,
        "authoriserComments": null,
        "eventCode": null, "accountType":
        "UBS"
    },
    "dataPayload": {
        "datasegment": null,
        "fromAccountAmt": 100,
        "fromAccountCcy": "GBP",
        "toAccountCcy": "GBP", "beneficiaryName":
        null,
        "beneficiaryAddress1": null,
        "beneficiaryAddress2": null,
        "beneficiaryAddress3": null,
        "beneficiaryAddress4": null,
        "identificationType": null,
        "identificationNumber": null,
        "exchangeRate": 1,
        "recievedAccount
        Ccy": null,

```

"recievedAccount
Amt": null,
"totalCharges": null,
"cashAmount":
100,
"netAccountCcy": null, "netAccountAmt":
null,
"narrative": "Cash Deposit",
"txnControllerRefNo": null, "recordId":
"f6b36a91-889d-4505-
aac0d7b98484d098", "cashAmtCcy":
null, "cashAmt":
null,
"chequeDate": null,
"chequeNumber": null,
"eventCode": null,
"ejId": null,
"emailId": null,
"fromAccountBranch": "000",
"fromAccountNumber": null,
"mobileNumber": null,
"orginalExchangeRate": null,
"payee": null,
"productCode": null, "reversalDate":
null,
"stationId": null,
"toAccountBranch": "000",
"toAccountNumber": "00000008010010",
"toAccountAmt": 100,
"txnBranchCode": "000",
"functionCode": null,
"txnCustomer": null,

```

    "tellerId": null,
    "txnDate": 1585161000000,
    "txnRefNumber": "9892566557744",
    "txnSeqNumber": null,
    "uniqueIdentifierNumber": null,
    "uniqueIdentifierType": null,
    "userRefNumber": null,
    "valueDate": null,
    "versionNumber": null,
    "referenceNumber": null,
    "createdBy": null, "createdTs":
    null,
    "updatedBy": null, "updatedTs":
    null,
    "demDtls": [],
    "fxInDemDtls": null,
    "fxOutDemDtls": null,
    "prcDtls": [],
    "addDtls": null,
    "txnDtls": null,
    "overrideDtls": null, "batchTableDetails":
    null,
    "cmcAddlFields": [
{
    "id": "OTH_passprt",
    "label": "Passport No",
    "type": "TEXT", "value":
    "43243"
},
{
    "id": "UDF_aadhar",
    "label": "Aadhar",

```

```

        "type": "TEXT",
        "value": "1243"
    },
    {
        "id": "TMIS_toDate",
        "label": "To Date",
        "type": "DATE",
        "value": ""
    },
    {
        "id": "TMIS_fromDate",
        "label": "From Date",
        "type": "DATE",
        "value": ""
    }
},
"extDetails": null,
"warDtls": [], "authoriserDtls":
[]
},
"errors": null,
"warnings": null,
"informations": null,
"authorizations": null,
"paging": ""
}

```

6.2 Populating Data in Corresponding Fields From UI

Unlike the other transaction screen data-segments, the ejlogged data is not required. Instead, two GET calls that happen during screen launch fetches all the details. To fetch the corresponding **Additional-FieldsMaintenance** screen record based on which it will display the maintained fields for this function code.

Endpoint:

CORE.GET_CMC_ADDITIONAL_ATTRIBUTES Request

URL - <http://whf00peb.in.oracle.com:8003/api-gateway/cmc-additional-attributeservices/cmcadditional-attributes-services/?uiKey=fsgbu-ob-cmn-ds-additionalfields@1006>

Sample Response:

```
{
  "data": [
    {
      "keyId": "33347926-842b-4232-af31-8c1b59612244",
      "makerId": "ABHINAV",
      "makerDateStamp": null,
      "checkerId": null,
      "checkerDateStamp": null,
      "modNo": 1,
      "recordStatus": "O",
      "authStatus": "A",
      "onceAuth": null,
      "doerRemarks": null,
      "approverRemarks": null,
      "links": [
        {
          "rel": "self",
          "href": "http://10.40.158.157:8005/cmcadditional-attributeservices/cmcadditionalattributes-services/33347926-842b-4232af318c1b59612244"
        }
      ],
      "description": "Additional Fields",
      "fieldMetaData": [
        {
          "id": "OTH_Mobile",
          "label": "Mobile Number",
          "type": "NUMBER",
          "required": true
        },
        {
          "id": "OTH
```



```

        "_From\\", "label\\": "\\From
Date\\", "type\\": "\\DATE\\", "required\\": true}, {"id\\": "\\OTH_To_D
ate\\", "label\\": "\\To
Date\\", "type\\": "\\DATE\\", "required\\": true}], "uiKey": "fsgbuob-cmn-
ds-additional-fields@1006", "validationMetaData":
"[{"id\\": "\\", "validateMethod\\": "\\compareFromToDates\\", "type\\": "\\
\\", "args\\": [{"ty
pe\\": "\\FIELD\\", "value\\": "\\OTH_From\\", {"type\\": "\\FIELD\\", "value\\
": "\\OTH_To_Date\\
}], "errorMsg\\": "Error Date 1 must be &gt; Date
2\\", "validationName\\": "\\Date
Validation\\"]]",
"applicationId": "OBTFFPM"
} ],
"paging": {
"totalResults": 1,
"links": {
"next": null,
"prev": null
}
}
}

```

6.3 Fetching the Saved Values

You can fetch the values saved for each field during the transaction. Endpoint:

CORE.GET_ADDITIONAL_ATTRIBUTES.

Request URL:

<http://whf00peb.in.oracle.com:8003/api-gateway/cmc-additionalattributesservices/additionalattributes/?uiKey=fsgbu-ob-cmn-dsadditionalfields@1006&dataReferenceKey=00a01dfd-0d6f-4400-a9c5-0f56551165e4>

Sample Response:

```
{
```

```

"ExtensibleDTO": [
{
    "id": "1644022a-179e-429b-82c8-873761c3ac74",
    "uiKey": "fsgbu-ob-cmn-ds-additional-fields@1006",
    "dataReferenceKey": "00a01dfd-0d6f-4400-a9c5-
0f56551165e4",
    "fieldMetaDataVersion": "1",
    "fieldData": [
        {
            "id": "OTH_Mobile",
            "label": "Mobile Number",
            "type": "NUMBER", "value":
            "678688789"
        },
        {
            "id": "OTH_From",
            "label": "From Date",
            "type": "DATE", "value":
            "678688789"
        },
        {
            "id": "OTH_To_Date",
            "label": "To Date",
            "type": "DATE",
            "value": null
        }
    ],
    "applicationId": "OBREMO"
}
]
}

```

7 Action URL and Static Tag Maintenance

7.1 Action URL Maintenance

Endpoints are maintained in cmn-transaction-services for the specific transaction based on function code. The operation has to be maintained as action URL in table SRV_TB_BC_ACTIONS_URL. Action URL will be called from all the domain services based on function code and action (like OPENCHECK, CREATE, OVERRIDE, REVERSAL, PENDING_APPROVAL, or AUTHORIZE). The database details are as follows:

Schema: BRANCHCOMMON

Table: SRV_TB_BC_ACTIONS_URL

If the action URL is not maintained for the specific operation of the particular transaction, the error message will be displayed as Action URL not maintained. Error code is maintained in ERTB_MSGS as RM-BC-UR-01.

7.2 Static Tag Maintenance

Static tag is maintained for accounting, till update, and debit-credit for each transaction based on the function code in table SRV_TB_TX_STATIC_TAGS.

The database details are as follows:

Schema: TRANSACTION

Table: SRV_TB_TX_STATIC_TAGS

TILL_TAGS, DRCCR_TAGS and ACCOUNTING_TAGS are maintained as JSON structure. Static tags will be fetched from cmn-transaction-services based on function code. If it is not maintained for the function code, the transaction will be failed.

8 Extensibility Use Cases for OBORN Servicing

8.1 New Transaction Screen – 1499 (Exact Clone of 1401)

For this use case, you need to ensure data is present in the tables similar to 1401. The below mentioned tables need to be checked in SMS schema:

- SMS_TM_MENU
- SMS_TM_MENU_Description
- SMS_TM_SERVICE_ACTIVITY
- SMS_TM_FUNCTIONAL_ACTIVITY
- SMS_TM_FUNC_ACTIVITY_DETAIL
- SMS_TM_ROLE_ACTIVITY

- SMS_TM_UI_ACTIVITY

The below mentioned tables need to be checked in common core schema:

- CMC_TM_SCREEN_CLASS
- CMC_TM_SCREEN_DS_MAPPING

The below mentioned tables need to be checked in branch common schema:

- SRV_TM_BC_FUNCTION_INDICATOR
- SRV_TM_BC_FUNCTION_CODE
- SRV_TM_BC_FUNCTION_PREF
- SRV_TM_BC_FUNCTION_PREF_DTLS
- SRV_TM_BC_BRANCH_ACCOUNTING
- SRV_TM_MENU_CONFIG
- SRV_TB_BC_ACTIONS_URL

The below mentioned tables need to be checked in transaction schema:

- SRV_TB_TX_STATIC_TAGS

Cash Deposit Clone

Account Number *
 Transaction Amount *
 Exchange Rate
 Total Charge Amount
 Narrative *

Transaction Amount: GBP100.00
 Account Amount: GBP100.00
 Exchange Rate: 1.00
 Total Charge Amount: GBP0.00
 Narrative: Cash Deposit

Charge Details

Denomination

Bills			Coins		
Denom Code	Units	Value	Denom Code	Units	Value
100	0	0	2	0	0
50	0	0	1	0	0
20	0	0	0.5	0	0
10	10	100	0.2	0	0
5	0	0	0.1	0	0
500	0	0	0.05	0	0

Current Till Position: £10,016,553.33
 Alerts: No record to display
 Frequent Customer Operations

Submit Cancel

Figure 5: Cash Deposit Clone

The screenshot shows a web form titled 'Cash Deposit Clone'. It contains several input fields: 'Account Number', 'Transaction Amount' (GBP100.00), 'Exchange Rate' (1.00), 'Account Amount' (GBP100.00), 'Total Charge Amount' (GBP0.00), and 'Narrative' (Cash Deposit). An 'Information' dialog box is overlaid in the center, displaying a green checkmark and the text 'Transaction completed successfully' with an 'OK' button. Below the form, there are sections for 'Charge Details' and 'Denomination'.

Bills			Coins		
Denom Code	Units	Value	Denom Code	Units	Value
100	0	0	2	0	0
50	0	0	1	0	0

Figure 6: Information Message

8.2 Exact Clone with Additional Fields Using Common Code

A new screen is available with function code 9999. The Additional Fields is shown as 4th data segment as below:

The screenshot shows a web form with a sidebar on the left containing links: 'KYC Details', 'Charge Details', 'Denomination', and 'Government Tax'. The main area is titled 'Additional Fields' and contains a sub-section 'Others' with three input fields: 'Financial Year' (2020-21), 'Tax Amount' (1.000), and 'PAN' (AABCF11340F). On the right side, there are three floating windows: 'Current Till Position' showing '£19,500.00', 'Alerts' showing 'No record to display', and 'Frequent Customer Operations'. At the bottom right, there are 'Submit' and 'Cancel' buttons.

Figure 7: Additional Fields Segment

- The library reference in weblogic.xml is available for extensibility, for example, obremo-srv-ext-common-txn. A new jar obremo-srv-cmn-common-txn, which holds most of the code of transaction service and can be a dependency in the external jar.

```
<wls:library-ref>
  <wls:library-name>obremo-srv-cmn-commontxn</wls:library-name>
</wls:library-ref> Response:
```

```
{
  "data": {
    "addDtIs": {
```

```
"signatureVerifyIndicator": "Y",
"hostStatus": null,
"hostMultiTripId": null,
"txnBranchCcy": "GBP",
"txnBranchDate": "2020-03-25T18:30:00.000+0000",
"txnType": "C",
"cashInOutIndicator": "I",
"ejLoggingRequired": null,
"ejTxnAmtMapping": "TO",
"ejTxnCcyMapping": "TO",
"adviceName": null,
"orchestratorId": null,
"rsp": null,
"isReversal": "N",
"crossCcyEnabled": null,
"isTotChargesReq": null
},
"txnDtls": {
"functionCode": "9999",
"txnBranchCode": null,
"txnBranchCcy": null,
"txnBranchDate": null,
"requestStatus": "COMPLETED",
"assignmentMode": null,
"txnId": "71a08a0f-ee2a-405b-a1e3-b77ca9e59b6e",
"txnRefNumber": "0002008600007160",
"tellerSeqNumber": null,
"overrideConfirmFlag": "N",
"supervisorId": null,
```

```

        "onlineOfflineTxn": null,

        "userComments": null,

        "authoriserComments": null,

        "eventCode": null, "accountType":

        "UBS"

    },

    "dataPayload": {

        "datasegment": null,

        "fromAccountAmt": 100,

        "fromAccountCcy": "GBP",

        "toAccountCcy": "GBP", "beneficiaryName":

        null,

        "beneficiaryAddress1": null,

        "beneficiaryAddress2": null,

        "beneficiaryAddress3": null,

        "beneficiaryAddress4": null, "identificationType":

        null,

        "identificationNumber": null,

        "exchangeRate": 1,

        "recievedAccount

        Ccy": null,

        "recievedAccount

        Amt": null,

        "totalCharges": null,

        "cashAmount":

        null,

        "netAccountCcy": null, "netAccountAmt":

        null,

        "narrative": "Cash Deposit",

```

"txnControllerRefNo": null, "recordId":
"bd40562d-06b4-4f95-
95fee66fa6eb7f13", "cashAmtCcy":
null, "cashAmt":

null,
"chequeDate": null,
"chequeNumber": null,
"eventCode": null,
"ejId": null,
"emailId": null,
"fromAccountBranch": "000",
"fromAccountNumber": null,
"mobileNumber": null,
"originalExchangeRate": null,
"payee": null,
"productCode": null, "reversalDate":
null,
"stationId": null,
"toAccountBranch": "000",
"toAccountNumber": "00000008010010",
"toAccountAmt": 100,
"txnBranchCode": "000",
"functionCode": null,
"txnCustomer": null,
"tellerId": null,
"txnDate": 1585161000000,
"txnRefNumber": "0002008600007160",
"txnSeqNumber": null,
"uniqueIdentifierNumber": null,
"uniqueIdentifierType": null,
"userRefNumber": null,


```

"valueDate": null,

"versionNumber": null,

"referenceNumber": null,

"createdBy": null, "createdTs":

null,

"updatedBy": null, "updatedTs":

null,

"demDtls": null,

"fxInDemDtls": null,

"fxOutDemDtls": null,

"prcDtls": null,

"addDtls": null,

"txnDtls": null,

"overrideDtls": null,

"batchTableDetails": null

},

"extDetails": null,

"warDtls": [], "authoriserDtls":

[]

},

"errors": null,

"warnings": null,

"informations": null,

"authorizations": null,

"paging": ""

}

```

Columns	Data	Model	Constraints	Grants	Statistics	Triggers	Flashback	Dependencies	Details	Partitions	Indexes	SQL
UI_KEY	DATA_REF_KEY	FIELD_META_DATA_VER	FIELD_DATA									
1 9811 fsgbu-ob-cmn-ds-additional-fields@9999	bd40562d-06b4-4f95-95fe-e66fa6eb7f13	1	{ "OTH_Year": "2020", "OTH_Amount": 100, "OTH_Number": "DAHFM214AH" }									
2 5ed3 fsgbu-ob-cmn-ds-additional-fields@9999	bd40562d-06b4-4f95-95fe-e66fa6eb7f13	1	{ "OTH_Year": "2020", "OTH_Amount": 100, "OTH_Number": "DAHFM214AH" }									
3 7c96 fsgbu-ob-cmn-ds-additional-fields@9999	bd40562d-06b4-4f95-95fe-e66fa6eb7f13	1	{ "OTH_Year": "2020", "OTH_Amount": 100, "OTH_Number": "DAHFM214AH" }									
4 2826 fsgbu-ob-cmn-ds-additional-fields@9999	bd40562d-06b4-4f95-95fe-e66fa6eb7f13	1	{ "OTH_Year": "2020", "OTH_Amount": 100, "OTH_Number": "DAHFM214AH" }									
5 632c fsgbu-ob-cmn-ds-additional-fields@9999	bd40562d-06b4-4f95-95fe-e66fa6eb7f13	1	{ "OTH_Year": "2020", "OTH_Amount": 100, "OTH_Number": "DAHFM214AH" }									

Figure 8: Common Core Additional Attributes

- In the debug, you can find that the common code is used, stemplmpl onCashSubmitTillAcc will be called.

```
lf-tuning)'] o.f.p.c.p.p.PlatoProxyEntityManager : PlatoProxyEntityManager :: Application :: Current A
lf-tuning)'] o.f.p.c.p.p.PlatoProxyEntityManager : PlatoProxyEntityManager :: Application :: Current T
lf-tuning)'] o.f.p.c.p.p.PlatoProxyEntityManager : The application [ App id = SRVCMNTXN / Tenant Id =
lf-tuning)'] o.f.p.c.p.provider.PlatoRegistry : appId [ SRVCMNTXN ]
lf-tuning)'] o.f.p.c.p.provider.PlatoRegistry : tenantId [ SRVCMNTXN ]
lf-tuning)'] o.f.p.c.p.provider.PlatoRegistry : emType [ APPLICATION ]
lf-tuning)'] o.f.p.c.p.provider.PlatoRegistry : Entity Manager Factory is available in Cache for th
lf-tuning)'] StepImpl : Here for function code 9999 and beanname is FC9999
lf-tuning)'] StepImpl : onCashSubmitTillAcc operation
lf-tuning)'] o.f.o.s.s.t.domain.CashService : inside onCashSubmitTillAcc
lf-tuning)'] o.f.o.s.s.t.s.TransactionServiceImpl : START fetching the data
lf-tuning)'] o.f.o.s.s.t.s.TransactionServiceImpl : START fetching the data
lf-tuning)'] o.f.o.s.s.t.domain.CashService : after calll to move data from work to main charges
lf-tuning)'] o.f.o.s.s.t.domain.CashService : Going to call EJ Creation
lf-tuning)'] o.f.o.s.s.srv.transaction.util.Common : GenerateEJIdStep ends
lf-tuning)'] o.f.o.s.s.t.domain.CashService : Going for enrichment
lf-tuning)'] o.f.o.s.s.t.domain.CashService : Going for validate Roles check
lf-tuning)'] o.f.o.s.s.srv.transaction.util.Common : inside validateRole
lf-tuning)'] o.f.o.s.s.srv.transaction.client.SMSImpl : Going to call userLoginId
lf-tuning)'] o.f.o.s.s.t.domain.CashService : Goinf for balance check
```

Figure 9: Common Code

8.3 Exact Clone with Additional Fields Using Extensible Code

A screen is created with function code 9999 and Additional Fields as 4th data segment.

Figure 10: Additional Fields Segment

- A library reference is added weblogic.xml (obremo-srv-ext-common-txn) for extensibility. A new jar obremosrvcmn-common-txn, which holds the most of the code of transaction service and can be a dependency in the external jar.

```
<wls:library-ref>
```

```
<wls:library-name>obremo-srv-cmn-common-txn</wls:library-name>
```

```
</wls:library-ref>
```

8.4 Jar Deployment in Weblogic:




<input type="checkbox"/>	 obremo-srv-cmn-transaction-services-5.2.0_snapshot	Active	✔ OK	Web Application	SERVICING	Global	1
<input type="checkbox"/>	 obremo-srv-cus-customer-services-5.2.0_snapshot	Active	✔ OK	Web Application	SERVICING	Global	1
<input type="checkbox"/>	 obremo-srv-ext-common-txn	Active		Library	SERVICING	Global	1

Figure 11: Jar Deployment

Response:

```
{
  "data": {
    "addDtls": {
      "signatureVerifyIndicator": "Y",
      "hostStatus": null,
      "hostMultiTripld": null,
      "txnBranchCcy": "GBP",
      "txnBranchDate": "2020-03-25T18:30:00.000+0000",
      "txnType": "C",
      "cashInOutIndicator": "I",
      "ejLoggingRequired": null,
      "ejTxnAmtMapping": "TO",
      "ejTxnCcyMapping": "TO",
      "adviceName": null,
      "orchestratorId": null,
      "rsp": null,
      "isReversal": "N",
      "crossCcyEnabled": null,
      "isTotChargesReq": null
    },
    "txnDtls": {
      "functionCode": "9999",
      "txnBranchCode": null,
      "txnBranchCcy": null,
      "txnBranchDate": null,
      "requestStatus": "COMPLETED",
    }
  }
}
```

```

    "assignmentMode": null,
    "txnId": "71a08a0f-ee2a-405b-a1e3-b77ca9e59b6e",
    "txnRefNumber": "0002008600007160",
    "tellerSeqNumber": null,
    "overrideConfirmFlag": "N",
    "supervisorId": null,
    "onlineOfflineTxn": null,
    "userComments": null,
    "authoriserComments": null,
    "eventCode": null, "accountType":
    "UBS"
  },
  "dataPayload": {
    "datasegment": null,
    "fromAccountAmt": 100,
    "fromAccountCcy": "GBP",
    "toAccountCcy": "GBP",
    "beneficiaryName": null,
    "beneficiaryAddress1": null,
    "beneficiaryAddress2": null,
    "beneficiaryAddress3": null,
    "beneficiaryAddress4": null, "identificationType":
    null,
    "identificationNumber": null,
    "exchangeRate": 1,
    "recievedAccountCcy": null, "recievedAccountAmt":
    null,
    "totalCharges": null,
    "cashAmount":
    ount":

```

null,
"netAccountCcy": null, "netAccountAmt":
null,
"narrative": "Cash Deposit",
"txnControllerRefNo": null, "recordId":
"bd40562d-06b4-4f95-
95fee66fa6eb7f13", "cashAmtCcy":
null, "cashAmt":
null,
"chequeDate": null,
"chequeNumber": null,
"eventCode": null,
"ejId": null,
"emailId": null,
"fromAccountBranch": "000",
"fromAccountNumber": null,
"mobileNumber": null,
"orginalExchangeRate": null,
"payee": null,
"productCode": null, "reversalDate":
null,
"stationId": null,
"toAccountBranch": "000",
"toAccountNumber": "00000008010010",
"toAccountAmt": 100,
"txnBranchCode": "000",
"functionCode": null,
"txnCustomer": null,
"tellerId": null,
"txnDate": 1585161000000,
"txnRefNumber": "0002008600007160",

```
    "txnSeqNumber": null, "uniqueIdentifierNumber":  
    null,  
    "uniqueIdentifierType": null,  
    "userRefNumber": null,  
    "valueDate": null,  
    "versionNumber": null,  
    "referenceNumber": null,  
    "createdBy": null, "createdTs":  
    null,  
    "updatedBy": null,  
    "updatedTs": null,  
    "demDtls": null,  
    "fxInDemDtls": null,  
    "fxOutDemDtls": null,  
    "prcDtls": null,  
    "addDtls": null,  
    "txnDtls": null,  
    "overrideDtls": null,  
    "batchTableDetails": null  
  },  
  "extDetails": null,  
  "warDtls": [], "authoriserDtls":  
  []  
},  
"errors": null,  
"warnings": null,  
"informations": null,  
"authorizations": null,
```

```

    "paging": ""
}

```

UI_KEY	DATA_REF_KEY	FIELD_META_DATA_VER	FIELD_DATA
1 9811 fsgbu-ob-cmn-ds-additional-fields@9999	bd40562d-06b4-4f95-95fe-e66fa6eb7f13 1		("OTH_Year": "2020", "OTH_Amount": 100, "OTH_Number": "DAHFM214AH")
2 5ed3 fsgbu-ob-cmn-ds-additional-fields@9999	bd40562d-06b4-4f95-95fe-e66fa6eb7f13 1		("OTH_Year": "2020", "OTH_Amount": 100, "OTH_Number": "DAHFM214AH")
3 7c96 fsgbu-ob-cmn-ds-additional-fields@9999	bd40562d-06b4-4f95-95fe-e66fa6eb7f13 1		("OTH_Year": "2020", "OTH_Amount": 100, "OTH_Number": "DAHFM214AH")
4 2826 fsgbu-ob-cmn-ds-additional-fields@9999	bd40562d-06b4-4f95-95fe-e66fa6eb7f13 1		("OTH_Year": "2020", "OTH_Amount": 100, "OTH_Number": "DAHFM214AH")
5 632c fsgbu-ob-cmn-ds-additional-fields@9999	bd40562d-06b4-4f95-95fe-e66fa6eb7f13 1		("OTH_Year": "2020", "OTH_Amount": 100, "OTH_Number": "DAHFM214AH")

Figure 12: Common Core Additional Attributes

- In the debug, the extensible code is used, which is present in the extension jar (obremono-srv-ext-commontxn.jar). Instead of `stemplImpl.onCashSubmitTillAcc`, `FC9999.onCashSubmitTillAcc` will be called, where you can add code that is required for the new dataSegment added or to achieve different functionality of charging, accounting, till updates, etc.

```

lf-tuning) ' o.f.p.c.p.p.PlatoProxyEntityManager : PlatoProxyEntityManager :: Application :: Current App
lf-tuning) ' o.f.p.c.p.p.PlatoProxyEntityManager : PlatoProxyEntityManager :: Application :: Current Te
lf-tuning) ' o.f.p.c.p.p.PlatoProxyEntityManager : The application [ App id = SRVCMNTXN / Tenant Id = r
lf-tuning) ' o.f.p.c.p.provider.PlatoRegistry : appId [ SRVCMNTXN ]
lf-tuning) ' o.f.p.c.p.provider.PlatoRegistry : tenantId [ SRVCMNTXN ]
lf-tuning) ' o.f.p.c.p.provider.PlatoRegistry : emType [ APPLICATION ]
lf-tuning) ' o.f.p.c.p.provider.PlatoRegistry : Entity Manager Factory is available in Cache for the
lf-tuning) ' FC9999 : Here for function code 9999 and beanname is FC9999
lf-tuning) ' FC9999 : onCashSubmitTillAcc operation
lf-tuning) ' o.f.o.s.s.t.domain.CashService : inside onCashSubmitTillAcc
lf-tuning) ' o.f.o.s.s.t.s.TransactionServiceImpl : START fetching the data
lf-tuning) ' o.f.o.s.s.t.s.TransactionServiceImpl : START fetching the data
lf-tuning) ' o.f.o.s.s.t.domain.CashService : after calll to move data from work to main charges e
lf-tuning) ' o.f.o.s.s.t.domain.CashService : Going to call EJ Creation
lf-tuning) ' o.f.o.s.s.srv.transaction.util.Common : GenerateEJIdStep ends
lf-tuning) ' o.f.o.s.s.t.domain.CashService : Going for enrichment
lf-tuning) ' o.f.o.s.s.t.domain.CashService : Going for validate Roles check
lf-tuning) ' o.f.o.s.s.srv.transaction.util.Common : inside validateRole
lf-tuning) ' o.f.o.s.s.srv.transaction.client.SMSImpl : Going to call userLoginId
lf-tuning) ' o.f.o.s.s.t.domain.CashService : Goinf for balance check

```

Figure 13: Debug Codes

9 Extensibility Use Cases for OBX

9.1 New Transaction screen – 1499 (Clone of 1401)

For this use case, make sure that the data is present in the below tables similar to 1401. The below mentioned tables need to be checked in SMS schema:

- SMS_TM_MENU
- SMS_TM_MENU_Description
- SMS_TM_SERVICE_ACTIVITY
- SMS_TM_FUNCTIONAL_ACTIVITY
- SMS_TM_FUNC_ACTIVITY_DETAIL
- SMS_TM_ROLE_ACTIVITY
- SMS_TM_UI_ACTIVITY

The below mentioned tables need to be checked in in Common Core schema:

- CMC_TM_SCREEN_CLASS
- CMC_TM_SCREEN_DS_MAPPING

The below mentioned tables need to be checked in branch Common schema:

- SRV_TM_BC_FUNCTION_INDICATOR
- SRV_TM_BC_FUNCTION_CODE
- SRV_TM_BC_FUNCTION_PREF
- SRV_TM_BC_FUNCTION_PREF_DTLS
- SRV_TM_BC_BRANCH_ACCOUNTING
- SRV_TM_MENU_CONFIG

Figure 14: Cash Deposit Clone

Figure 15: Information Message

9.2 New Data Segment in Existing 1401 Screen

For this use case, it is needed to implement UI Component and Service side to persist data. The steps to create UI Component are as follows:

- Start OBX and create XDL by running command `obx xdl-gen`.
- Once XDL is created, go to Cmder tab, and press Y for XDL generation.

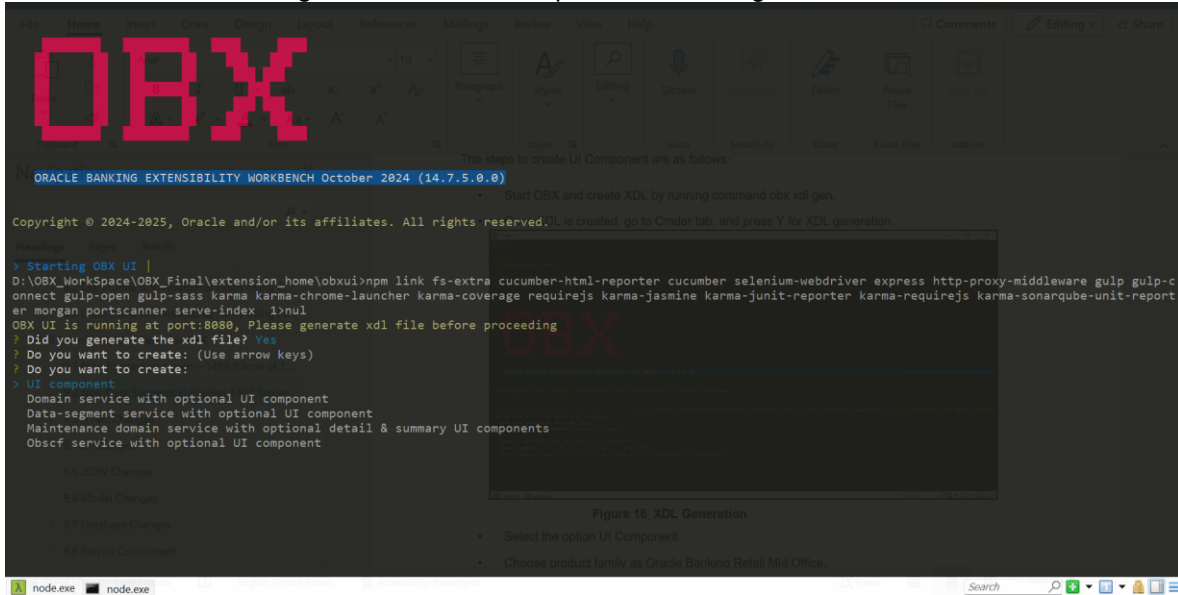


Figure 16 XDL Generation

- Select the option UI Component.
- Choose product family as Oracle Banking Retail Mid Office.
- Specify the name of virtual page/data-segment/stand-alone component to be created.
- Specify absolute path of the XDL generated. (XDL is generated inside `extension_home` folder).
- NOTE: A new UI Component will be created in `extension_home` folder with prefix `obxvp/obx-ds`. In the Cmder tab, OBX will prompt to modify `Metadata.js` file of the newly created component. In addition, the component-server will start running at port 8002.



Figure 17: XDL Path

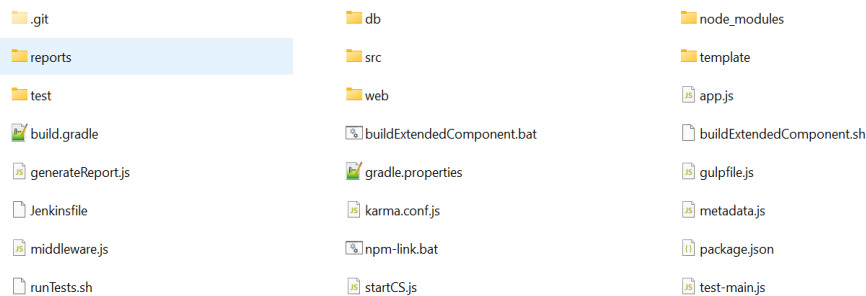


Figure 18: Extension Home Folder

- The generated UI component contains boiler plate code to do the common operations of Save, Get, Get All etc. Changes needed in the newly created component from OBX tool from UI side.

9.3 HTML Changes

- The HTML fields look like Figure 19: HTML Changes for all the screens. According to the screen design, one can change the HTML values like payload() and mobileNumber. If mobileNumber field is entered by the user, value of mobileNumber will directly update the JS payload that will be going as a part of save call.

```

<div id="dialog-content" role="main" class="oj-sm-padding-2x-top">
</div>
</div>
</oj-dialog>
<div class="oj-panel oj-sm-margin-2x demo-panel-customizations">
  <div class="oj-flex oj-form-layout oj-sm-only-flex-direction-column oj-lg-flex-items-1">
    <div class="oj-flex-item">
      <div class="oj-form oj-sm-odd-cols-12 oj-md-labels oj-form-cols-labels-inline oj-form-cols-max">
        <div class="oj-flex">
          <div class="oj-flex-item">
            <oj-label for="depositorname" show-required="true">
              <span data-bind="text : labels.obxvpadditionaldetails.depositorName"></span>
            </oj-label>
            <oj-input-text id="depositorname" required="true"
              value="{{payload().depositorName}}"></oj-input-text>
          </div>
        </div>
        <div class="oj-flex">
          <div class="oj-flex-item">
            <oj-label for="mobilenumber" show-required="true">
              <span data-bind="text : labels.obxvpadditionaldetails.mobileNumber"></span>
            </oj-label>
            <oj-input-text id="mobilenumber" required="true"
              value="{{payload().mobileNumber}}"></oj-input-text>
          </div>
        </div>
      </div>
    </div>
  </div>
</div>

```

Figure 19: HTML Changes

- The `oj-validation-group` is required for configuring the HTML as part of validation.

```

<div class="oj-form oj-form-no-dividers oj-lg-form-across">
  <oj-validation-group data-bind="attr : { 'id' : 'tracker' + unique()}" valid="{{groupValid}}">
    <div id="fsgbu-ob-remo-srv-ds-cash-deposit-input-panel" class="oj-flex wizard-input-panel ">

```

Figure 20: Validation

9.4 JS Changes

Perform the following steps to implement JS changes:

- Add all the dependencies in define block.
- The JS **self.payload** is an observable, which will hold all the info inputted from the HTML. All keys in **self.payload** is directly linked with HTML.

```

define(['ojs/ojcore',
'jquery',
'knockout',
'ojL10n!./resources/nls/bundle',
'./model/additionaldetails-model',
'ojs/ojarraydataprovider',
'ojs/ojbutton',
'ojs/ojknockout',
'ojs/ojinputtext',
'ojs/ojcheckboxset',
'ojs/ojtable',
'cmn-cca/fsgbu-ob-cmn-fd-lov/loader',
'cmn-cca/fsgbu-ob-cmn-fd-date/loader',
'cmn-cca/fsgbu-ob-cmn-fd-amount/loader',
'ojs/ojswitch',
'ojs/ojpagingcontrol',
'ojs/ojdialog', 'components/fsgbu-ob-remo-srv-cmn-ct-datasegment/loader'],
function (oj, $, ko, labels, model, ArrayDataProvider) {
    /**

```

Figure 21: JS Changes

```

self.payload=ko.observable({
    "datasegment": ko.observable(self.datasegment()),
    "depositorName": ko.observable(),
    "mobileNumber": ko.observable(),
});

```

Figure 22: JS Self Payload

- Save method implementation will look like Figure 23: Save Method. In the next line, it is making a promise and calling the save function of cmn-ct-datasegment providing the payload and endpoint as parameters. If save is success, it will resolve and for failures it will come to reject.

```

self.save = function (wiz, data) {
    if (self.validate()) {
        self.payload().isMainDs = false;
        return new Promise(function (resolve, reject) {
            self.cmnCtDatasegment().save(self.payload(), "OBREMO.SAVE_ADDITIONAL_DETAILS").then(function (response) {
                if (!self.isEmptyNullOrUndefined(response.errors)) {
                    reject(response);
                }
                else {
                    resolve(response)
                }
            });
        });
    }
    else {
        // show messages on all the components
        // that have messages hidden.
        tracker.showMessages();
        tracker.focusOn("@firstInvalidShown");
    }
};

```

Figure 23: Save Method

- The function null check is as shown below:

```
self.isEmptyNullOrUndefined = function (value) {
  if (value === "" || value === undefined || value === null) {
    return true;
  } else {
    return false;
  }
};
```

Figure 24: Function Null Check

- The validate function is shown in the Figure 25: Validate Function, which will check all mandatory fields during save.

```
self.validate = function () {
  tracker = document.getElementById("tracker"+self.unique());
  if (tracker.valid === "valid") {
    return true;
  }
  else{
    return false;
  }
}
```

Figure 25: Validate Function

9.5 JSON Changes

The data and datatransferPayload properties need to be exposed from JSON. The data property is used to take the information of transaction specific and the datatransferPayload property is used to share data between data segments.

```
{
  "name": "obx-vp-additionaldetails",
  "version": "1.0.0",
  "jetVersion": ">=5.2.0",
  "properties": {
    "name": {
      "description": "The name to display",
      "type": "string"
    },
    "data": {
      "description": "The name to display",
      "type": "object",
      "writeback": true
    },
    "dataTransferPayload": {
      "description": "The name to display",
      "type": "object",
      "writeback": true
    }
  },
  "methods": {
    "save": {
      "description": "Save and Close"
    }
  },
  "events": {}
}
```

Figure 26: JSON Changes

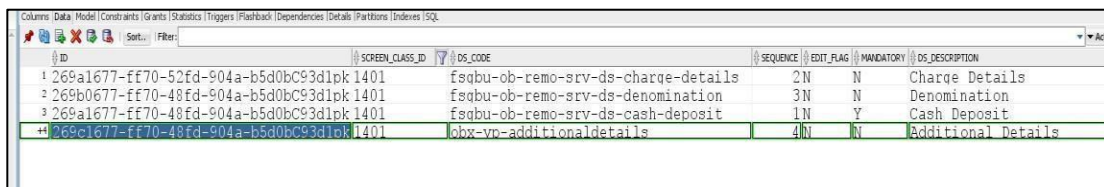
9.6 Model Changes

There will be no methods in the model. All the REST calls need to go through cmn-ctdatasegment similar to Save. Perform the following steps to make model changes:

- Run the DB Scripts present in this component.
- NOTE: The OBX generates SQL script with default HEADER_APPID as PXDSSRV001 for all components. This script can be changed and used.
- Create extended war for the component and deploy.

9.7 Database Changes

- Add the newly created data segment name in the PRODUCT_EXTENDED_LEDGER table (this will be done when DB script from UI component is run).
- Make a fourth Data Segment entry for function code 1401 in CMC_TM_SCREEN_DS_MAPPING table of CMNCORE. The DS_CODE should be the name of the UI Component created. The entry is as shown in the Figure 27: Data Segment Entry.



The screenshot shows a database table with columns: ID, SCREEN_CLASS_ID, DS_CODE, SEQUENCE, EDIT_FLAG, MANDATORY, and DS_DESCRIPTION. The table contains four rows of data. The fourth row is highlighted in green.

ID	SCREEN_CLASS_ID	DS_CODE	SEQUENCE	EDIT_FLAG	MANDATORY	DS_DESCRIPTION
1 269a1677-ff70-52fd-904a-b5d0bc93d1pk	1401	fsqbu-ob-remo-srv-ds-charge-details	2	N	N	Charge Details
2 269b0677-ff70-48fd-904a-b5d0bc93d1pk	1401	fsqbu-ob-remo-srv-ds-denomination	3	N	N	Denomination
3 269a1677-ff70-48fd-904a-b5d0bc93d1pk	1401	fsqbu-ob-remo-srv-ds-cash-deposit	1	N	Y	Cash Deposit
4 269c1677-ff70-48fd-904a-b5d0bc93d1pk	1401	obx-vo-additionaldetails	4	N	N	Additional Details

Figure 27: Data Segment Entry

- If the service is created separately than UI Component, change the endpoint URL in SQL script for table PRODUCT_SERVICES_LEDGER accordingly.

9.8 Service Component

- Start OBX and use the XDL file that is already generated.
- Select the domain service with optional UI component.



Figure 28: Domain Service

- Select product family as **Oracle Banking Retail Mid Office**



Figure 29: Product Family

- Specify the service name as additional Details and all the remaining details as mentioned in the Figure 30: Service Name.

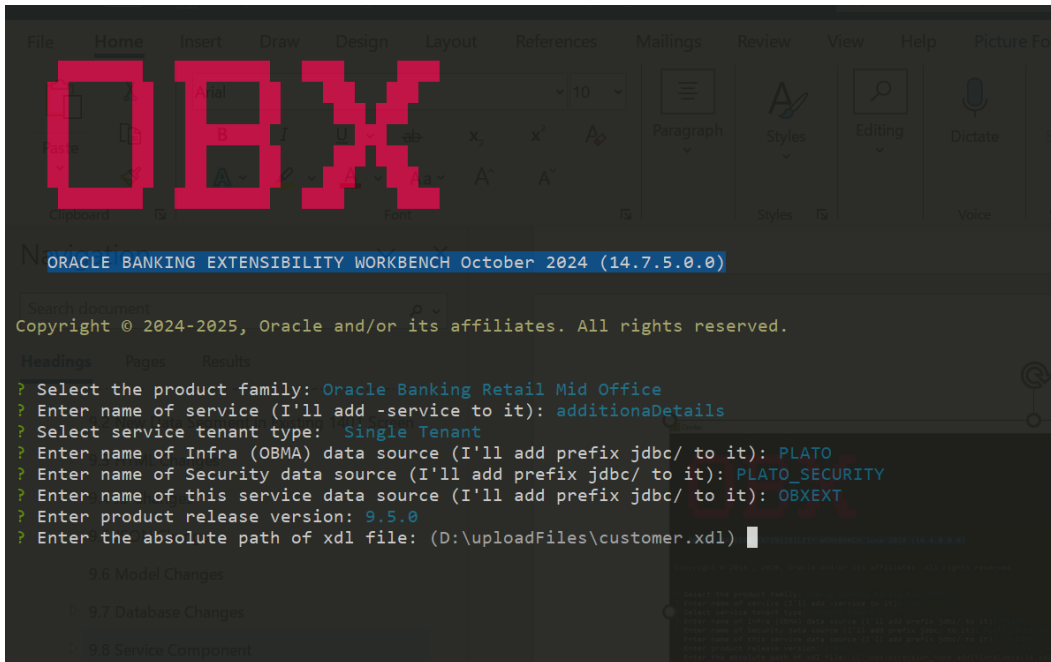


Figure 30: Service Name

- A new service is generated in **extension_home** folder with prefix **obremo-additionadetails-service**

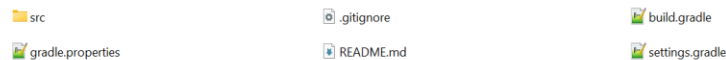
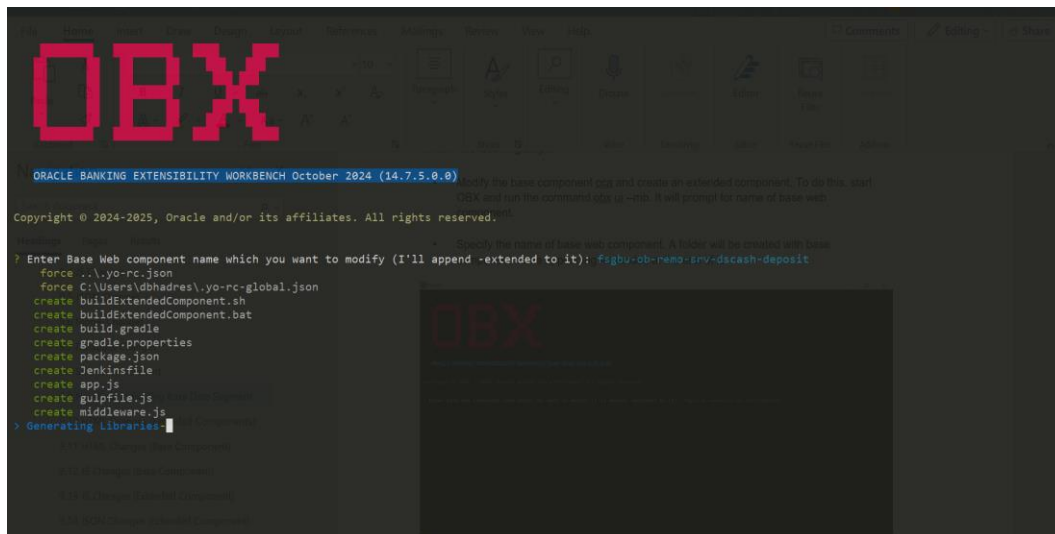


Figure 31: Extension Home Folder

- Run the DB scripts present in this service.
- NOTE: It will create a new table to persist data of new data segment. For example, a table is created as **ADDITIONALDETAILS**. This table can be created in existing schema or in a new schema.
- If you need to create a new schema, mention that in table. **PRODUCT_SERVICES_CTX_LEDGER** while running UI Component Script.
- Restart plato servers once this change is completed.
- If required, make appropriate changes in the service, build it, and deploy.
- NOTE: After deploying extended war and additional details service along with proper DB entry, you can see a new data segment in the appshell screen.
- Fill the necessary details and click **Submit**, the data for new DS will be saved in new table.



```

Initializing empty Git repository.....
Initialized empty Git repository in D:/OBX_WorkSpace/OBX_Final/extension_home/fsgbu-ob-remo-srv-dscash-deposit-extended/.git/
create template\js\appController.js
create src\components\fsgbu-ob-remo-srv-dscash-deposit-extended\fsgbu-ob-remo-srv-dscash-deposit-extended.json
create src\components\fsgbu-ob-remo-srv-dscash-deposit-extended\fsgbu-ob-remo-srv-dscash-deposit-extended.scss
create src\components\fsgbu-ob-remo-srv-dscash-deposit-extended\fsgbu-ob-remo-srv-dscash-deposit-extended.html
create src\components\fsgbu-ob-remo-srv-dscash-deposit-extended\resources\nls\bundle.js
create src\components\fsgbu-ob-remo-srv-dscash-deposit-extended\fsgbu-ob-remo-srv-dscash-deposit-extended.js
create db\ui-config\config_extended.sql
create generateReport.js
create karma.conf.js
create runTests.sh
create test-main.js
create template\index.html
create template\js\knockout-mapping.js
create template\js\main.js
create test\mocks\common.js
create test\mocks\commonFunction.js
create test\cucumber\features\support\hooks.js
create test\cucumber\features\support\timeouts.js
create test\cucumber\features\support\world.js
create src\components\fsgbu-ob-remo-srv-dscash-deposit-extended\loader.js
create test\components\fsgbu-ob-remo-srv-dscash-deposit-extended\fsgbu-ob-remo-srv-dscash-deposit-extendedSpec.js
create test\cucumber\features\fsgbu-ob-remo-srv-dscash-deposit-extended.feature
create test\cucumber\features\steps\fsgbu-ob-remo-srv-dscash-deposit-extended.js
extended-component created

```

Figure 34: Base Web Component

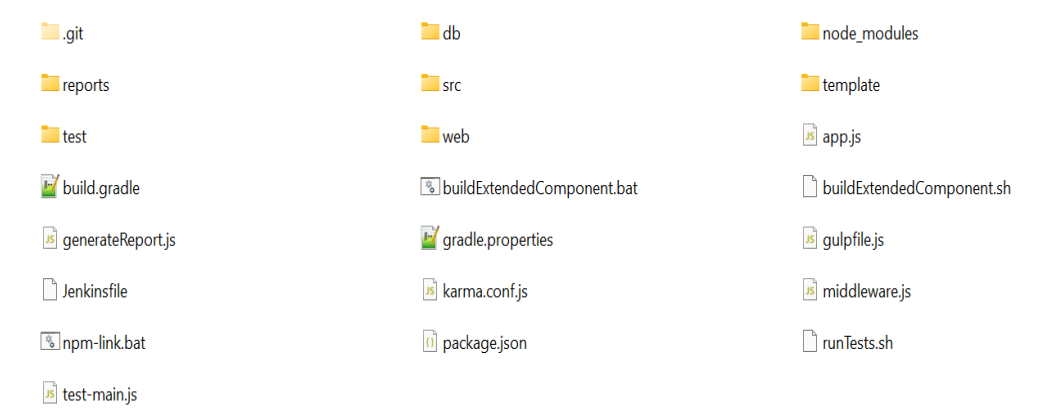


Figure 35: Extended Folder

- NOTE: Changes needed in the extended component from UI side.

9.10 HTML Changes (Extended Components)

The extended component contains the boiler plate codes, in which you need to make the changes as shown in the below figure HTML Changes (Extended Component). After you make the necessary changes, the additional fields will be added after the existing fields in the base component.

```
<oj-dialog id='extensiondialog'>
  <!-- <div id='aadharfield' class="oj-flex-item">
    <div class="oj-xl-6 oj-lg-6 oj-md-12 oj-sm-12 oj-flex-item oj-flex wizard-input-field"> -->
    <div id='aadharfield' class="oj-flex-item">
      <oj-label id="ui-id-12-labelled-by">
        <span>
          <!--ko text: labels.aadharNoLbl-->
          <!-- /ko -->
        </span>
      </oj-label>
      <oj-input-text id='aadharNo' value="{{data.aadharNo}}" label-hint="Aadhar Number">
        <input data-internal="" type="text" placeholder="">
      </oj-input-text>
    <!-- </div>
  </div> -->
  <div id='panfield' class="oj-flex-item">
    <!-- <div class="oj-xl-6 oj-lg-6 oj-md-12 oj-sm-12 oj-flex-item oj-flex wizard-input-field">
    <div class="oj-flex-item"> -->
    <oj-label id="address3">
      <span>
        <!--ko text: labels.panNoLbl-->
        <!-- /ko -->
      </span>
    </oj-label>
    <oj-input-text id='panNo' value="{{data.panNo}}" label-hint="Pan Number">
      <input data-internal="" type="text" placeholder="">
    </oj-input-text>
  <!-- </div>
</div> -->
</div>
</oj-dialog>
```

Figure 36: HTML Changes (Extended Component)

The following changes are required only if you need to add the additional field at the end of the base component and in a separate extension panel. You can choose to add the additional fields in the existing base component or in the extension panel as per the requirement.

```
<!-- <div id="extensionpanel" class="oj-panel oj-panel-shadow-sm oj-sm-margin-2x demo-mypanel">
  <h3 class="oj-header-border">Extension</h3>
  <oj-form-layout id="extension" max-columns="{{columns}}" direction="row">
    <oj-input-text value="{{data.mobile}}" label-hint="Mobile Number"></oj-input-text>
    <oj-input-text value="{{data.address3}}" label-hint="Address3"></oj-input-text>
  </oj-form-layout>
</div> -->
```

Figure 37: Extension Panel

9.11 HTML Changes (Base Component)

Perform the HTML changes in the base component as shown in Figure 38: HTML Changes (Base Component).

```
<!-- ko if: ifExtension -->
<fsgbu-ob-remo-srv-ds-cash-deposit-extended data="{{payload}}" base="{{base}}">
</fsgbu-ob-remo-srv-ds-cash-deposit-extended>
<!-- /ko -->
```

Figure 38: HTML Changes (Base Component)

9.12 JS Changes (Base Component)

Perform the JS changes in the base component as shown in Figure 39: JS Changes (Base Component).

```
self.loadExtendedCCA = ko.observable('fsgbu-ob-base-component-extended');
self.ifExtension = ko.observable(false);

self.loadExtendedComponent = function () {
  // eslint-disable-next-line no-undef
  if (requirejs.s.contexts._.config.paths['components/' + self.loadExtendedCCA()]) {
    var componentName = ['components/' + self.loadExtendedCCA() + '/loader'];
    require(componentName, function () {
      self.ifExtension(true);
    });
  }
};
```

Figure 39: JS Changes (Base Component)

The part of code shown below is present in JS or view model file. From the self.connected method, you need to call self.loadExtendedComponent method.

```
self.connected = function (context) {
  self.loadExtendedComponent();
};
```

Figure 40: Self Connected Method

9.13 JS Changes (Extended Component)

In the bindings applied, it will take the ID of the fields and add the additional fields after the field base component. Both additional fields will be added after the field of base component for which the ID is **lastTab**.

```

self.bindingsApplied = function (context) {

    self.entityNameTemplate = document.getElementById('aadharfield');
    self.newentityNameTemplate = self.entityNameTemplate.cloneNode(true);
    document.querySelector("#lastTab").insertAdjacentHTML('afterEnd', self.newentityNameTemplate.outerHTML);

    self.entityNameTemplate1 = document.getElementById('panfield');
    self.newentityNameTemplate1 = self.entityNameTemplate1.cloneNode(true);
    document.querySelector("#lastTab").insertAdjacentHTML('afterEnd', self.newentityNameTemplate1.outerHTML);

    applyBindings(context);
};
}
function applyBindings(context) {
    ko.applyBindings(mainContentViewModel(context), $("#aadharfield")[0]);
    ko.applyBindings(mainContentViewModel(context), $("#panfield")[0]);
}

```

Figure 41: JS Changes (Extended Component)

9.14 JSON Changes (Extended Component)

Perform the HTML changes as shown in Figure 42: JSON Changes (Extended Component) to add data and base property for extended component.

```

{
  "name": "fsgbu-ob-remo-srv-ds-cheque-withdrawal-extended",
  "version": "1.0.0",
  "jetVersion": ">=5.2.0",
  "properties": {
    "name": {
      "description": "The name to display",
      "type": "string"
    },
    "data": {
      "description": "The name to display",
      "type": "object",
      "writeback": true
    },
    "base": {
      "description": "The name to display",
      "type": "object",
      "writeback": true
    }
  },
  "methods": {},
  "events": {}
}

```

Figure 42: Json Changes (Extended Component)

9.15 JSON Changes (Base Component)

In base component JSON file, the properties isExtensible and authMode are present. You need to make changes in the existing appshell UI component so that it reads the extended component. In addition, it will contain DB scripts which need to be run.

```

"name": "fsgbu-ob-remo-srv-ds-cash-deposit",
"version": "1.0.0",
"isVirtualPage": "true",
"isExtensible": true,
"properties": {
  "name": {
    "description": "The name to display",
    "type": "object"
  },
  "totalDS": {
    "description": "The totalDS to display"
  },
  "data": {
    "description": "The name to display",
    "type": "object",
    "writeback": true
  },
  "authMode": {
    "description": "Authorization mode",
    "type": "boolean"
  },
}

```

Figure 43: JSON Changes (Base Component)

9.16 DB Changes

Add the newly created data segment name in the PRODUCT_EXTENDED_LEDGER table. Perform the following steps to make the service level change:

- Add a new field named `additionalFields` with data type `String` in work and main table entity classes of the respective service. The corresponding setters and getters should also be added in these classes.
- `@Column(name = "ADDITIONAL_FIELDS") private String additionalFields;`
- Add a column with the name **ADDITIONAL_FIELDS** in the main and work tables of the DB with CLOB data type.
- For persistence of data in main table, add **additionalFields** with data type `String` in model class.
- Deploy the changed service, extended war component, and changed appshell.
- **NOTE:** After deployment, the two additional fields named **Pan No** and **Aadhaar No** will be added in existing data segment.

- Specify the necessary details and click **Submit**. The additional fields will be saved in respective work and main table in an additional column **ADDITIONAL_FIELDS**.

Figure 44: Data Segment with Additional Fields

- In the request payload from UI to backend, the values appear as follows:

```

▼ Request Payload view source
▼ {dataset: "fsgbu-ob-remo-srv-ds-cheque-withdrawal", chequeDate: "2020-03-26",...}
  ▶ addDtls: {txnType: "C", cashInOutIndicator: "O", ejTxnAmtMapping: "FROM", ejTxnCcyMapping: "FROM",...}
  additionalFields: "{"aadharNo":"1234567890","panNo":"123456abc"}"
  chequeDate: "2020-03-26"
  chequeNumber: "123456"

```

Figure 45: Request Payload

- The data will get saved in newly added column Additional Fields in the respective table.

TXN_DATE	FROM_ACC_NO	FROM_CUR	EX_CUR	TO_ACC_BRN	TO_ACC_CCY	TO_ACC_AMT	NARRATIVE	CREATED_TS	STATUS	ADDITIONAL_FIELDS
26-MAR-20	000000001060032	000	GBP	10	1.000	GBP	10.Cheque Withdrawal.	(null)	(null)	{"aadha...

Figure 46: SRV_TB_CH_CASH_TXN Table

9.17 Add New Columns in Base Component Table

- Create an extended component for the base cca by making these changes in the base accordingly.
- Changes in base

In HTML

```
<!-- ko if: ifExtension -->
<componentName-extended data="{{base}}">
</componentName-extended>
<!-- /ko -->
```

In JS

```
        self.base
= this;

        self.ifExtension = ko.observable(false);
        self.connected = function () {            if
(requirejs.s.contexts._.config.paths['components/componentName-
extended']) {
            require(['components/componentName-extended/loader'], function () {
self.ifExtension(true);

            });
        }
    }
}
```

- Changes in extended

```
        self.bindingsApplied = function (context) {
context.props.then(function (properties) {
console.log(properties.data.columnArray);

        properties.data.columnArray.splice(columnIndex, 0, {
headerText: "Manager Id",                field: "ManagerId"

        });
tableId.refresh(properties.data.columnArray);

    });
}
```

- Changes needed at service level

For data inside table, custom projection service had to be written, custom events needs to be raised while custom fields persistence. For base fields, a call can be made from projection service to base service to fetch data and persisting the same over projection schema.

9.18 Steps for adding extra column in task grid

- Clone the respective Free/My/Hold Task components
- Then the additional column can be added using the following example code snippet

```
○ self.additionalColumns = [{
    dataIndex:
'customerName',
    dataType:
'string',
    displayType: 'text',
    width: '60px',
    sortable: true,
    resizable: true,
    accessTo: ['AVAILABLE', 'HOLD', 'ACQUIRED']
}];
```

The above code needs to be added in js file of the cloned components.

- While calling 'fsgbu-ob-cmn-fd-work-list' from the html of the cloned components please make a call like this (which also sends additional columns as a property)
- Example:

```
<fsgbu-ob-cmn-fd-work-list id='completedTaskGridCCA' dashboard-id='STANDARD'
dashboardqueue-name='ACQUIRED'
process-code={{processCode}} dashboard-queue-type='L' worklist-columns='{{columnArray}}'
additional-columns='{{additionalColumns}}' page-size=20 dependent-
vm="{{dialogParameters}}"></fsgbu-ob-cmn-fd-work-list>
```

- Making these changes would display the extra column in the task screens.

9.19 Steps to use Additional Buttons provision in Task Screen

- In the custom component (example - fsgbu-ob-slp0-vp-wl-locked-task-extended) from where you will be calling 'fsgbu-ob-cmn-fd-work-list', make the following changes
 - a. In the js file you can declare an array of the buttons you want to include like this-

```
self.extraButtons = [{
    label: 'Extraa',
    icons: {
start: 'oj-ux-ico-refresh' },
    display: 'all',
accessTo: ['L', 'F', 'H', 'C', 'S', 'A', 'O', 'T', 'WFCC']
}, {
    label: 'Extrab',
    icons: { start: 'oj-ux-ico-refresh' },
display: 'all',
    accessTo: ['L', 'F']
}
]
```

And also the method which needs to be executed on the button click

```
self.extraa = function(data){
console.log("it got called");
}
```

Note: The function name should be same as label of the button (in lower case)

b. In the html file, additional buttons attribute needs to be included like this:

```
<fsgbu-ob-cmn-fd-work-list id='completedTaskGridCCA' dashboard-id='STANDARD'
dashboard-queue-name='ACQUIRED' dashboard-queue-type='L' worklist-
columns='{{columnArray}}' additional-columns='{{additionalColumns}}'
additional-buttons='{{extraButtons}}' page-size=20>
</fsgbu-ob-cmn-fd-work-list>
```

c. In the json file, the methods which would be implemented on the custom button click needs to be exposed

```
"methods": {
  "extraa": {
    "description": "Would be implemented on Extraa button click"
  },
  "extrab": {
    "description": "Would be implemented on Extrab button click"
  }
}
```

9.20 Steps to create common-extended folder for extending configJSON.js file

- Create a folder inside extended-components\js\components.
- Folder structure \common-extended\js\util.
- Next we will add a file configJSON.js in the created folder.
- The code inside this configJSON.js would be like

```
define(['cmn-util/configJSON'], function (baseobj) {
baseobj.applicationObject.entityIdByProcessCode['CUSTOM'] = {'ccName': 'fsgbu-
ob-remo-deposit-ct-process-flow', 'Name': 'RD Amount Block', 'shortName': 'RD
Amount Block'};
});
```

- Some understanding of the code: -
 - Including the base object by giving the path of configJSON.js base file.

- Then for example adding the entry for custom process as shown above.
- The extended configJSON file would be loaded from base commonFunction.js
- Insertion of the below script into PRODUCT_EXTENDED_LEDGER table

```
Insert into PRODUCT_EXTENDED_LEDGER
(ID,CCA_NAME,CCA_TYPE,PARENT_CCA_NAME,PRODUCT_NAME) select nvl(new_uuid
,'common-extended','config',null,'EXTENDED_COMPONENTS'from
PRODUCT_EXTENDED_LEDGER;
```

9.21 Customizing Existing LOV Fetch Result

- Inscope Datasegment can be used for addition of new fields. (using jquery, at any position, we can add the field)
- Service Extensibility to be used for overriding the base method, OBX tool will generate the base service jar from base service war and this jar should be used to override the base service method and implement the custom changes.
- From UI, call will go to custom service , from custom service, call will go to base service for base field persistence as Java to Java call, then custom functionality to be implemented for persistence of custom fields as part of REST call to another custom service.
- For LOV data, custom projection service to be written. Custom Event needs to be raised while custom fields persistence. For base fields, a call can be made from projection service to base service to fetch data and persisting over the projection schema.

9.22 Steps for adding Pre/post methods in extended components

Suppose here we consider that we want to persist custom fields on postnext call (which means first 'self.next' method of base would get called and then the control will come in postnext method written in extended component).

- Write postnext method in .js file of the extended component – wherein you can call the custom Api for persisting the custom fields.
- Expose this method in the .json file of the extended component.
- Similarly we can add prenext method as well.(it would get executed before 'self.next' method of base executes).
- Note : The hooks for these methods to work should be a part of common infrastructure components in appshell.

Below is the list of CCAs and the common methods which has pre and post hooks :

CCA Name	Common method name	Pre hook present	Post hook present
fsgbu-ob-cmn-ct-authorization	compare	No	Yes
	approve	No	Yes
fsgbu-ob-cmn-ct-act-summary-template	delete	No	Yes
	reopen	No	Yes
	close	No	Yes
fsgbu-ob-cmn-ct-maintenance	save	Yes	Yes
fsgbu-ob-cmn-ct-wizard	next	Yes	Yes
	previous	Yes	Yes
	saveClose	Yes	Yes
	cancel	Yes	Yes
	hold	Yes	Yes
	Applicable for custom footer buttons as well	Yes	Yes
fsgbu-ob-cmn-ct-rs-authorization	approve	No	Yes
fsgbu-ob-cmn-ct-summary-template	delete	No	Yes
	open	No	Yes
	close	No	Yes

9.23 ENDPOINT Overrides

To enhance the endpoint override extensibility, we've added a new column, CCA_NAME, to the PRODUCT_SERVICE_EXT_LEDGER table.

This column provides an extensibility for overriding the existing endpoint behaviour for specific UI components.

How to configure:

1. Determine the component for which you want to override the endpoint.
2. Enter the component's name in the CCA_NAME column of the PRODUCT_SERVICE_EXT_LEDGER table.
3. PRODUCT_NAME & ENDPOINT_KEY must be same as endpoint we are extending.
4. The ENDPOINT_VALUE field should be populated with the new endpoint URI, while the SERVICE_NAME field should specify the corresponding service to which this endpoint belongs.
5. An entry of extension service should also be present in PRODUCT_SERVICE_CTX_LEDGER to pick up the new APPID or other properties.
6. If CCA_NAME column contains NULL value, then endpoint override will be applicable across all components subscribed to respective ENDPOINT_KEY.

ID	PRODUCT_NAME	ENDPOINT_KEY	ENDPOINT_VALUE	SERVICE_NAME	CCA_NAME
1	1 OBEDX	CORPORATE_PREFERENCES	/web/v1/corporatemaintenance/corporatenames	obedx-core-service	fsgbu-ob-edx-wd-todays-snapshot

ID	PROD...	ENDPOINT_KEY	ENDPOINT_VALUE	REQUEST...	SERVICE_NAME
1	388 OBEDX	CORPORATE_PREFERENCES	/web/v1/corporatemaintenance	GET	obedx-core-service

How it works:

When a request is made for the component, the ext orchestrator service will now **consult the CCA_NAME** column. If a matching entry exists, the endpoint specified in the ext orchestrator service (PRODUCT_SERVICE_EXT_LEDGER) will take precedence over the existing endpoint of base product.

This new approach offers several advantages:

- Any endpoint can be extended using this approach.
- The PRODUCT_SERVICE_EXT_LEDGER table is independent of product-related flyway updates, ensuring that future changes won't impact existing overrides.
- This extensibility allows for specific endpoint overrides, other components are unaffected with their original endpoints.

9.24 Steps to create util-extended folder

- Create a folder inside extended-components\js\components in app-shell for component you want to make label-changes.
- Folder structure:
<%componentName%>-util-extended\resources\<component-name>\nls.
Example : for sms it would look like: sms-util-extended\resources\sms\nls.
- Next we will add a file bundle.js in the created folder.
- The code inside bundle.js would be like

```
define(['ojl10n!' + window.location.origin + '/<%componentName%>-component-  
server/js/components/resources/<%componentName%>/nls/bundle.js'],  
function (baseLabels) {  
baseLabels.fsgbuobsmsmnusers.lblhomeBranch = "Foreig111n Branch"
```

```
baseLabels.fsgbuobsmsmnusers.lblstatusChangedOn = "Yogesh"    return
{
    'root': baseLabels
};
});
```

- Some understanding for the code: -
 - Including the base labels by giving the path of bundle.js of main component.
 - Then changing the labels accordingly like in the example above -> Home Branch is replaced with "Foreign111n Branch".
 - Returning the labels (including the changes).
- Insertion of the below script into **PRODUCT_EXTENDED_LEDGER** table

```
Insert into PRODUCT_EXTENDED_LEDGER
(ID,CCA_NAME,CCA_TYPE,PARENT_CCA_NAME,PRODUCT_NAME)
select nvl(new_uuid , '<%=componentName%>-util-
extended','util',null,'EXTENDED_COMPONENTS'from PRODUCT_EXTENDED_LEDGER;
```

9.25 Dynamic Data Configuration (DDC)

DDC is an infrastructure component comprising a user interface and a service. It empowers developers to define prepared statements for dynamic data retrieval. The DDC service's response is utilized by UI components or invoking services to render List of Values (LOV) results.

DDC infra can be utilized with OBX code to call endpoint and bind the response.

Prerequisites:

- For domain services to perform dynamic data queries on the domain schema, the @ComponentScan annotation must include the "oracle.fsgbu.plato.validation" where domain services reside.

```
@SpringBootApplication
@ComponentScan({ "oracle.fsgbu.obedx", "oracle.fsgbu.gcs", "oracle.fsgbu.gus", "oracle.fsgbu.plato.flyway", "oracle.fsgbu.
sms", "oracle.fsgbu.plato.common", "oracle.fsgbu.plato.core", "oracle.fsgbu.plato.logger", "oracle.fsgbu.plato.eventhub",
"oracle.fsgbu.plato.batch", "oracle.fsgbu.plato.coherence", "oracle.fsgbu.plato.validation" })
@EnableEurekaClient
@EnableScheduling
@EnableAsync
public class WorkflowApplication extends SpringBootServletInitializer implements WebApplicationInitializer {
```

- A database schema created for the DDC service.
- A configured JDBC data source named jdbc/PLATODYNADATA on the server.
- Configure newly created schema name in PROPERTIES table of PLATO schema.

APPLICATION	PROFILE	LABEL	KEY	VALUE
plato-dynamic-data-services	jdbc	jdbc	flyway.domain.schemas	OBEDX_PLATODYNADATA_DEV1
plato-dynamic-data-services	jdbc	jdbc	flyway.domain.placeholderReplacement	false
plato-dynamic-data-services	jdbc	jdbc	flyway.domain.outOfOrder	true
plato-dynamic-data-services	jdbc	jdbc	flyway.domain.locations	db/migration/domain
plato-dynamic-data-services	jdbc	jdbc	flyway.domain.ignoreMissingMigrations	true
plato-dynamic-data-services	jdbc	jdbc	flyway.domain.db.jndi	jdbc/PLATODYNADATA

Deployment Steps:

1. Deploy the DDC service to the server.
2. Once deployed, the DDC user interface should be accessible.

Configuration steps:

1. Select the desired **product processor**.
2. Specify the **service name**.
3. Define the **unique key** for the data.
4. List the required **columns**.
5. Provide the **from query** to retrieve data.
6. Set the **paging parameters** (if applicable).
7. Determine the desired **response format**.

Dynamic Data Configure

Reset Save Update Delete Get All Exit

Product Processor: PLATO_PASSWORD

Service Name: plato-password-policy-service

Key: test1

Column List: DEFAULT_VALUE

From Query: PLATO_TM_PASSWORD_POLICY_DETAIL where FIELD_NAME=? FIELD_NAME

Paging Param: OFFSET ?offset ROWS FETCH NEXT ?limit ROWS ONLY

Format Response: {"data": %response%, "paging": {"totalResults": %totalResultCount%, "error": "%error%"}}

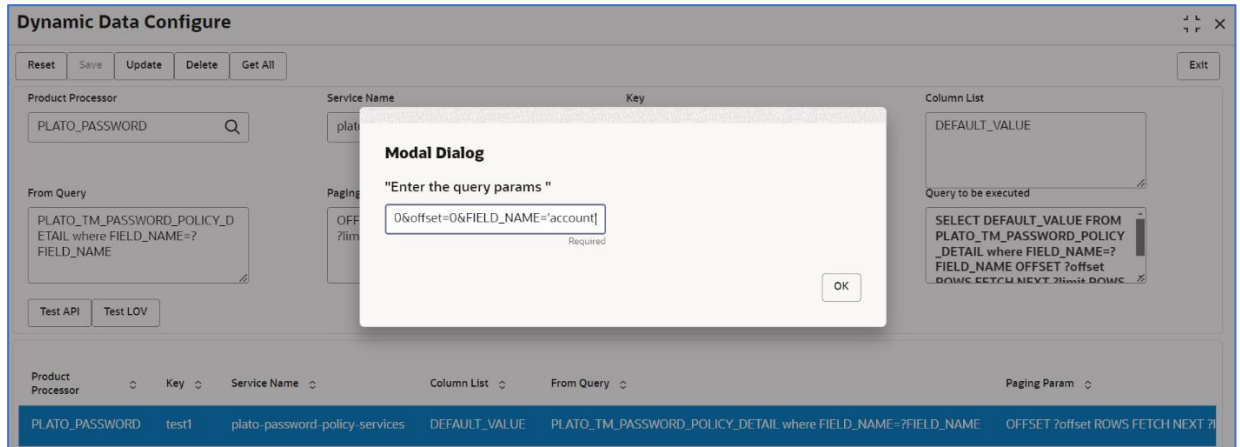
Query to be executed: SELECT DEFAULT_VALUE FROM PLATO_TM_PASSWORD_POLICY_DETAIL where FIELD_NAME=? FIELD_NAME OFFSET ?offset ROWS FETCH NEXT ?limit ROWS ONLY

Test API Test LOV

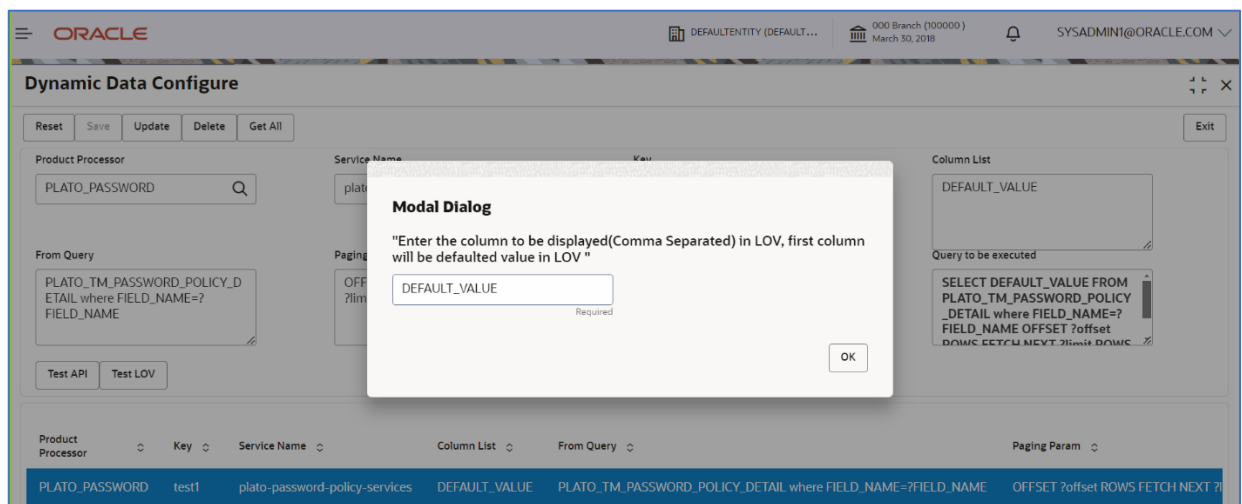
Product Processor	Key	Service Name	Column List	From Query	Paging Param
PLATO_PASSWORD	test1	plato-password-policy-services	DEFAULT_VALUE	PLATO_TM_PASSWORD_POLICY_DETAIL where FIELD_NAME=? FIELD_NAME	OFFSET ?offset ROWS FETCH NEXT ?limit ROWS ONLY

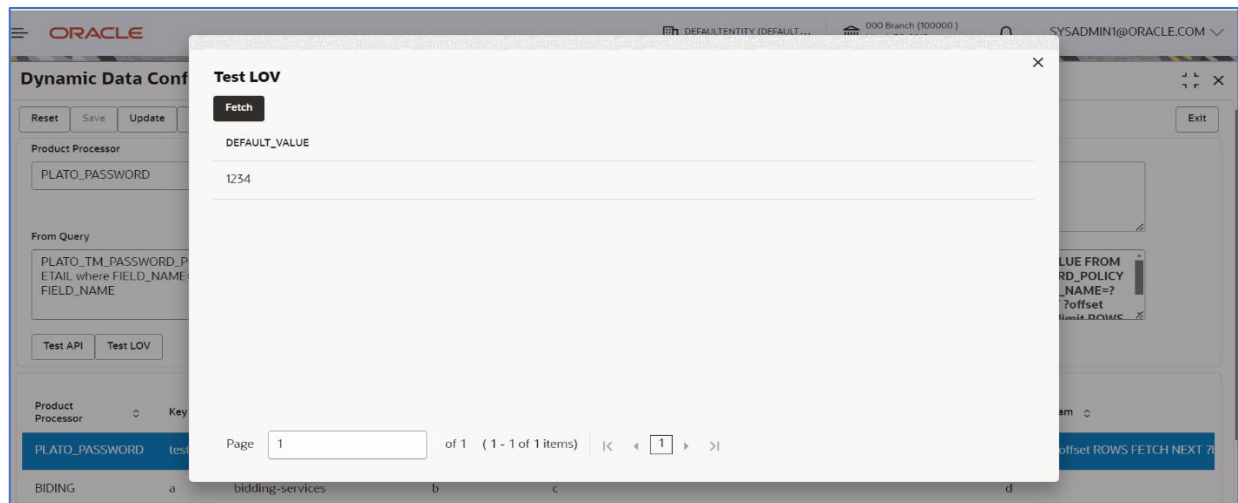
Test Query:

- **Test API:** Use the test API to execute the query. Provide any necessary query parameters and click "OK." The results will be displayed based on the query.



Test LOV: If applicable, use the test LOV (List of Values) to test the query.





Once satisfied with the results, save the dynamic data query.

9.26 Task Screen Custom Config

This document outlines how to customize the task screen using a CUSTOM_CONFIG table ,you can show or hide existing columns, and even add additional filters to the task search screen for specific fields.

Prerequisites:

Ensure that all columns on the task screen are listed in the CUSTOM_CONFIG table present in PLATO_ORCH schema. All the default columns would already be present in this table.

- For all the default columns in Task screen, TASK_SCREEN_VIEW column value will be set to YES by default. If consulting wishes to hide any default column, they can set it to 'NO'.
- Also, if they wish to add new custom column, they need to add the key (key in which we will get the value of custom field in response of task screen 'plato-orch-service/api/v1/extn/tasks' api) of that column in CUSTOM_FIELD_NAME column in CUSTOM_CONFIG table.

Adding Custom Filters

1. Determine the custom field for which you want to add a filter.
2. Update CUSTOM_CONFIG table:
 - Add the field name (custom field key) to the CUSTOM_FIELD_NAME column.
 - Set the SEARCH_SCREEN_VIEW column to YES for this field

Once these changes are made, the additional filter will be displayed on the search screen's UI.

CUSTOM_FIELD_NAME	MAPPED_...	TASK_SCREEN_VIEW	SEARCH_SCREEN_V...	WIDTH	ORDER_N...
1 busProcessCode	COLUMN4	NO	YES	200	11
2 priority	(null)	YES	NO	75px	1
3 processName	(null)	YES	NO	170px	2
4 processRefNo	(null)	YES	NO	200px	3
5 applicationNumber	(null)	YES	NO	165px	4
6 stage	(null)	YES	NO	200px	5
7 startTime	(null)	YES	NO	200px	6
8 branchCode	(null)	YES	NO	120px	7
9 referenceNumber	(null)	YES	NO	200px	8
10 customerNumber	(null)	YES	NO	200px	9
11 amountWithCurrency	(null)	YES	NO	200px	10

Menu Item Search...

Tasks

Awaiting Customer Clarification

Business Process Maintenance

Completed Tasks

Free Tasks

Hold Tasks

My Tasks

Search

SubProcess Tasks

Supervisor Tasks

Select Branch

Processes/Tasks

Priority

Process

Amount

Additional Filters

busProcessCode

Hiding/Adding Columns on the Task Screen

1. Identify the column you want to hide.
2. Update CUSTOM_CONFIG table:
 - Set the TASK_SCREEN_VIEW column to NO for that column.

After updating the configuration, the column will no longer be visible on the task screen.

CUSTOM_FIELD_NAME	MAPPED_...	TASK_SCREEN_VIEW	SEARCH_SCREEN_V...	WIDTH	ORDER_N...
1 busProcessCode	COLUMN4	NO	YES	200	11
2 priority	(null)	YES	NO	75px	1
3 processName	(null)	NO	NO	170px	2
4 processRefNo	(null)	YES	NO	200px	3
5 applicationNumber	(null)	YES	NO	165px	4
6 stage	(null)	YES	NO	200px	5
7 startTime	(null)	YES	NO	200px	6
8 branchCode	(null)	YES	NO	120px	7
9 referenceNumber	(null)	YES	NO	200px	8
10 customerNumber	(null)	YES	NO	200px	9
11 amountWithCurrency	(null)	YES	NO	200px	10

My Tasks

Refresh
Release
Escalate
Delegate
Flow Diagram

<input type="checkbox"/> Edit	Priority	Process Reference Number	Application Number	Stage	Application D
<input type="checkbox"/> Edit	1	300ILCI012269	300ILCI012269	TEST STAGES5	18-05-05

Similarly , we can even add new custom column in Task screens. For this they need to add the custom field name(key in which we will get the value of custom field in response of task screen 'plato-orch-service/api/v1/extn/tasks' api) of that column in CUSTOM_FIELD_NAME column in CUSTOM_CONFIG table.

Configurations needed from backend

Configurations needed from backend side to get the custom field in 'plato-orch-service/api/v1/extn/tasks' response –

During workflow initiation, the customer provides key-value pairs for specific columns. In the CUSTOM_CONFIG table, columns are mapped under the MAPPED_COLUMN_NAME field. For instance, COLUMN4 is mapped to a custom_field_name, such as CustomField.

Here's how it works: In the CUSTOM_CONFIG table, COLUMN4 is mapped to the field CustomField. During workflow initiation, the customer provides the value for COLUMN4, such as COLUMN4 = CF_1. The system uses this mapping to interpret the value as follows: CustomField (from the CUSTOM_CONFIG mapping) will get the value CF_1 for that task , provided by the customer during initiation. This allows the customer to input COLUMN4 = CF_1 during workflow initiation, and it will be mapped with CUSTOM_FIELD_NAME based on the mapping defined in the CUSTOM_CONFIG table This way, you can map any internal column to a custom field name that suits your specific use case.

Additionally, the columns that can be used for such mappings currently range from COLUMN1 to COLUMN20, providing flexibility to define up to 20 custom fields.

Welcome Page Platoorch CUSTOM_CONFIG HTASK_ADDN_DTLS					
Columns Data Model Constraints Grants Statistics Triggers Flashback Dependencies Details Partitions Indexes SQL					
Filter:					
CUSTOM_FIELD_NAME	MAPPED_COLUMN_NAME	TASK_SCREEN_VIEW	SEARCH_SCREEN_VIEW	WIDTH	ORDER_N...
1 CustomField	COLUMN4	YES	NO	170px	2

```
{
  "createTime": 1585655473582,
  "updateTime": 1588595146622,
  "name": "MK-SQL",
  "description": "Test Workflow6",
  "version": 2,
  "tasks": [
    {
      "name": "TestTask",
      "taskReferenceName": "test",
      "description": "first test task",
      "inputParameters": {
        "COLUMN4": "CF_1",
        "FUNCTIONAL_CODE": "PLATORULE_FA_FACT_NEW",

```

10 Reference and Feedback

10.1 Reference

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

- Oracle Banking Extensibility Workbench Installation Guide

10.2 Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at

<http://www.oracle.com/us/corporate/accessibility/index.html>

10.3 Feedback and Support

Oracle welcomes customers' comments and suggestions on the quality and usefulness of the document.

Your feedback is important to us. If you have a query that is not covered in this user guide.