Oracle Banking Digital Experience

UX Extensibility Toolkit
Release 19.2.0.0.0

Part No. F25153-01

December 2019

ORACLE®

User Interface Workbench

December 2019

Oracle Financial Services Software Limited

Oracle Park

Off Western Express Highway

Goregaon (East)

Mumbai, Maharashtra 400 063

India

Worldwide Inquiries:

Phone: +91 22 6718 3000

Fax:+91 22 6718 3001
www.oracle.com/financialservices/

Table of Contents

1. Preface 5
1.1 Intended Audience 5
1.2 Documentation Accessibility 5
1.3 Access to Oracle Support 5
1.4 Structure 5
1.5 Related Information Sources 5
2. User Interface Workbench 6
3. UI Workbench Manual Build 8
3. UI workbench installation 9
4. Layout Selection 10
5. Folder Creation 11
6. REST API Selection 15
7. REST API Configuration 19
8. Chain REST APIs 21
9. Design Component 27
10. Available Components 44
11. Available Attributes 76
11.1 Label: 76
11.2 Value: 81
11.3 Options: 91
11.4 Value change handler: 96
11.5 Validations: 101
11.6 Required field: 106
11.7 Add Loop: 107
11.8 Add Custom Attributes: 109
11.9 Conditional Field: 111
11.10 Grid: 116
11.11 Select anchor type: 119
11.12 Add formatter: 123
11.13 Select Size: 127
11.14 Enter Image Path: 128
11.15 Enter Initials: 129
11.16 Select Type: 129

11.17 Selection Mode: 131
11.18 Enter Allowed File Extensions: 132
11.19 Image source: 132
11.20 Enter Minimum Length: 133
11.21 Enter Maximum Length: 134
11.22 Enter Step: 134
11.23 Source variable: 135
11.24 Id attribute: 136
11.25 Renderer ID: 136
11.26 Pagination: 137
11.27 Indexer: 141
11.28 ID: 143
11.29 Enter menu launcher: 143
11.30 Columns: 144
11.31 Row renderer: 148
11.32 Aria label 149
11.33 Tag type: 150
11.34 Binding source: 151
11.35 Enter rows 154
11.36 Selected step: 155
11.37 REST API Chain and Hook Function 157
11.38 Select Type of Container 160

1. Preface

1.1 Intended Audience

This document is intended for the following audience:

- Customers
- Partners

1.2 Documentation Accessibility

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

1.3 Access to Oracle Support

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

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

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

1.4 Structure

This manual is organized into the following categories:

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

The subsequent chapters describes following details:

- Configuration / Installation.

1.5 Related Information Sources

For more information on Oracle Banking Digital Experience Release 19.2.0.0.0, refer to the following documents:

- Oracle Banking Digital Experience Licensing Guide

2. User Interface Workbench

User Interface Workbench is a development tool to design the User Interface of an Application using APIs to fetch data, enabling communication of one screen with another and generating screens to be displayed to the end user. This tool aims at minimizing developers' efforts and time by generating the working screens automatically with minimal user input.

The tool also enables the developers to make the changes in the existing screens created by them or provided out of box, which are designed using the UI Workbench tool.

Components:

The screens created consist of the following components:

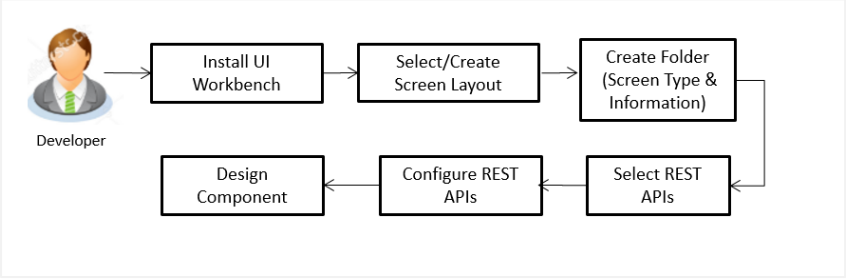
- Template: It decides visual representation of data on the front-end
- Model: It is responsible for fetching data from REST APIs
- Bindings: It consists of logic for processing the data fetched from the APIs
- Resource Bundle: It comprises of strings which are to be displayed on the screen
- Hooks: It comprises of user provided logic.
- Metadata: It consists of summarized information, which can generate all the above mentioned artefacts.

Pre-requisites:

- Basic knowledge of Swagger. (To know more about swagger refer https://swagger.io/)
- Basic knowledge of JavaScript, Knockout Js
- Basic knowledge of Oracle JET
- Basic knowledge of Sass

Workflow:

The process to create the screen ready to be used by the user / customer is as follows:



1. **Install UI Workbench**: The tool **can** be easily installed simply by running the setup file.
2. **Select/Create Screen Layout**: User can select one of the predefined screen layouts according to his requirement

3. **Create Folder**(**Screen Information)**: User has to provide basic screen details like the name of the screen and the type of the screen e.g. (transaction, inquiry, and widget)

4. **Select REST APIs:** User can select multiple REST APIs which will be needed for fetching required information to display on the screen.

5. **Screen Designing**: User can design how the screen content will be displayed by using the form elements (e.g. Input Box, Text Area) available in the tool. User can drag and drop these elements and design the screen as per requirement

Home

3. UI Workbench Manual Build

In the project directory, place folders **obdx-ui-workbench-core** and **obdx-ui-workbench-gui** side by side.

Follow the steps mentioned below to setup your environment for build.

1. Open terminal inside **obdx-ui-workbench-core** and perform **npm install** to setup the workspace and then execute **npm run build** to build the core.
2. Open terminal inside **obdx-ui-workbench-gui** and perform **npm install** and then **npm link ../obdx-ui-workbench-core.** This will locally install the core package to GUI project.
3. Run **npm run build** to create executables for macOS, Windows and Linux. Please note that executables for Windows can only be created on Windows setup and code signing for macOS applications can only be done on macOS operating system. To create executables for Windows and macOS on Linux machine, use Wine and Mono on Linux. For more details you can refer to https://www.electron.build/

   Alternatively, you can use **docker** to simplify the operations as follows and create Windows & macOS builds on Linux by running the following command in **obdx-ui-workbench-gui** folder root:

```
docker                    run                    --rm                   -ti
--env               ELECTRON_CACHE="/root/.cache/electron"               \
--env       ELECTRON_BUILDER_CACHE="/root/.cache/electron-builder"       \
 -v                           ${PWD}:/project                            \
 -v                ~/.cache/electron:/root/.cache/electron               \
 -v       ~/.cache/electron-builder:/root/.cache/electron-builder       \
 electronuserland/builder:wine-mono /bin/bash -c "npm run build"
```

4. The artifacts will be available inside **`uiworkbench/`** directory.

3. UI workbench installation

As a first step, user should download the UI Workbench setup from the Oracle Software Delivery Cloud portal. Once the setup is downloaded, double click to install the tool.

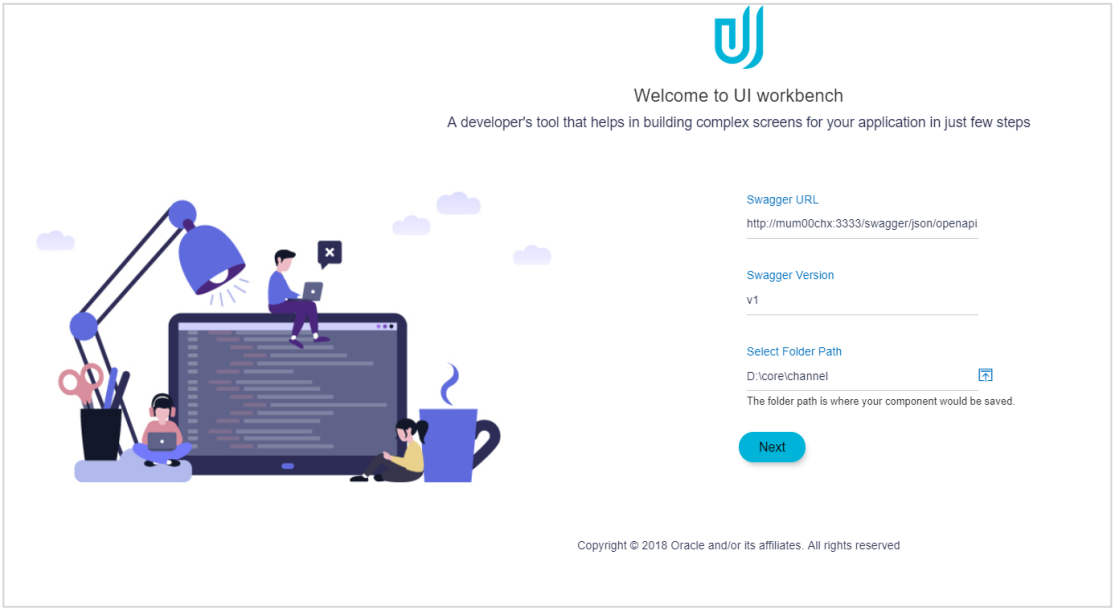
After installation, user would be directed to the following Landing Page. User is expected to provide the URL where the swagger document of the RESTful APIs are hosted.

To know more about swagger refer https://swagger.io/

Swagger version (By default it is v1)

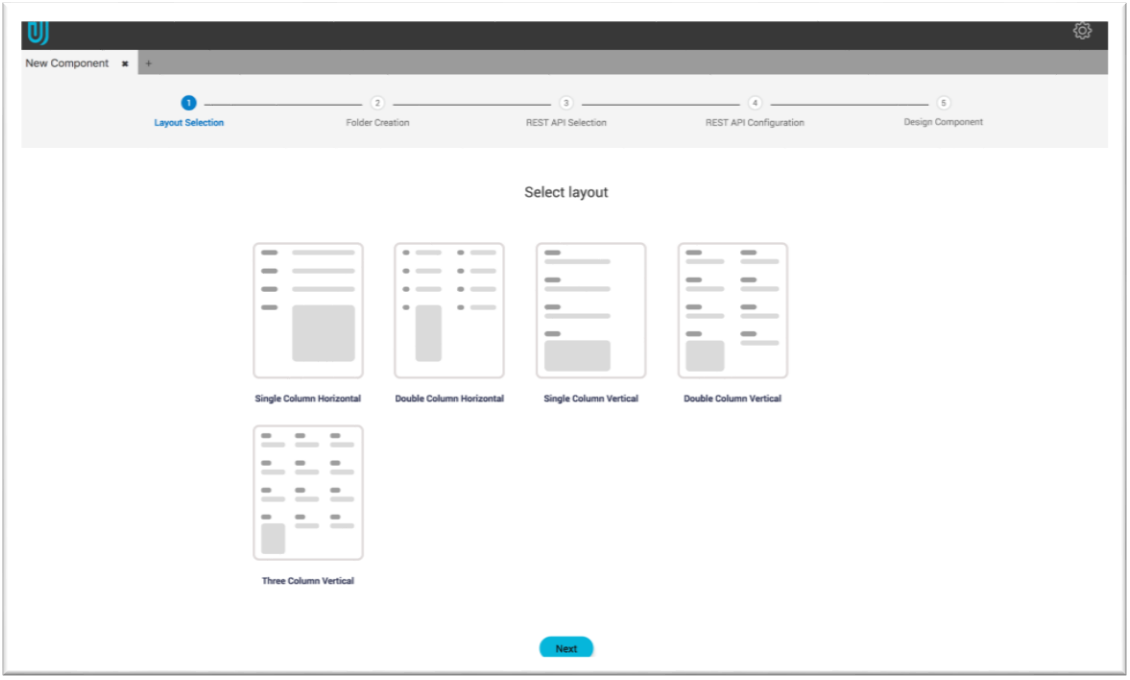
Directory path : <OBDX codebase location>/core/channel

Provide these details at the time of installation. The path can be changed anytime later by clicking on settings. Please note: Every time the path is changed, data is lost.

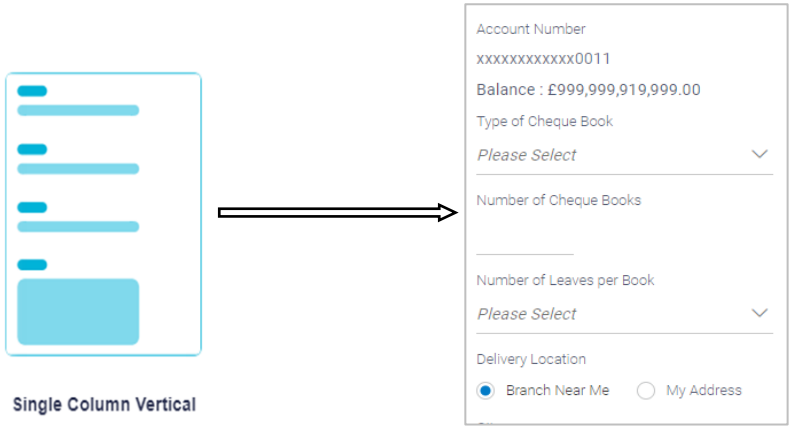


4. Layout Selection

Next step is to select the screen layout. The tool offers five pre-defined templates, which will decide the layout of the screen that the user wants to create. The user can select any of these layout from the list provided or can create a custom layout in case the desired layout is not found in the mentioned list. To know the steps to create the custom layout Please refer **Grid** section.



E.g. incase if user select the following layout, the label and value fields will be vertically aligned and in a single column.



In case the user wants to change the layout once, he has moved on to the next step, he can navigate to this step by clicking back button at the bottom of the screen. Layout can be changed at any given point of time during the process of screen creation.

5. Folder Creation

As a part of next step, following screen is displayed to the user to create the new folder. User is expected to furnish following information.

- **Module Name:**

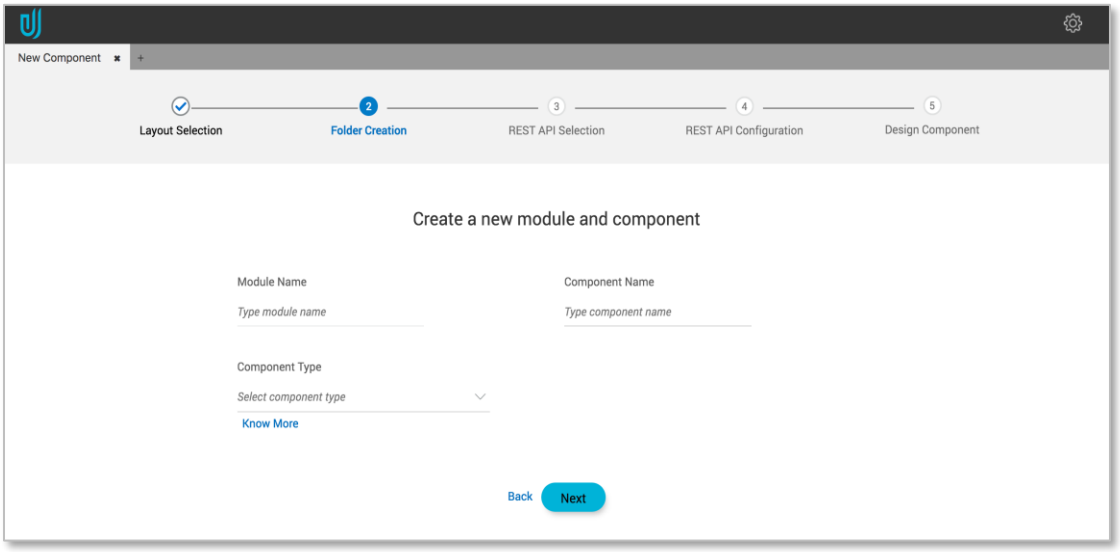
  Module is a category of the screen the user are creating, for e.g. a fund-transfer screen would be of category Payments. So in this case, the module name would be payments. Module consists of various components, which contains the artefacts of the screen the user wants to generate.

- **Component Name:**

  Component name would be the type of screen e.g. Funds transfer, Cheque Book Request, Letter of Credit Initiation, Bill Payment, New Deposit etc.

- **Component Type:**

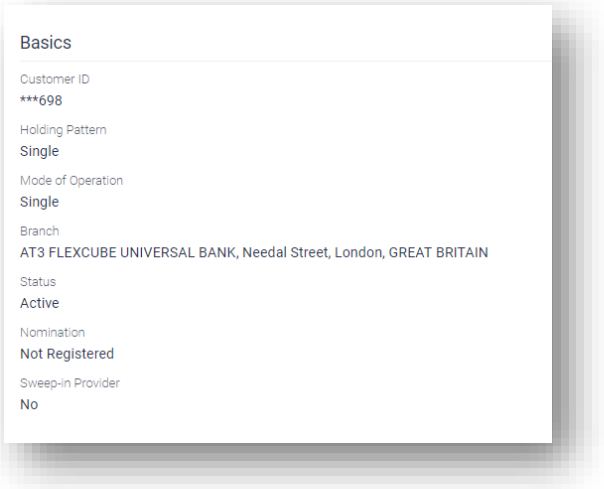
  Component type is the type screen layout the user wants to be displayed. User can select from the options as Individual page Transaction Page or a Widget. (Explained in the detail below)



Following are the different component types and the examples where these are used.

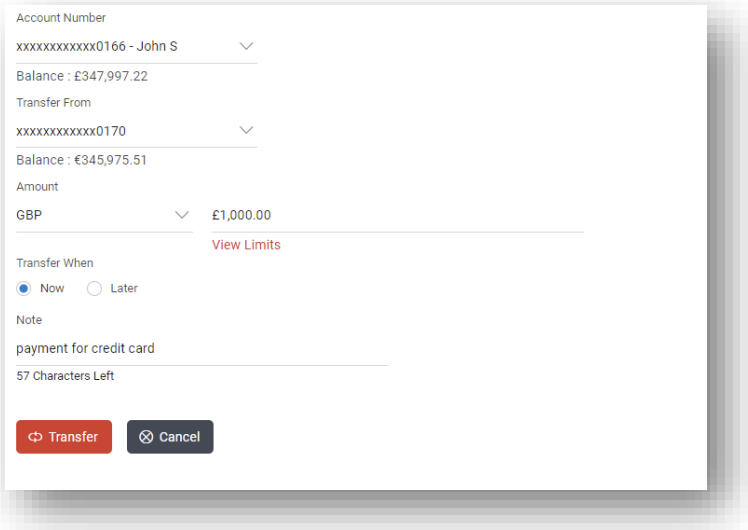
- **Individual Page :**

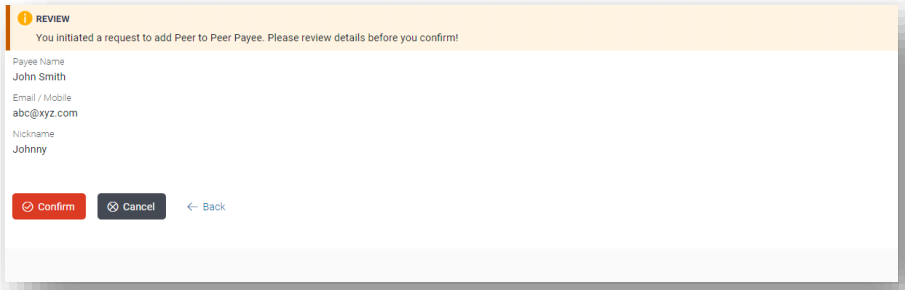
  Individual page is a stand-alone page that consists of the detailed information for the Selected transaction/component e.g. Account Details (Component) in a standalone page shows the various Balances, transactions and other information of the selected accounts.

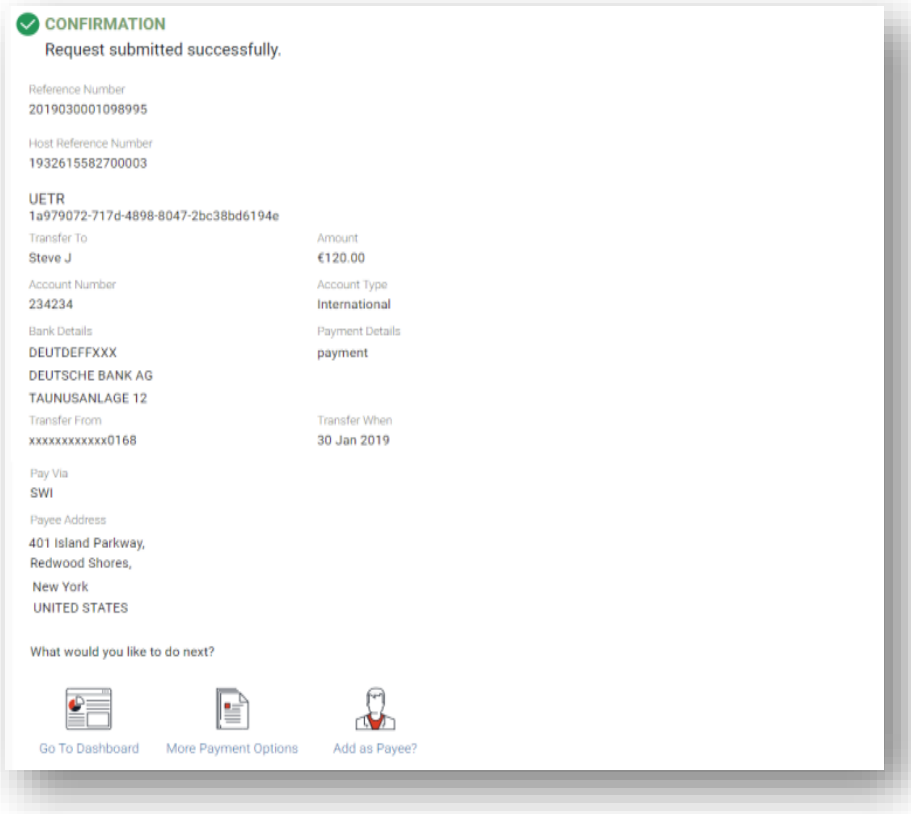


- **Flows:**

Flows consists of pre-defined multiple pages in accordance to OBDX UI framework. For e.g. A transaction where the customer or user enters information asked on the screen, reviews the same and submit it for further processing. It consists of multiple screens.







In case of such scenario, one must select component type as FLOW.

Types of FLOW supported are

1. CREATE:

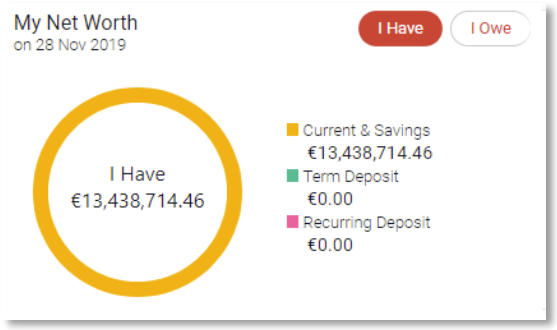
This is an n-step flow, which is similar to a transaction. The only difference is the initiation page can be any number of pages as per the requirement and not only one. For e.g. first page can comprise of primary details of a customer e.g. name, surname and second page can consist of additional information like address. Review and confirmation page will each consist of a single page only.

Once the flow type is selected as CREATE, additional details are required to be filled such as

- Flow name : The name will be the identifier of the entire flow
- Number of Stages : The number of pages before the review screen appears in the flow
- Stages navigation display: The type of navigation control to navigate between the pages
- Confirmation Template: Type of layout for confirmation screen
- Show summary : if summary is to be shown about the progress of the form filled and continue where the user had left the last time, the switch can be toggled to YES

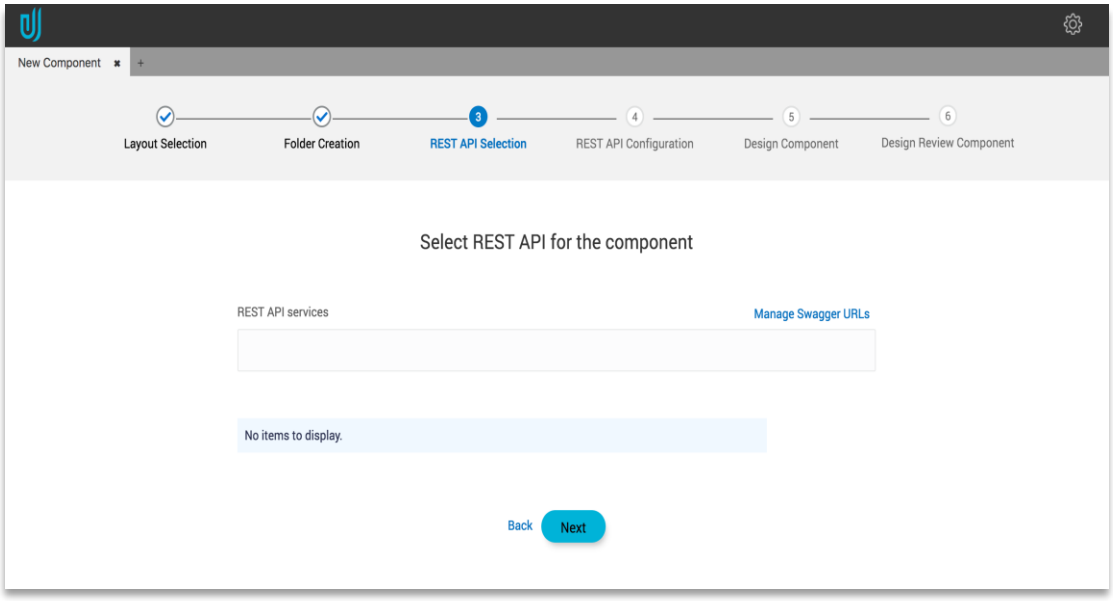
- **Widget -**

  Widget is a small section on the screen especially on the Dashboard that displays commonly used functions or important information in a summarized form. The component generated of this type will be shown on the dashboard.



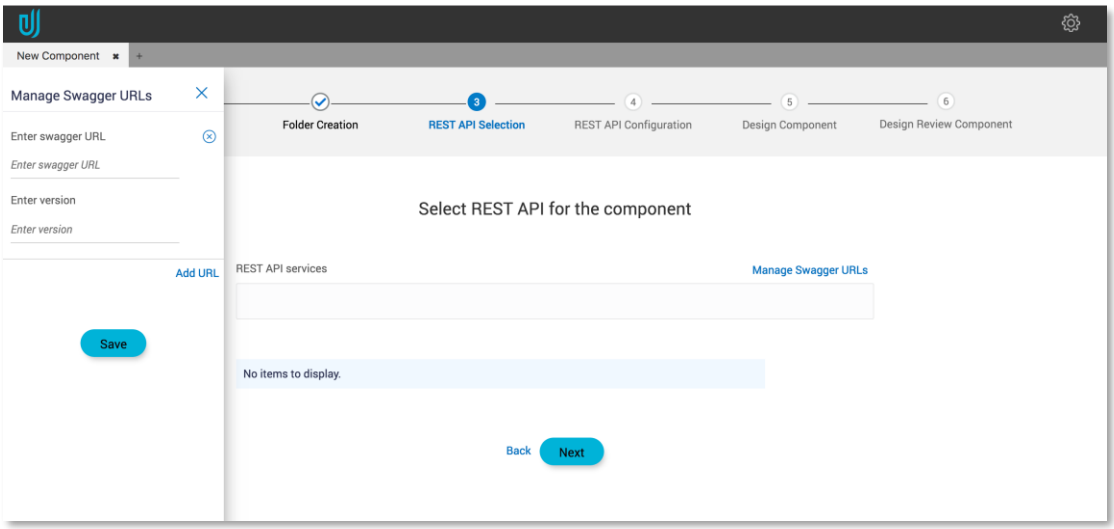
6. REST API Selection

In this step, the user has to select the REST APIs to enable the functioning of the elements on the screen. The dropdown will contain all the REST APIs mentioned in the swagger document hosted on the URL which user had specified in the landing page.



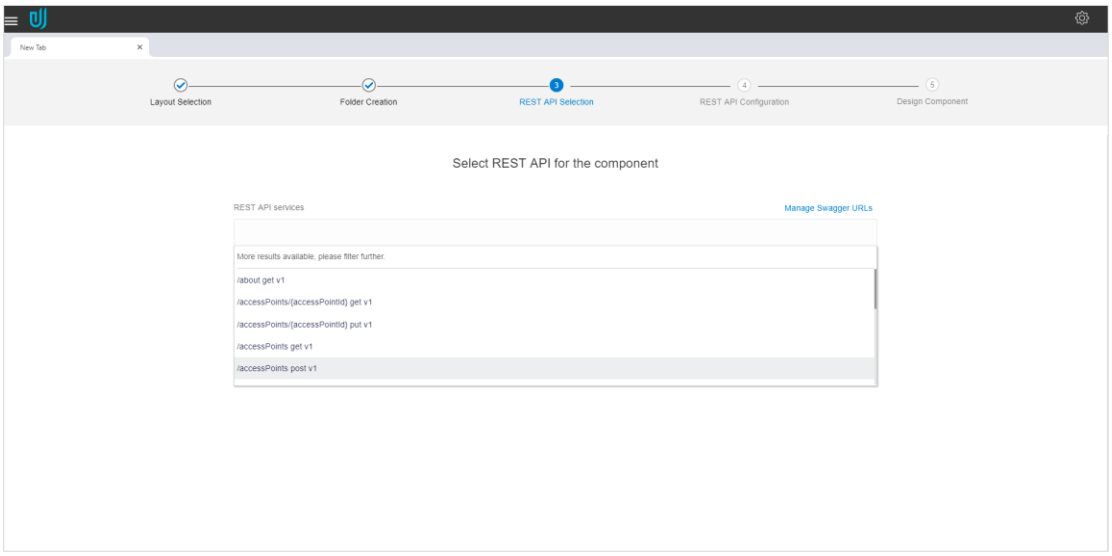
In case, the user needs REST APIs whose documentation is hosted on other URLs apart from the ones the user had mentioned earlier, the user can add them by selecting manage Swagger URLs link.

On clicking the link, a panel will open on the left side. By default, it will contain the URL and the version, the user had specified during installation. User can change this URL or can add multiple URLs by clicking on ‘Add URL’ text link, depending upon the requirements.

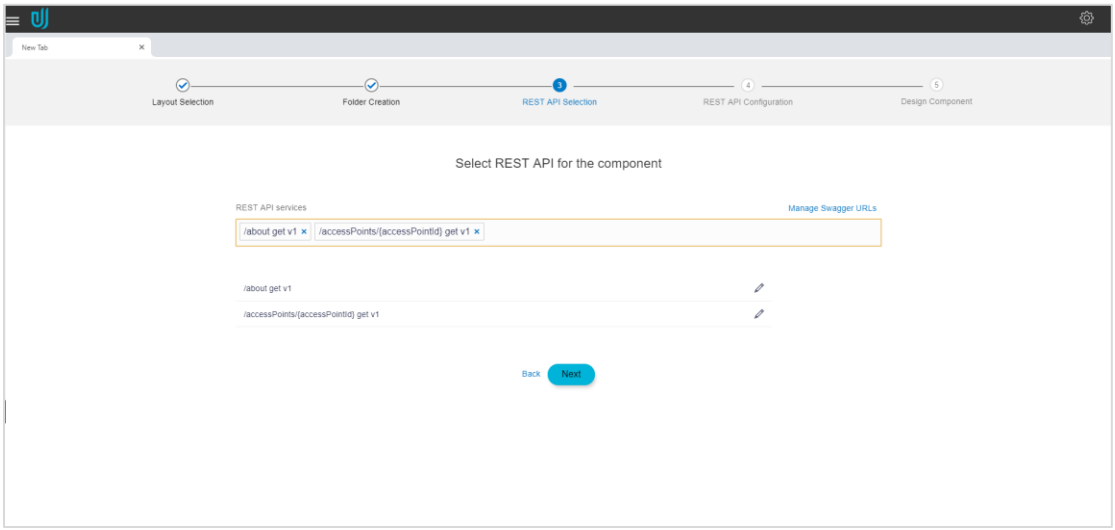


Once the user has entered the URLs he can save the details by clicking the save button. After the dialog box closes, all the REST APIs will be consolidated and are available in the dropdown

User can select multiple RESTs as per requirement. All the selected RESTs are shown in the list view form.



User can configure the REST properties by clicking on the edit icon.



On clicking on the edit button, a panel will open on right, which has three configuration options.

Parameters are the options that the user can pass with the endpoint (such as specifying the response format or the amount returned) to influence the response.

- **Required Parameters:**

  They are also known as path parameters. These parameters are part of the REST URL written within curly braces. E.g. /accessPoints/ {accessPointId}.Here accessPointId is the required parameter.

- **Optional Parameters:**

  They are also known as query parameters. These parameters are part of the REST URL written at the end of the URL followed by? Symbol. E.g. /accessPoints? NoOfpoints=3. Here NoOfPoints is an optional parameter.

The value of these parameters can be set by two ways.

1. Current component sets the value.
2. Value is passed to the current component from the previous component.
   In case the parameter values are being passed from the previous component, the variable name has to be mentioned in this pane
   The current parameter value is set in later steps, refer **Value** attribute.

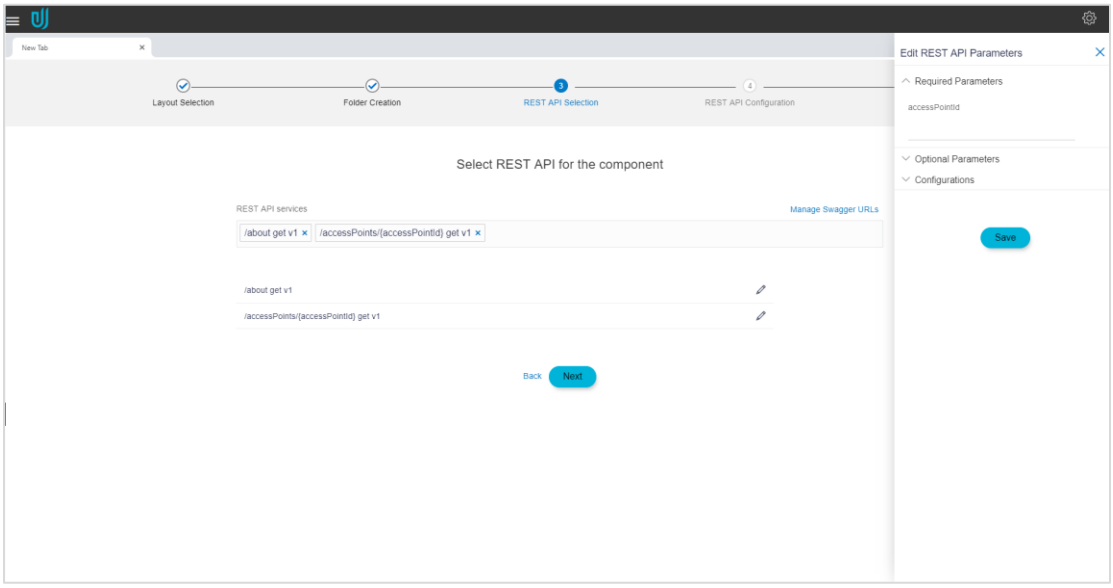
- **Configurations:**

  The GET REST API can be used for two purposes.

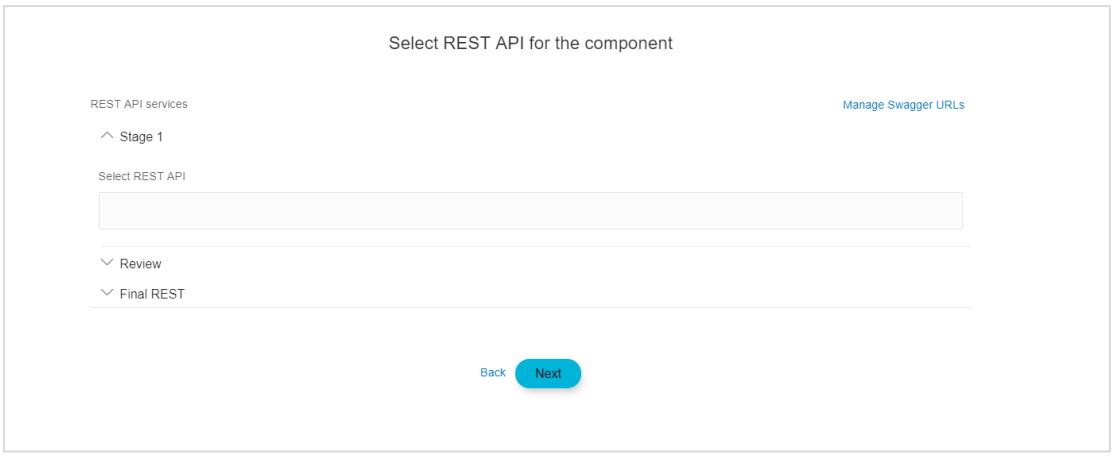
  1. To fetch details

  2. To download a file

  If user wants to use GET REST API as a download service, user can enable the switch named as **'Use as download Service'** in configurations section



In case of FLOW, user can select RESTs for each step.



The final REST dropdown value should consist of the REST that will be fired once the user has confirmed the details and which will make the concerned transaction.

7. REST API Configuration

In this step, the user can configure how the selected REST APIs will be called when the screen is loaded. There are two sections in this step.

- **Chain selected REST APIs**

  There are 3 ways how REST APIs can be called:

  - **Independent**

    If all the selected REST APIs are to be called irrespective of any dependency on other APIs, then the user has to select the NO option.

  - **Sequential**

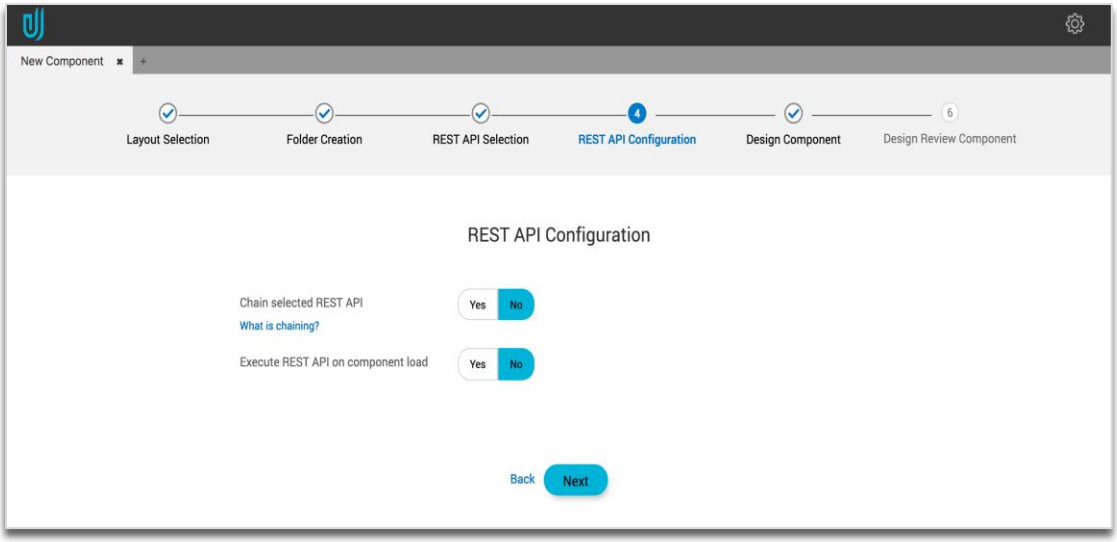
    If a REST API calling is dependent on the response of other REST APIs then such case is known as sequential manner. For e.g. A REST API fetching details of cities in a country is dependent on the response of API fetching the list of countries. If REST APIs are to be called in such a manner, select the option ‘Yes’.

    On selecting the ‘Yes’ option, a panel appears on the left having list of selected REST APIs

  - **Parallel**

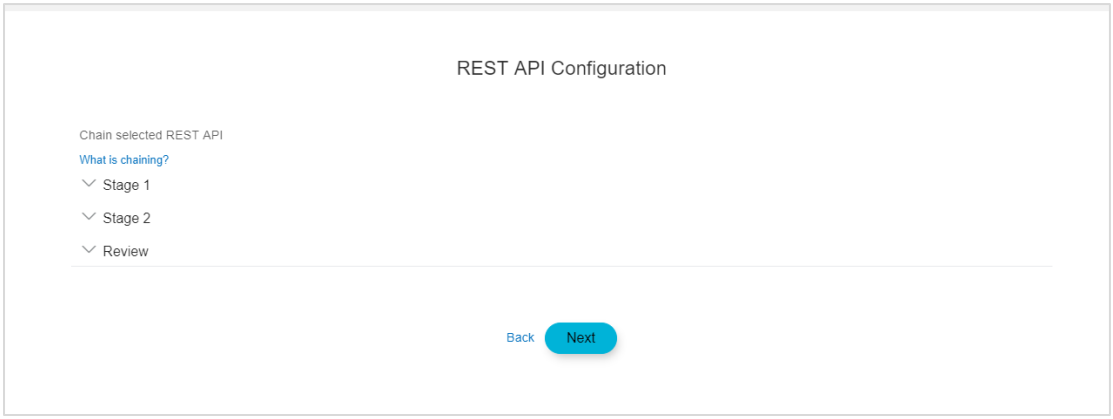
    If REST APIs are to be called together and the combined response of these APIs are needed then they are to be called parallelly. If REST APIs are to be called in such a manner, select the option ‘Yes’.

    On selecting the ‘Yes’ option, a panel appears on the left having list of selected REST APIs



How to chain Rests, refer Chain RESTS.

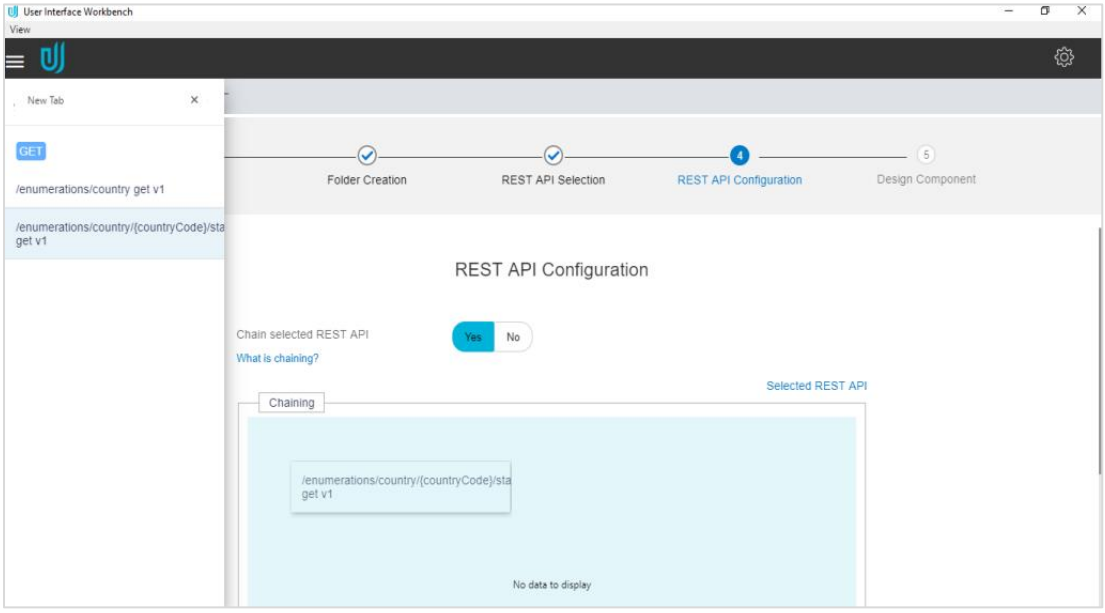
In case of FLOW, user can configure RESTs for each step.


REST API Configuration
Chain selected REST API
What is chaining?
Stage 1
Stage 2
Review
Back
Next

8. Chain REST APIs

User is expected to chain the REST APIs, if the REST APIs are to be called parallel or sequentially. To chain the APIs, the user must select 'Yes' option in the previous screen. Once the user selects 'Yes', a panel appears on the left having list of selected REST APIs.

To start chaining, user needs to drag the APIs and drop it in the chaining section shown in the middle section of the screen.



Once the user has dropped the REST APIs in the middle section, a right panel gets opened where user has to provide information about the REST type and if the RESTs are parallel independent or dependent.

Some common terms used in chaining are:

REST Type:

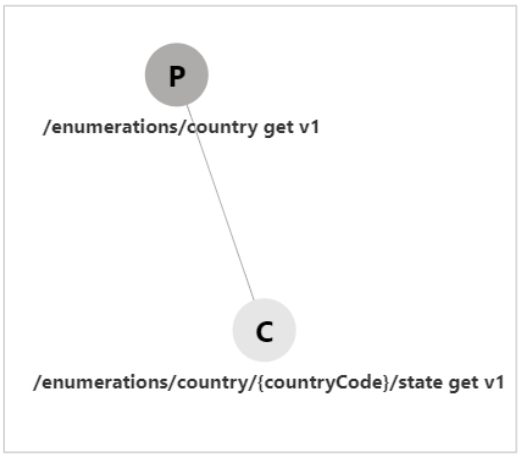
- **Parent** – If this option is selected then, every service call flow needs a parent rest, which will be called first.
- **Child** - If this option is selected then, service to be called after parent call finishes.

Is this REST Parallel Independent or Parallel Dependent?

- **Parallel Independent** – On selecting this option, service calls can be fired independently.

- **Parallel Dependent** – On selecting this option, service calls logic, which depend on response of an immediate parallel service call, those immediate service calls needs to be parallel dependent

The chaining structure is similar to tree structure. The top most node would be parent and the other nodes would be child nodes.

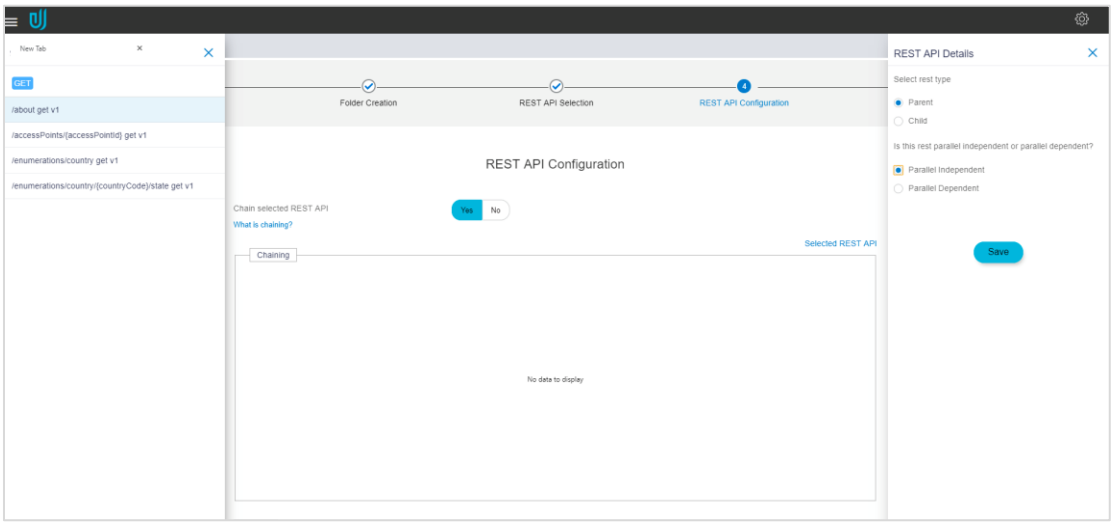


If user selects REST type as Parent then the node with letter P will be generated.

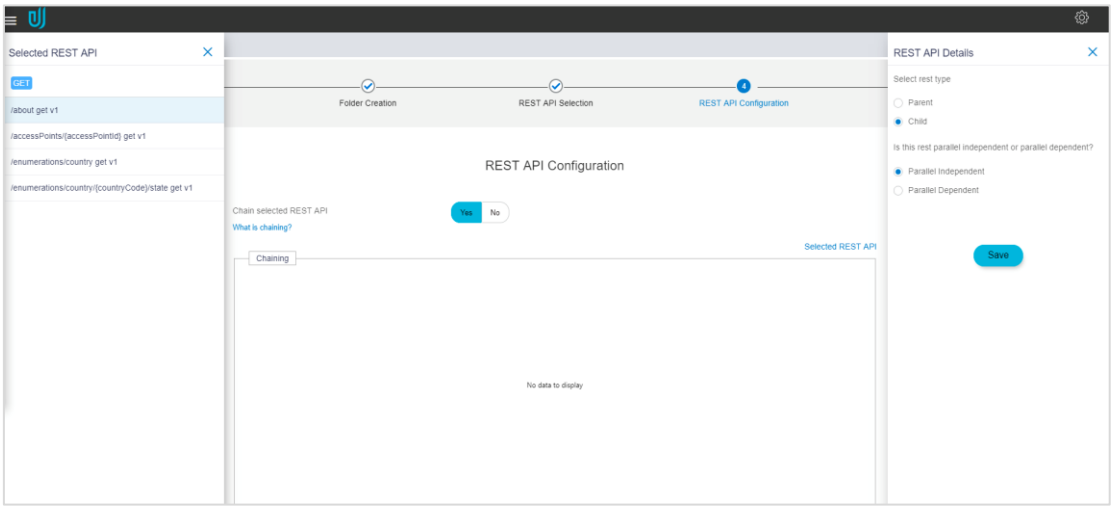
If user selects REST type as Child then the node with letter C will be generated.

Consider an example if two REST APIs are to be called in a sequential manner.

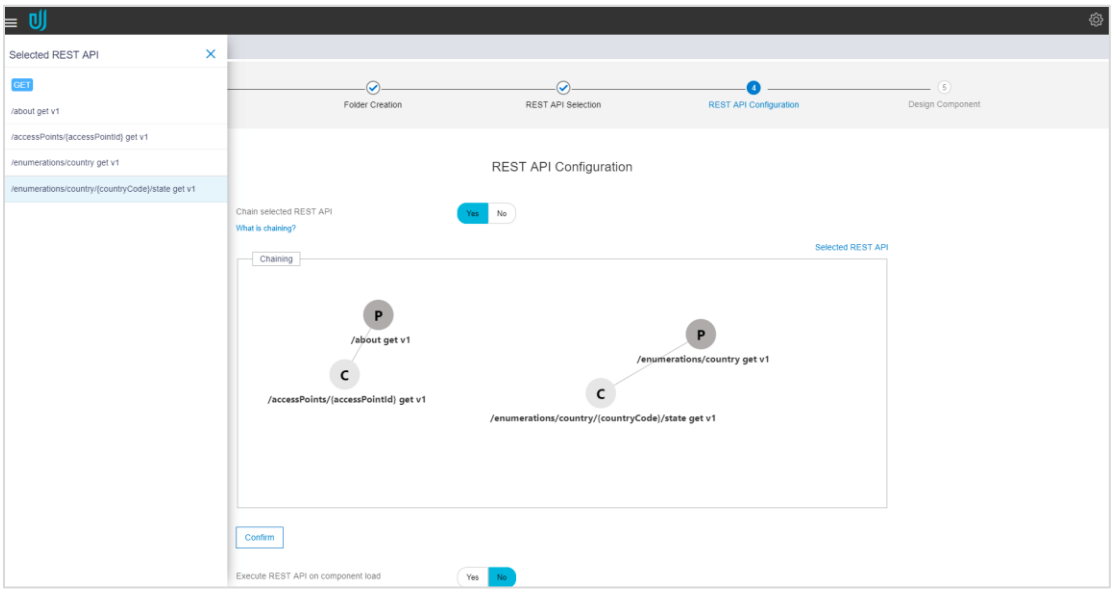
The first API to be called will be declared as Parent and since no other API is to be called along with this API, it will be parallel independent.



The second API which is to be called after the response of the first API is received, will be declared as a child and parallel independent

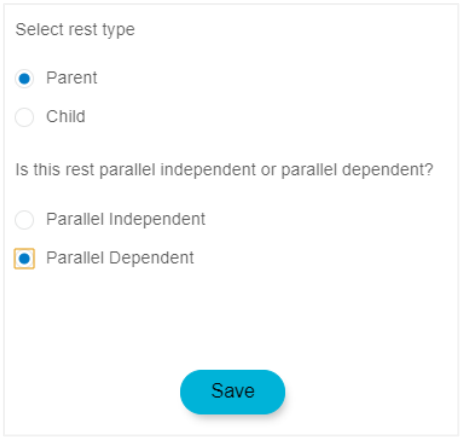


To establish the linkage between the parent and its child nodes and distinguish them from each other, the connection has to be made. To make a connection, drag the child REST from the chaining section, and drop it on the Parent REST and then drag the child node down, the connection will be made.



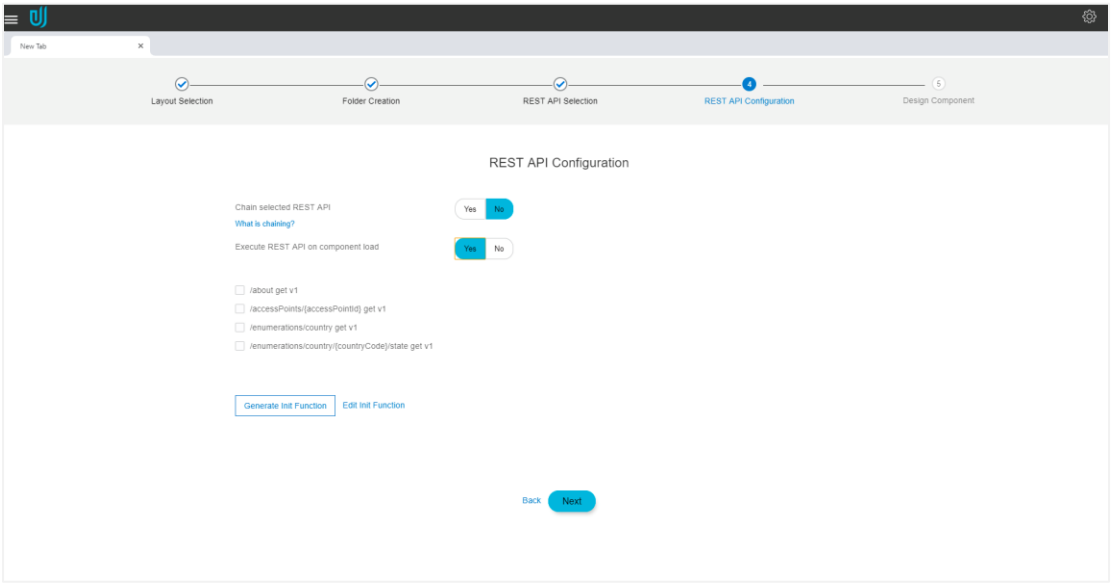
The connection is only to be established if the services are to be called in sequential manner.

In case, all the services are to be called together and the wait till response of each service has been received, connection will be not established as they all act as parent nodes. In such case, all REST APIs will be configured as parent node and will be parallel dependent since they will have to wait for the response of the selected APIs.

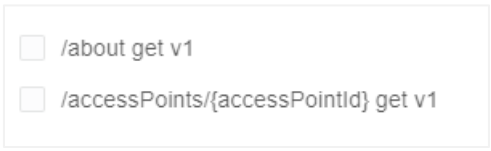


If out of the selected RESTs, some of them are sequential or parallel Dependent and rest of them are parallel Independent, the chaining is still to be performed. To declare a REST to be independent of other rests, select the rest type as Parent and Parallel independent and do not make any connections with those RESTs. Incase all the REST APIs are independent of each other then there is no need of chaining.

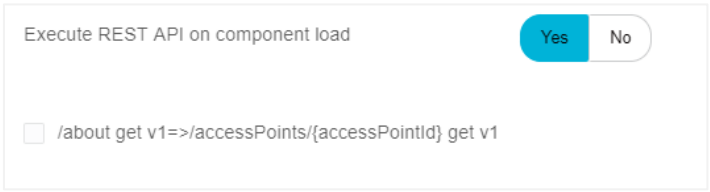
If user wants to call the REST APIs as soon as the screen loads, the option ‘Yes’ needs to be selected.



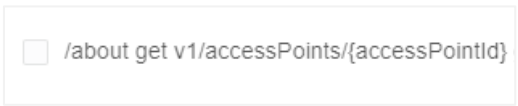
Once the yes option is selected, the REST Chains will be shown to user. In case of independent RESTs their name will appear.



If there is a sequential rest then it will be displayed as



If there is a parallel dependent chain then it will be displayed with their names together

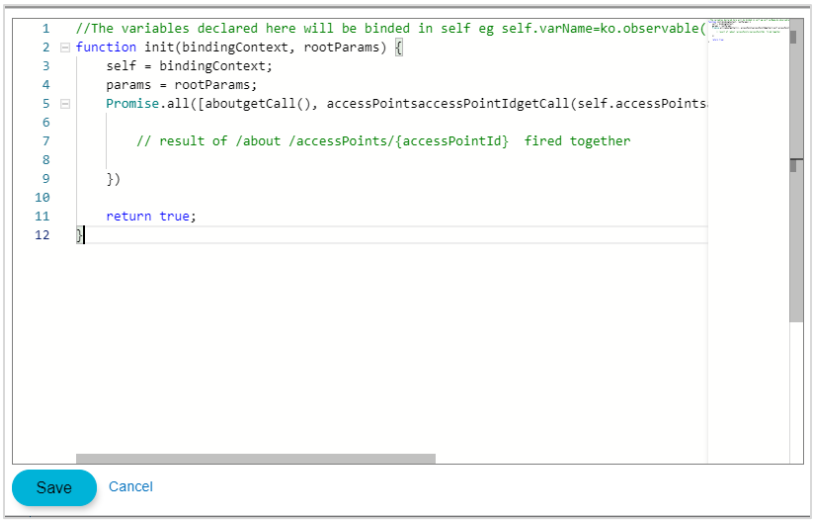


User can select which REST chains he wants to call when the screen loads. Once he has selected the chains, user needs to click on generate init function to generate the code to fire the services on the screen load.

Generate Init Function

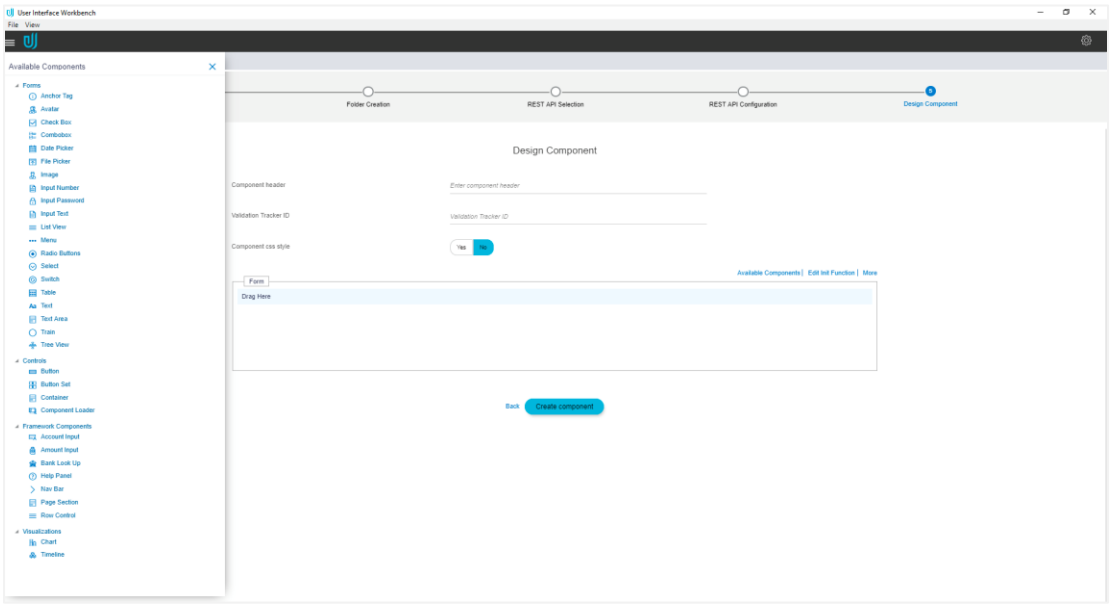
Once the function is generated user can click on Edit Init Function link to edit the code to perform desired actions.

Edit Init Function

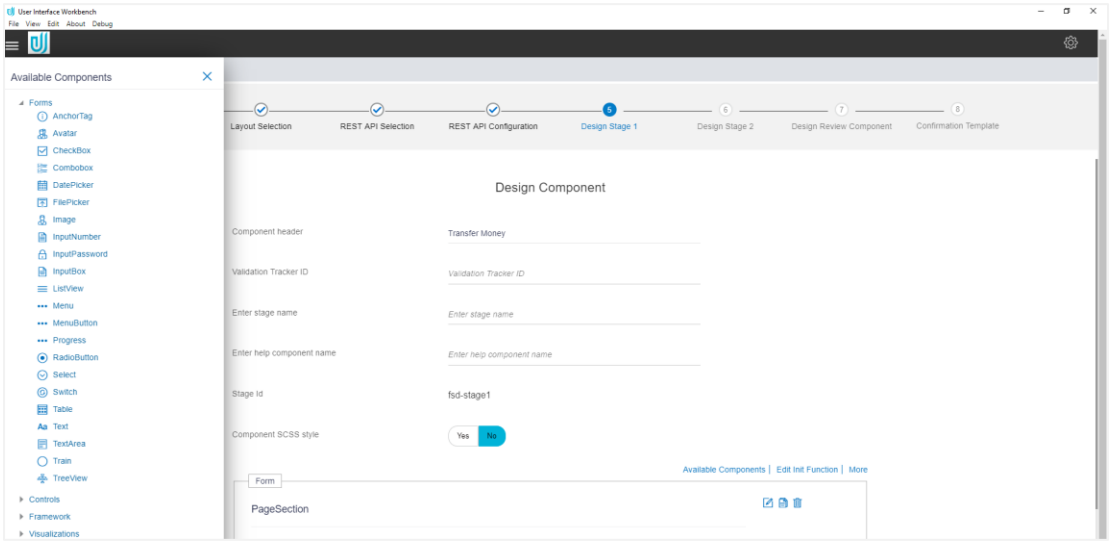


9. Design Component

This step enables the user to design how the contents of the screen are to be displayed. User can make use of available form elements e.g. Input Text, Button etc. Each element can further be configured, placed and styled by the user as per the screen requirement.



If the type of component is FLOW, the name of the stage changes to Design Stage 1, Design Stage 2, depending on the number of stages



The following information is captured on this screen.

- **Component Header**

  The text entered in this field will be displayed as the Page Title. E.g. Cheque Book Request, Loan Repayment etc.

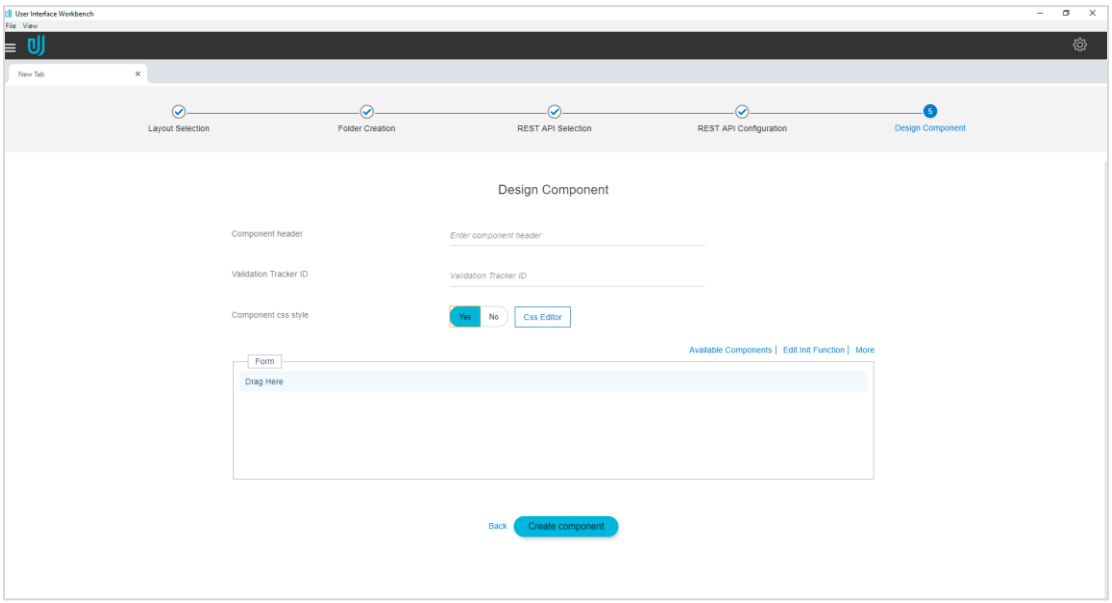
- **Validation Tracker ID**

  The ID provided by the user should be the ID of the element within which the form elements reside. The tool by default generates this ID. So this is an optional field. However, there may arise a situation where the user needs to use such an ID. In such case, user can provide custom ID in this field.

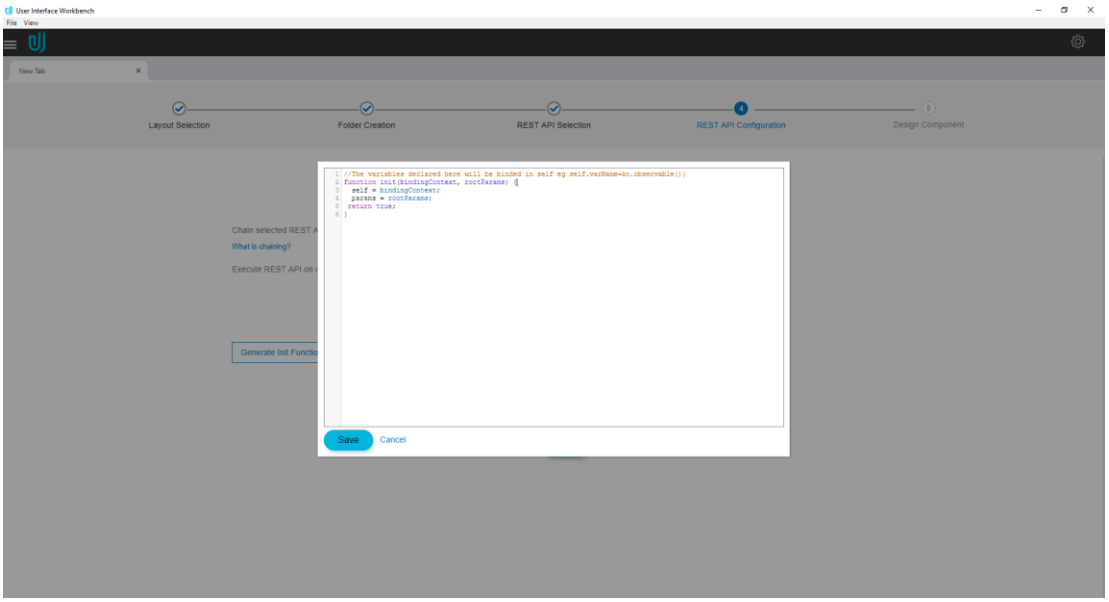
- **Component CSS Style**

  User has the provision to provide custom style sheet (CSS) for the screen he wants to generate and change the look and feel of the component as per his requirement.

  To create custom style sheet, user has to toggle the button to Yes. A CSS Editor button will appear on the right of the toggle button.

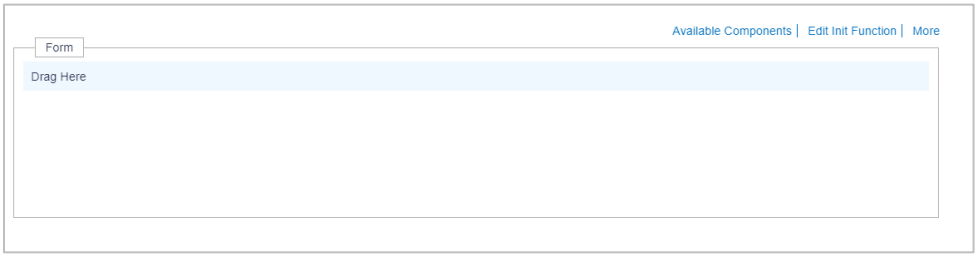


On this button click, an editor window pops up. This code editor supports Sass(Syntactically awesome style sheets). User can provide the custom classes in this editor.



After component creation, Sass file and processed Sass i.e. CSS file will be generated that will contain user defined style definition written in sass editor.

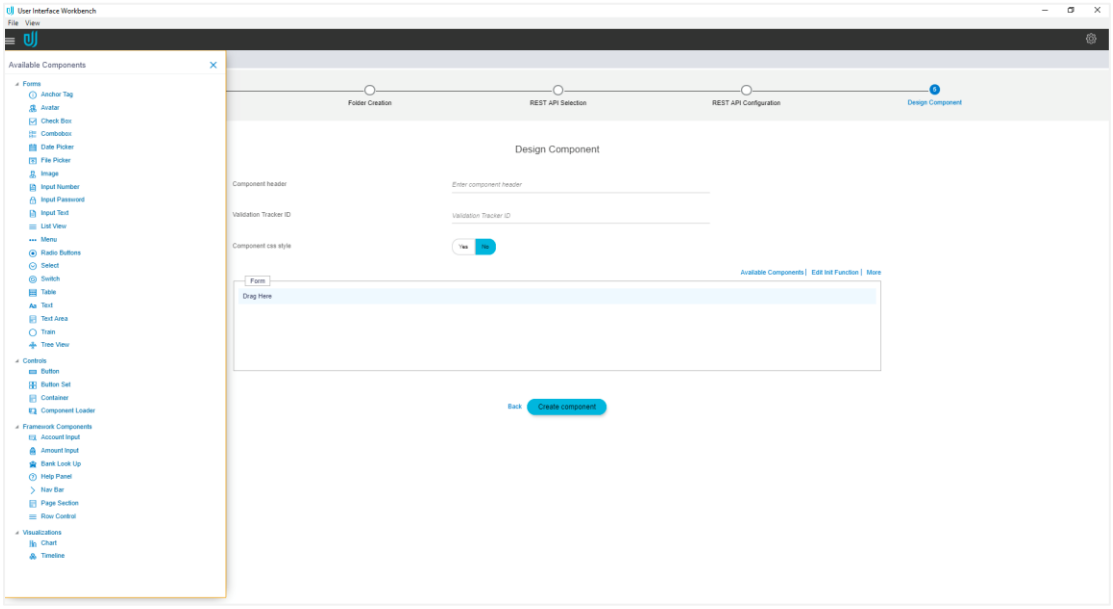
The next section can be considered as a canvas where user can design the screen contents.



The links in the top right section in the above image are explained below:

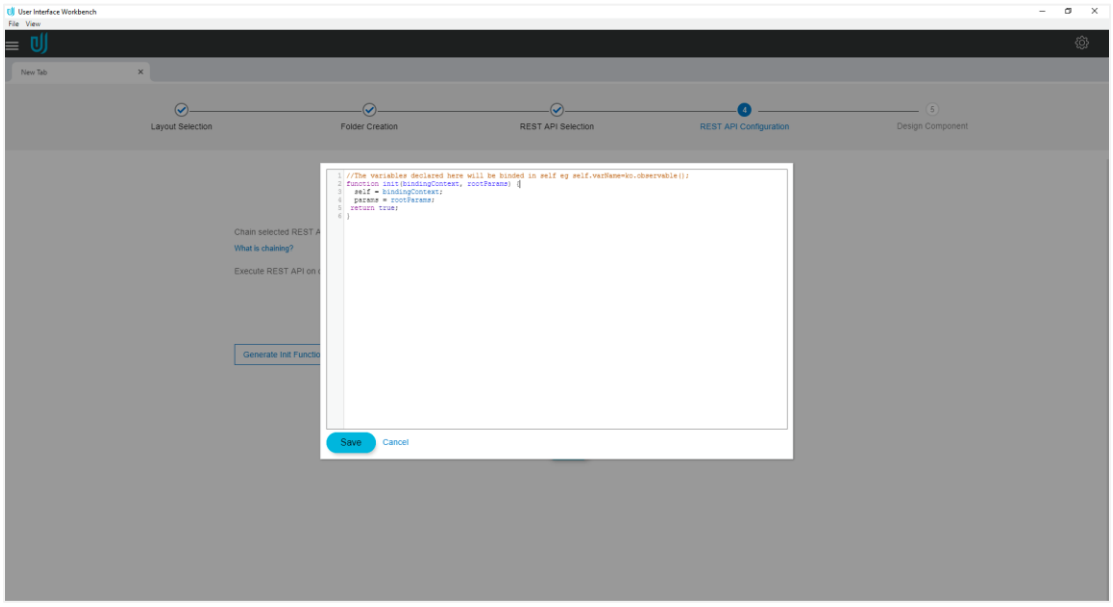
- **Available Components**

  On clicking this link, a panel opens on the left side, which contains the UI elements supported by the tool.



- **Edit Init Function**

  This link opens the editor for writing 'init function', which you generated in previous step (REST API Configuration). After making changes to init function, click save to apply the changes. Clicking on cancel button will simply discard the changes.

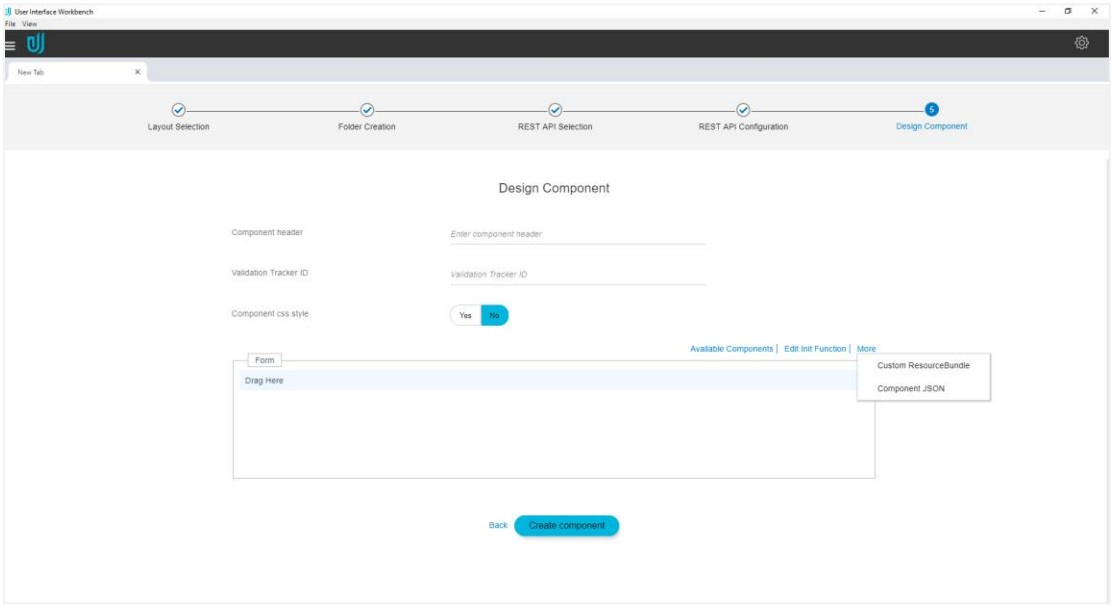


- **More**

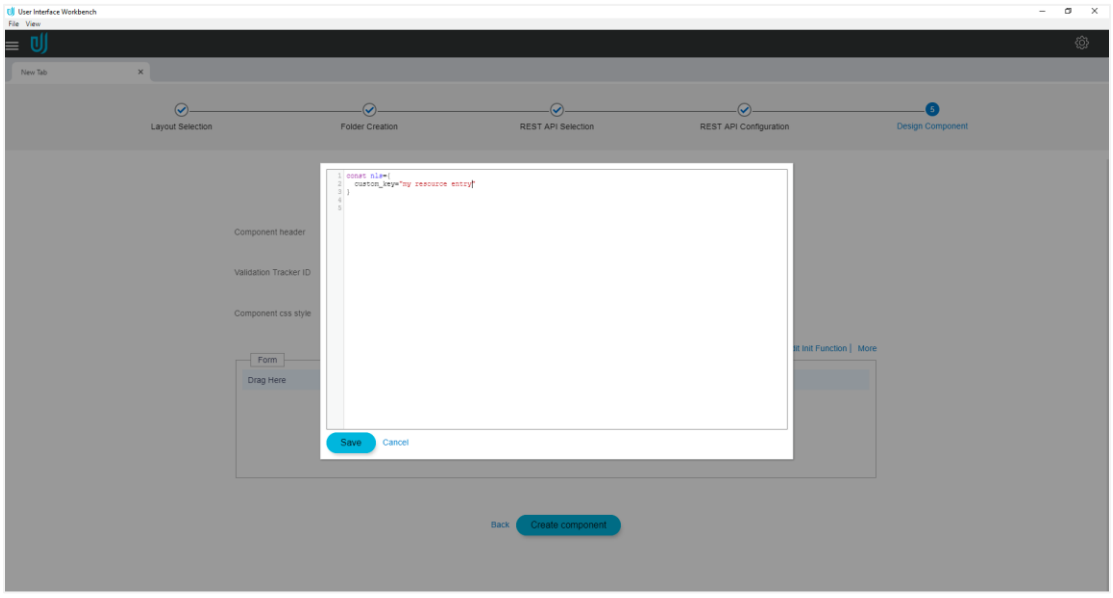
  On clicking more, context menu is opened that has following options:

  - **Custom Resource Bundle**

    To add custom entry to the resource bundle file (which contains all the text that will be displayed on the screen), user can select this option. After selecting this option, editor window will pop up where user can add the custom entry.



To add new entry, add key and the resource value inside the braces of 'const nls' , as shown in the picture below:

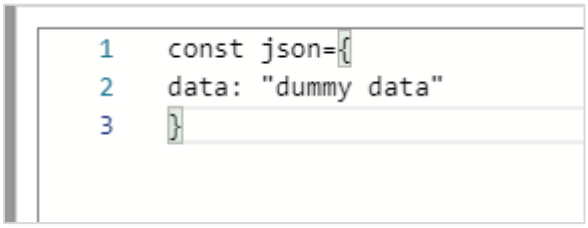


This resource bundle value can be used in two way in the component

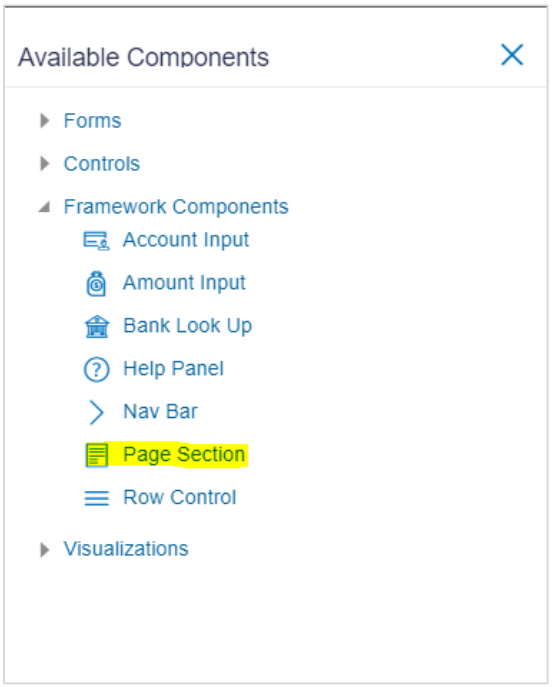
1. Inside init function: Here resource bundle value can be accessed with code *self.nls.custom_key*.
2. Inside html: In normal scenario, tool creates resource bundle for every string attached to html element. But in special case, this can be simply done by writing $component.nls.custom_key.

- **Component JSON**

  Selecting this option will open editor window. On creation of component, this JSON file is created inside the component.

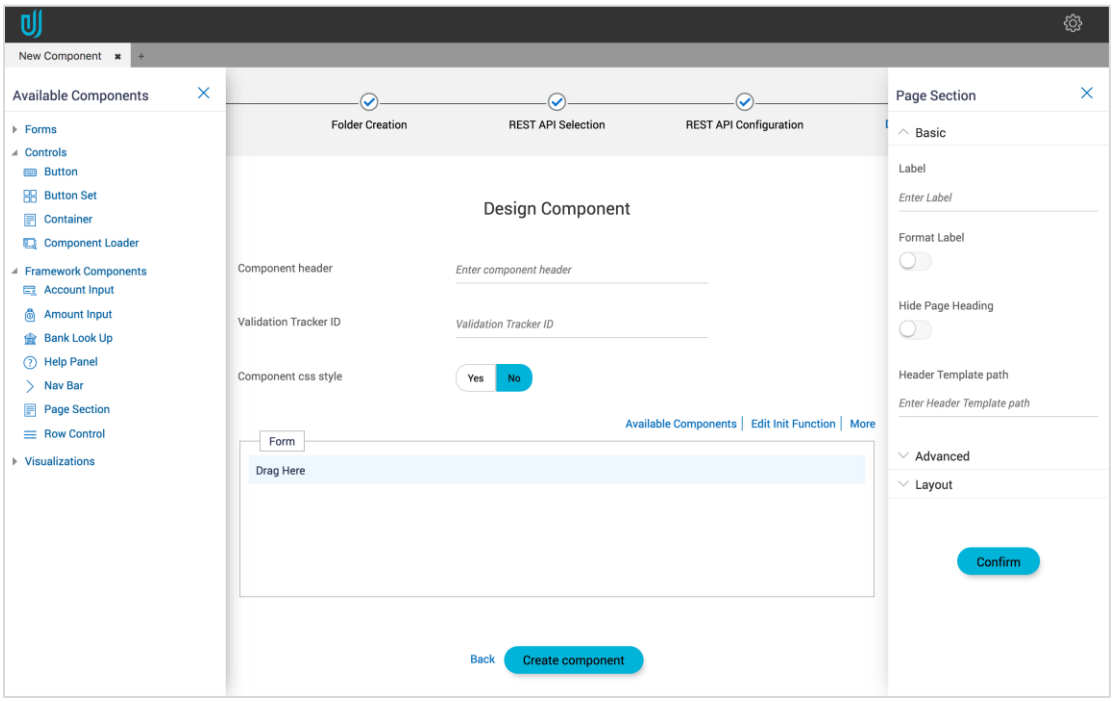


To create a screen, first element has to be ***Page Section*** *(refer Page section doc)*, which can be found in framework Components section of left panel.



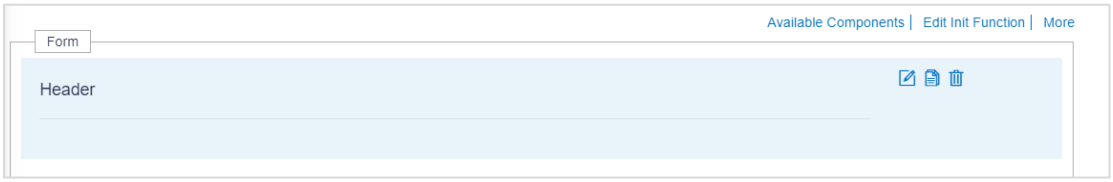
To add any element from left panel to form area, either drop it on a particular location or just click on it to add it as a last element in form area.

After clicking on Page Section option panel on the right will be opened where details for this element are captured.

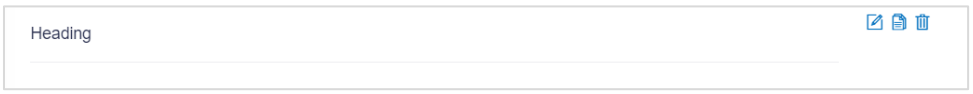


After filling compulsory fields, click confirm to add element. In case all the required fields are not filled, error message will be thrown.

For Page Section, only Label element is compulsory. After confirming, Page Section is added to the form area.



To make changes to the element, click on edit icon or to remove the element click on delete icon, or to copy element just below this click on copy icon.

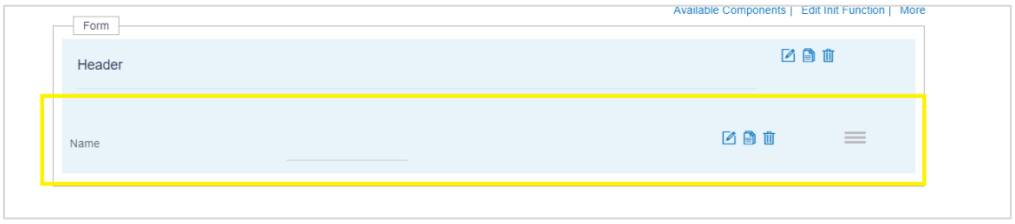


User can add as many Page Sections as needed in the screen and add form elements to it.

Adding form element to page-section

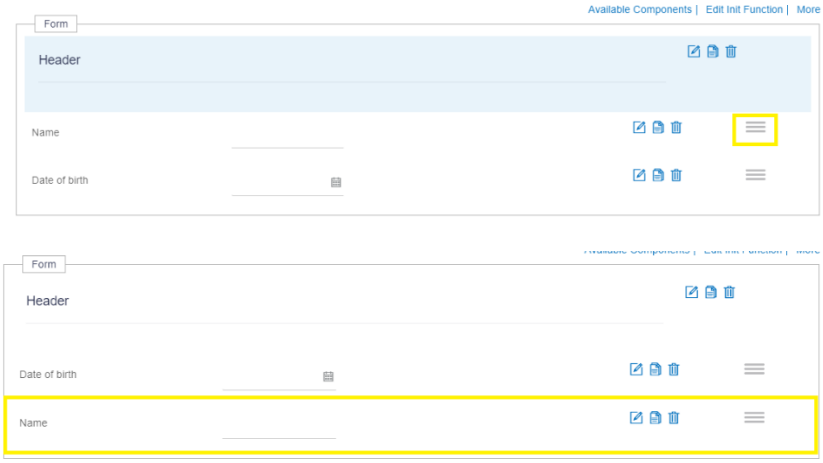
After adding Page Section, any element, which are present in available element panel can be added or dropped below it. For example, to add input box click, on available components ->Expand Forms section ->click on Input Text element

Fill in the required details in the right panel for the element and click confirm. Input element will be added to the form layout below Page Section.



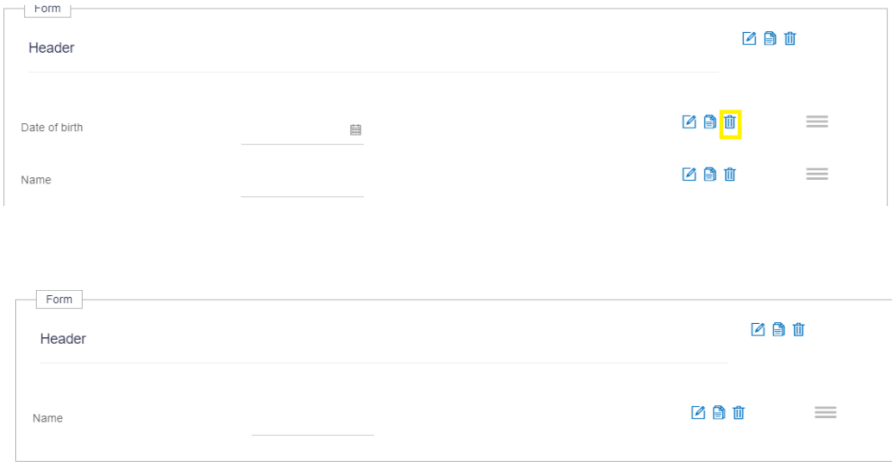
Reordering a form element

Added elements inside form area can be re-ordered among themselves. To re-order an element click and hold the right most drag icon and release at desired position.



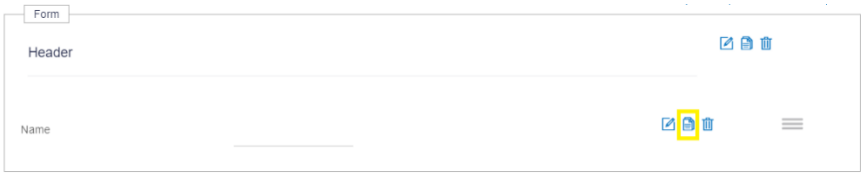
Deleting a form element

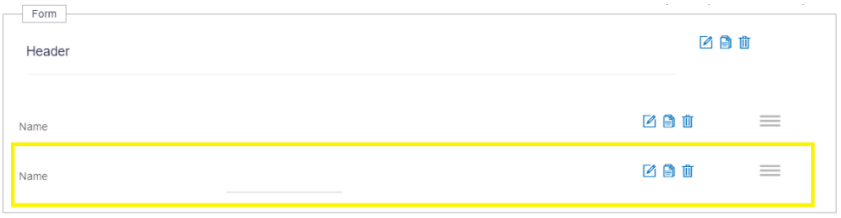
To delete form element click on delete icon.



Copying a form element

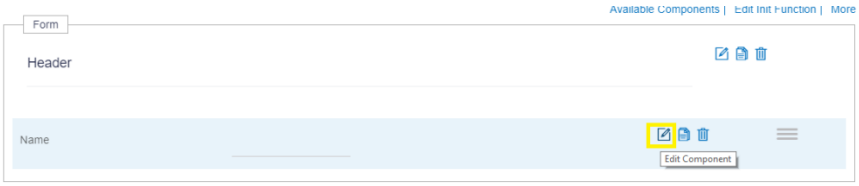
To copy an already present form element in form area, click the copy icon, new copy of the selected element will added below it, which can again be re-ordered if required.

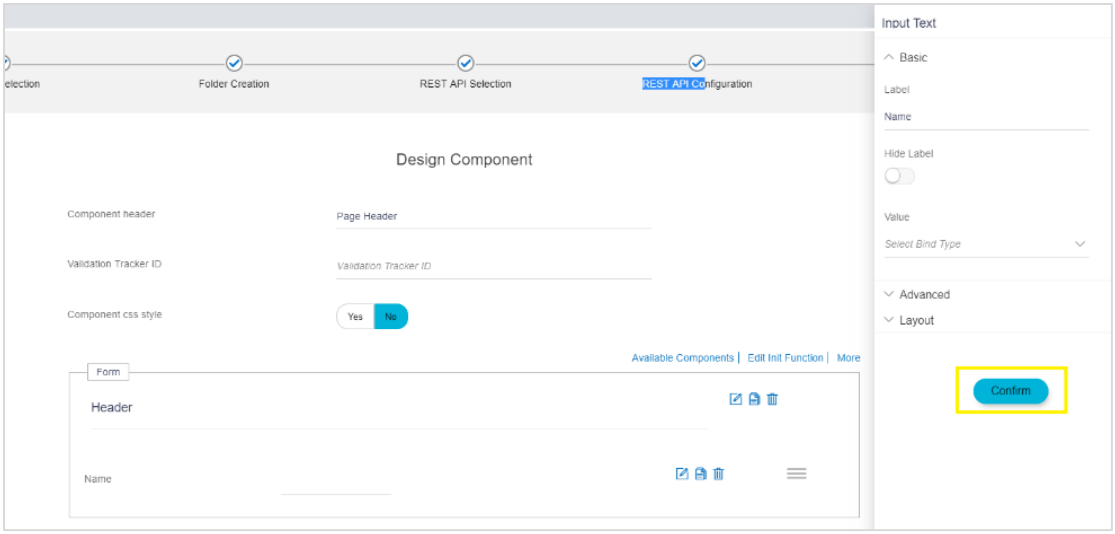




Editing a form element

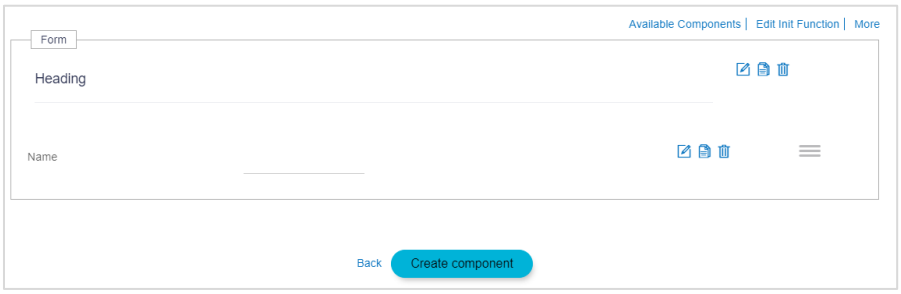
To edit an already present form element in form area click the edit icon, a right panel will open where the details can be updated. After editing, click on confirm to save the changes.



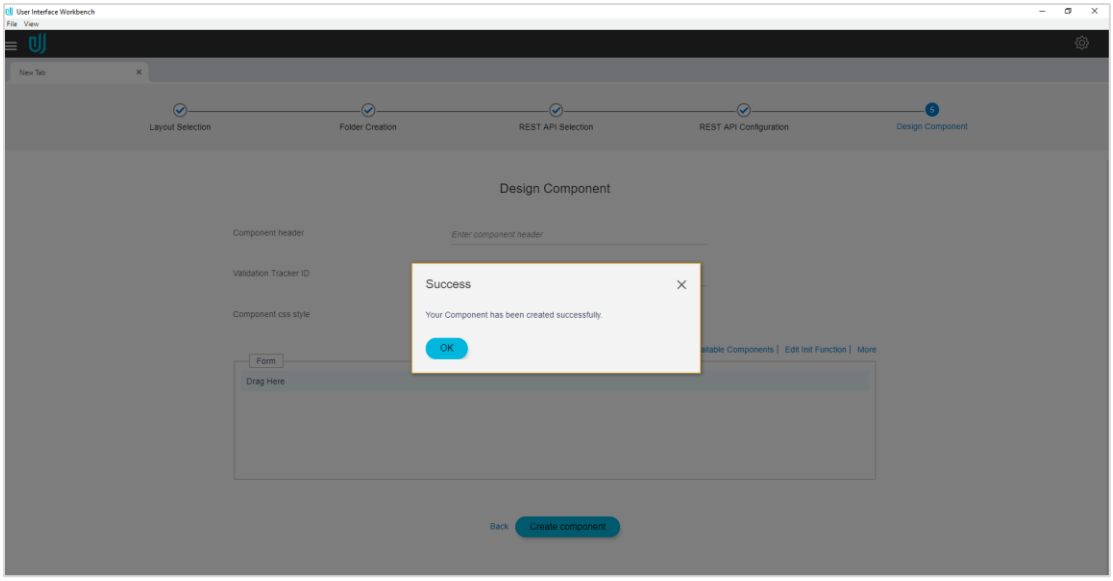


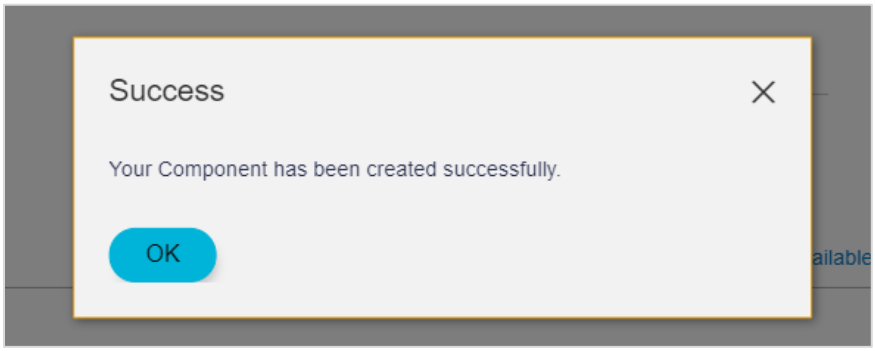
Creating Component

Once the user is satisfied with the screen design, user can click on the create component button to generate the component.



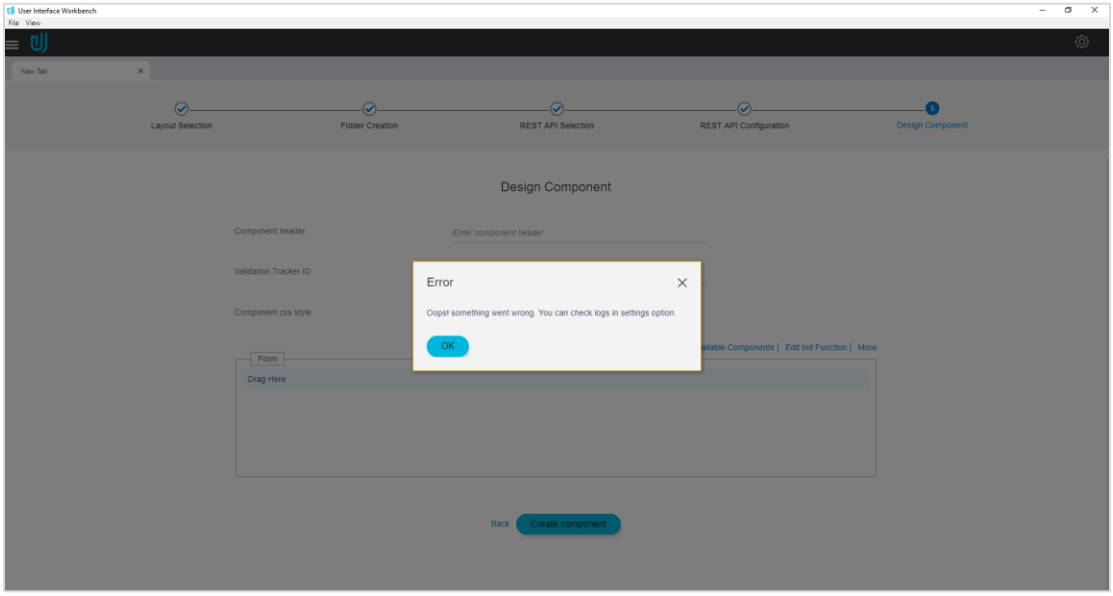
If all the details filled for each form element are valid then component will be generated and a success message will be displayed in a dialog box stating that the components have been created successfully.

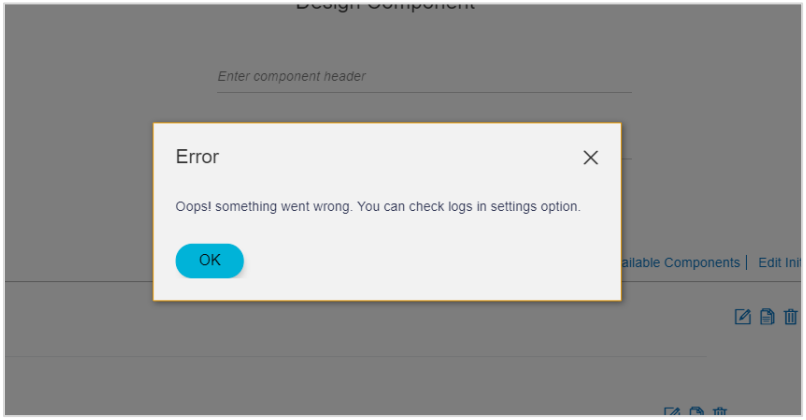




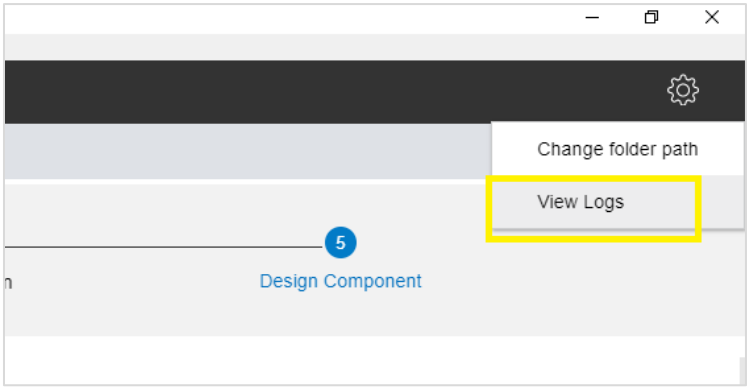
Click the OK button or cancel icon to close the pop up. User can make further changes to it and then update the created component by clicking on ‘Create Component’ button.

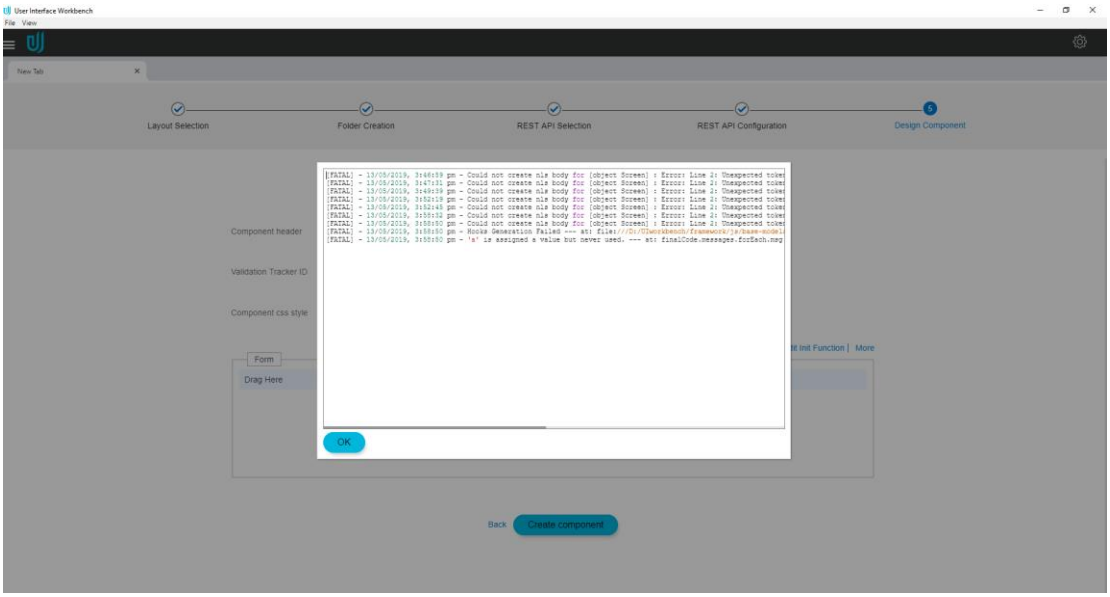
In case there is an error in case of component generation, error message will be displayed in a dialog box.





To find out what errors caused the screen creation failure, click on View Logs option which can be found in the setting icon on top right corner of the tool in the dark grey panel.





User can fix the errors and can click on Create Component again. If there are no further errors, component will be generated and success window will be displayed.

Generated Artefacts

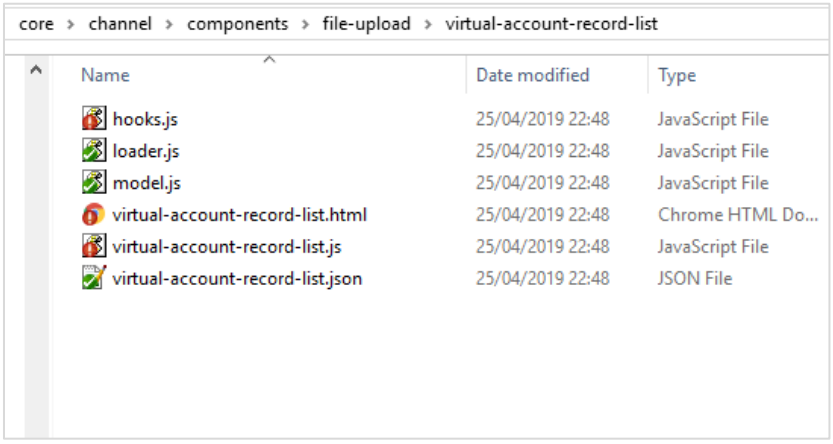
After Creating components following files are generated inside channel path selected by the user;

a. Inside channel_path/component/module_name/component_name:

1. Hooks.js – It contains all the custom code written by user in the Init function or Hook function.
2. Loader.js – This is the entry file to load the component.
3. Model.js – This file is responsible for communicating with server and transferring data between component and server.
4. Component_name.js- This file is the view model for the component.
5. Component_name.html – It contains the html code for the designed layout.
6. Component_name.json – Code written in component JSON window is added in this file
7. Component_name.scss - If Component SCSS style was turned on, a SCSS file is created which contains the code written in SCSS Editor.
8. Component_name.css - If Component CSS style was turned on, the same file is created which contains code written in CSS Editor.

e.g. Below screenshot refers to a virtual-account-record-list component which is generated using the tool.

Note: Generated artefacts must not be overwritten manually.



b. Inside channel_path/resources/nls:

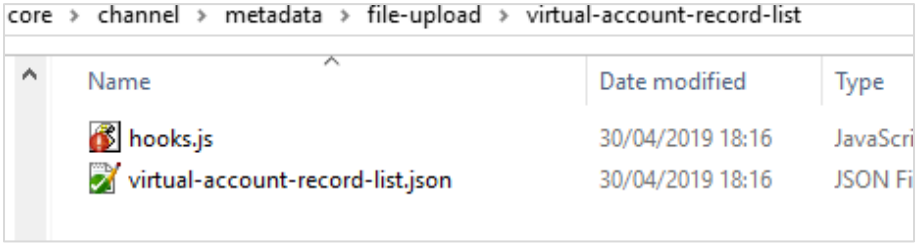
1. component_name.js – It contains the language specific values entered by user.

c. Inside channel_path/metadata/module_name/component_name:

1. component_name.json- This is the manifest file for the component.

2. hooks.js- It contains all the custom code written by user in Init function or Hook function.

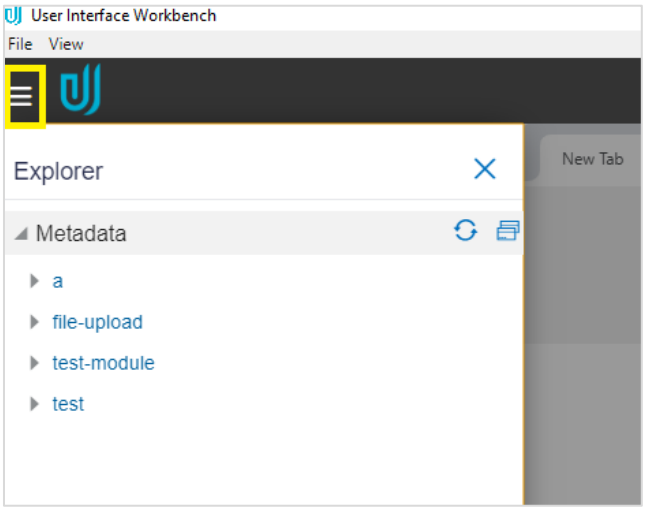
E.g. metadata files for Virtual-Account-Record-List component



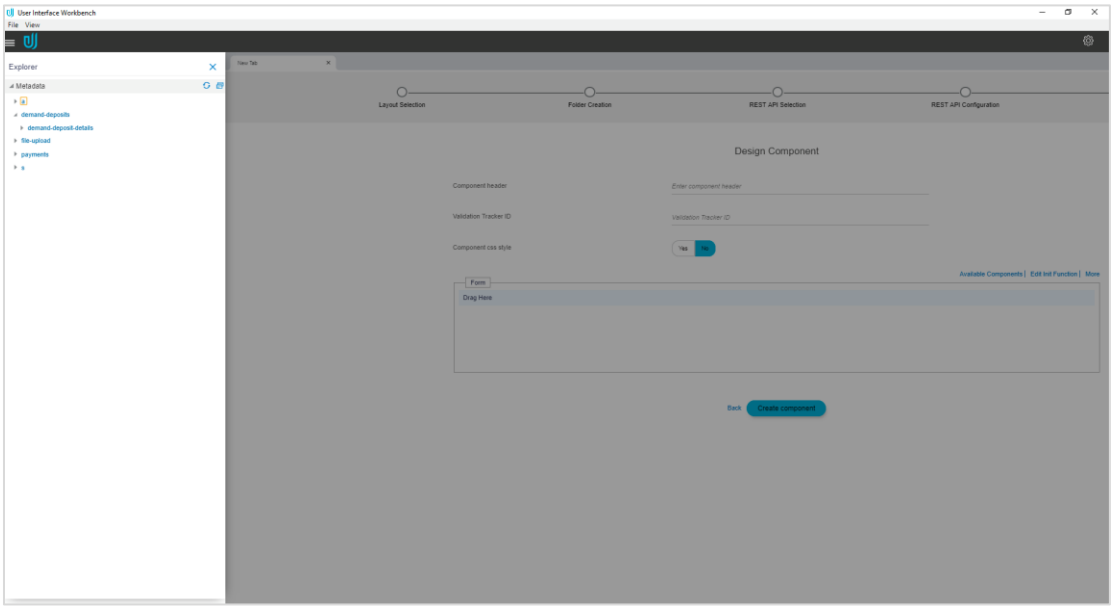
Editing Component

All the components generated using the tool can be edited by opening the component in the tool.

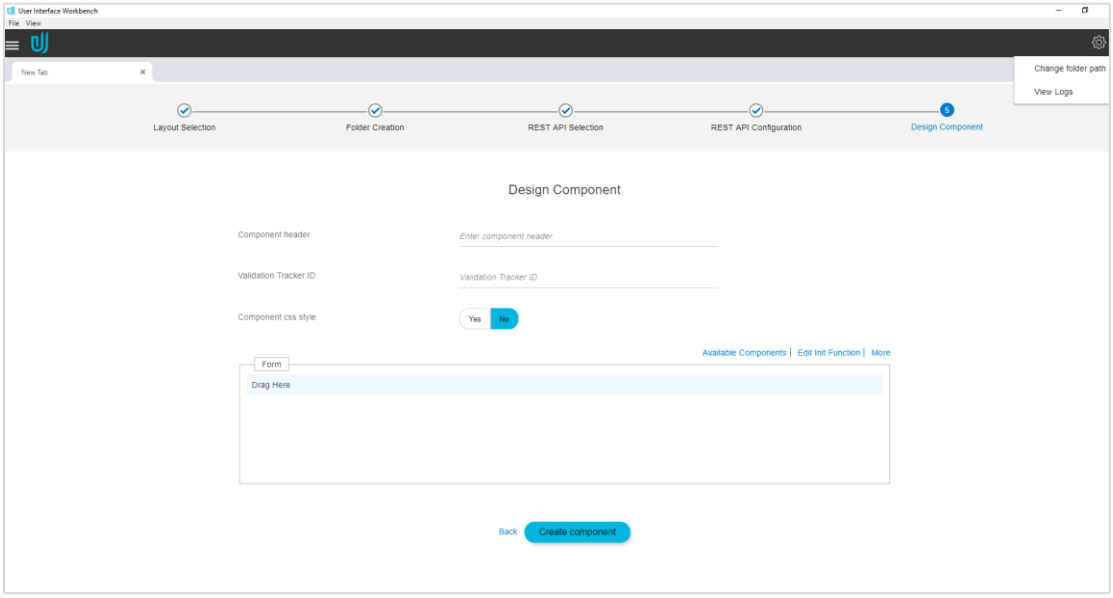
To edit a component, click on File Explorer icon, available at top left corner of the tool in the grey panel.



All the created components are present under Metadata accordion. To start editing a component, expand the module_name->component-> click on the component name



Once the user clicks on the screen name, he will be directed to the screen created during the first time.



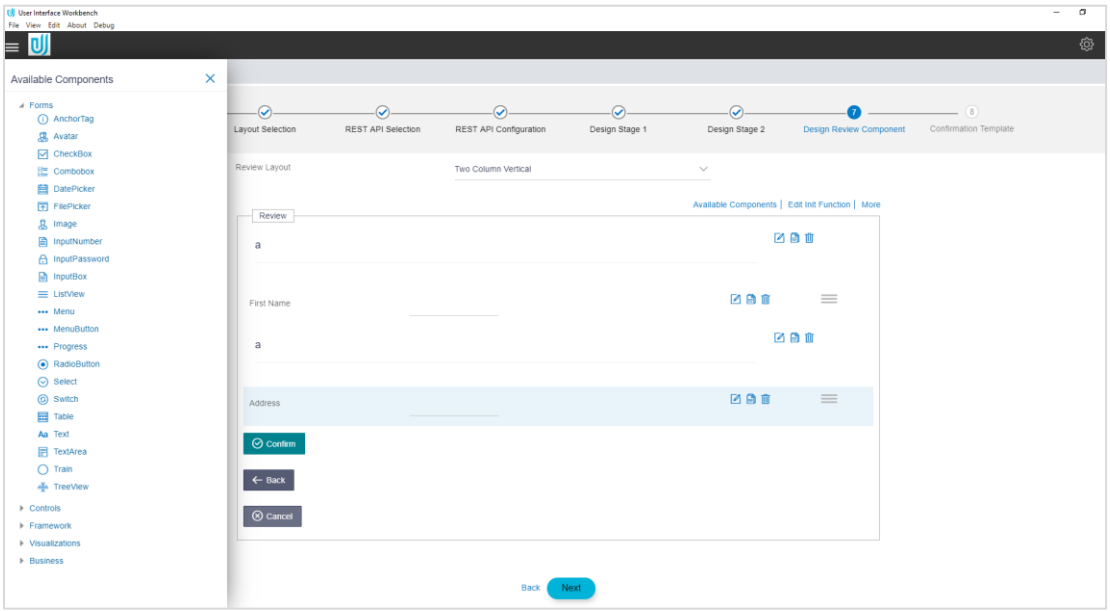
The user can now edit the screen as per requirement and save the changes by clicking on Create Component Button.

Design Review Component

If the type of component is FLOW, this step is automatically added. It is similar to design component stage.

In this Stage, a default design is created using the fields which have been designed on the design stages.

Thus a review page can be generated without user intervention.



In case the design does not meet the requirements, user can make changes in the design as needed. Once the user manipulates the screen, the tool stops automatic design generation. For e.g. If user adds a field in one of the design pages, then he will have to add that field on the review design step also.

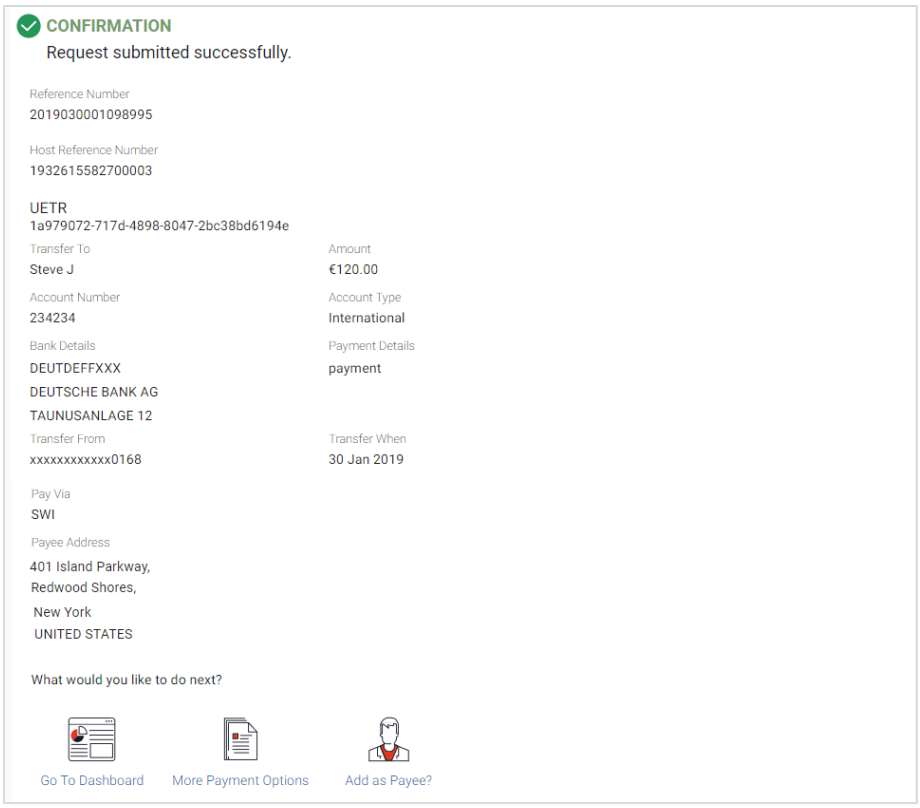
Confirmation Screen:

Confirmation screen is used to show confirmation of the operation, only in case of flows. User can design his own confirmation screen. Confirmation template option will be available when Component Type is Flow & Flow Type is Create.

There will be 2 options available for Confirmation Template:

1. **Standard**

   It is a standard template of confirmation screen, which means user cannot modify or add any component in this screen. Screen will look like image below.



2. **Animated Page**

   In this case user can design his own confirmation screen.

   One extra step will get added in train after review screen, this will be the design step for confirmation screen.

   Toolkit will generate a page with an animated imaged centrally placed by default.

User can delete or change this image. He can also add any text of his choice e.g. Thank You or Transaction Confirmed or can add other components e.g. Buttons, Icons etc.

Confirm folder will get generated in NLS, metadata & in created component folder also.

10. Available Components

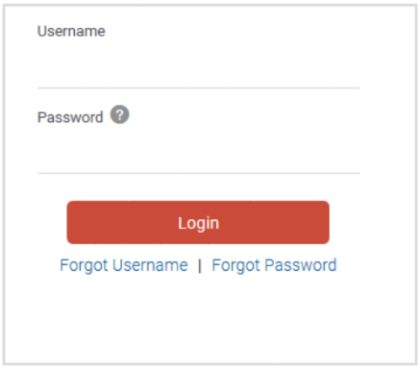
There are four types of components available in this panel.

A) Forms
B) Controls
C) Framework Components
D) Visualization

A) Forms

This section contains the components which are required in designing a form.

For example, a Login form accepts user name and password and validates a user. In the example below, developer can use Input Text and Input Password element for user name and password respectively.



Following is the list of available components under this section.

1. Anchor Tag
2. Avatar
3. Check Box
4. Combo Box
5. Date Picker
6. File Picker
7. Image
8. Input Number
9. Input Password
10. Input Text
11. List View
12. Menu
13. Radio Buttons
14. Select

15. Switch
16. Table
17. Text
18. Text Area
19. Train
20. Tree View

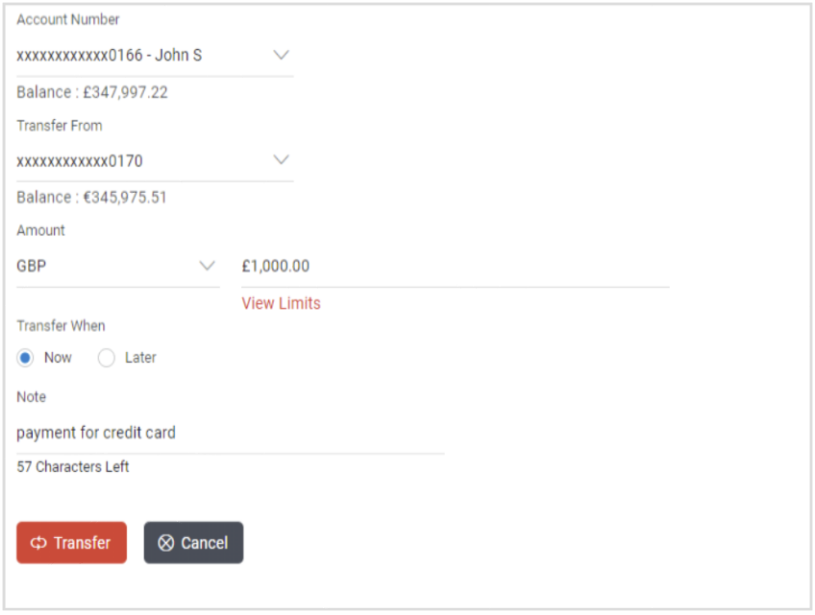
B) Controls

This section contains the components, which are used to control the activity on the form.

For example, user has a form to transfer money from one account to another account in which he will have the following two actions:

- Transfer button to confirm the transaction
- Cancel button to cancel the transaction

In the below example, buttons are displayed:



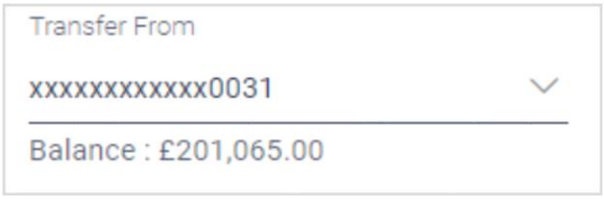
Following is the list of available components under this section.

1. Button
2. Button Set
3. Container
4. Component Loader

C) Framework Components

This section contains the components, which are predefined & supported by OBDX. It has in-built functionality.

In the below example, an account input component which lists all the accounts of the user and shows its details like balance and address of the account selected by the user .



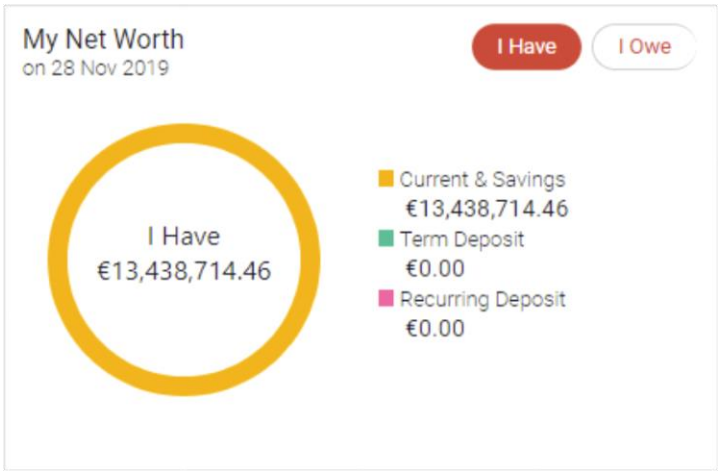
Following is the list of available components under this section.

1. Account Input
2. Amount Input
3. Bank Look Up
4. Help Panel
5. Navigation Bar
6. Page Section
7. Row Control

D) Visualization

This section contains the components, which are useful to design a widget that display s information graphically.

In the below example, a chart that shows the net worth of different accounts (CASA) of the user



Following is the list of available components under this section.

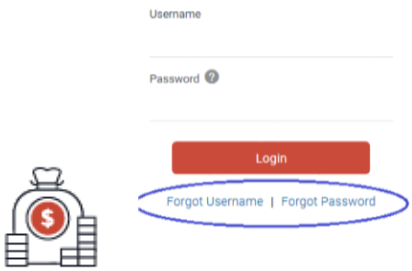
1. Chart
2. TimeLine

Details of components

A) Forms:

1) Anchor Tag :

Usage: This component is used to add a link. This link can be an image / icon or text.
Example:



Supported attributes:

a) Basic:

- Label
- Format Label

b) Advanced:

- Select anchor type
- Add formatter
- REST API chain
- Hook functions
- Add Loop
- Add Custom Attributes
- Conditional Field

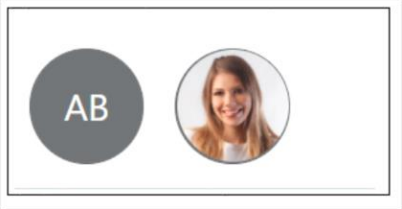
c) Layout:

- Grid

2) Avatar

Usage: This component is used to display initials for name or product and images in a circle.

Example:



Supported attributes:

a) Basic:

- Label

b) Advanced:

- Select Size
- Enter Image Path
- Enter Initials
- Add Loop
- Add Custom Attributes:
- Conditional Field

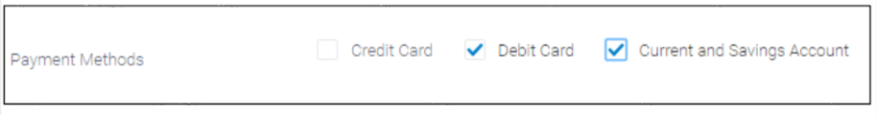
c) Layout:

- Grid

3) Check Box:

Usage: This component is used to add checkbox to allow a user select one or more options from the available choices.

Example:



Supported attributes:

a) Basic:

- Label
- Hide Label

- Value
- Options

b) Advanced:

- Value Change Handler
- Validations
- Required Field
- Add Loop
- Add Custom Attributes
- Conditional Field

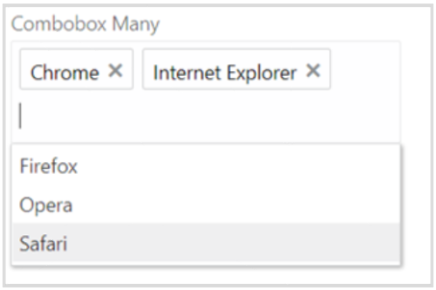
c) Layout:

- Grid

4) Combo Box:

Usage: This component is used to add a drop down to allow user to make multiple selection from the predefined options or type his own option.

Example:



Supported attributes:

a) Basic:

I) Label:

- Hide Label
- Value
- Select Type
- Options

b) Advanced

- Value Change Handler
- Validations
- Required Field
- Add Loop
- Add Custom Attributes
- Conditional Field

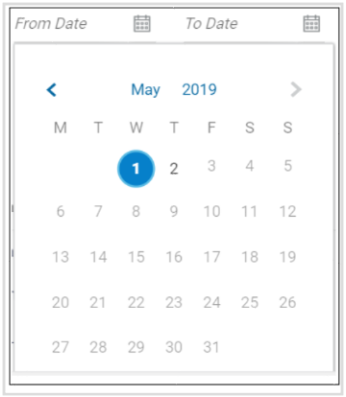
c) Layout:

- Grid

5) Date Picker:

Usage: This component is used to add an input box to accept date as an input from user.

Example:



Supported attributes:

a) Basic:

- Label
- Hide Label
- Value

b) Advanced:

- Value Change Handler
- Validations

- Required Field
- Add Loop
- Add Custom Attributes
- Conditional Field

c) Layout:

- Grid

6) File Picker:

Usage: This component is used to add an action as a button to accept the documents (files) as an input from user.

Example:

Supported attributes:

a) Basic:

- Label

d) Advanced:

- Selection Mode
- Enter Allowed File Extensions
- REST API chain
- Hook functions
- Required Field
- Add Loop
- Add Custom Attributes
- Conditional Field

e) Layout:

- Grid

7) Image:

Usage: This component is used to add an image.

Example:

Supported attributes:

a) Basic:

- Image source

b) Advanced:

- Add Loop
- II)Add Custom Attributes
- III)Conditional Field

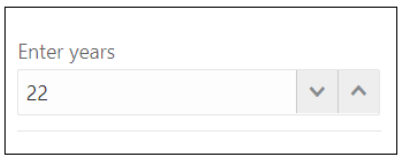
c) Layout:

- Grid

8) Input Number:

Usage: This component is used to add an input box to accept numbers as an input from the user .

Example:



Supported attributes:

a) Basic:

- Label

- Hide Label
- Value

b) Advanced:

- Enter Minimum Length
- Enter Maximum Length
- Enter Step
- Value Change Handler
- Validations
- Required Field
- Add Loop
- Add Custom Attributes
- Conditional Field

c) Layout:

- Grid

9) Input Password:

Usage: This component is used to add an input box to accept the password as an input from the user.

Example:

Supported attributes:

a) Basic:

- Label
- Hide Label
- Value

b) Advanced:

- Value Change Handler

- Validations
- Required Field
- Add Loop
- Add Custom Attributes
- Conditional Field

c) Layout:

- Grid

10) Input Text:

Usage: This component is used to add an input box to accept text as an input from the user

Example:

Supported attributes:

- Basic
- Label
- Hide Label
- Value

a) Advanced:

- Value Change Handler
- Validations
- Required Field
- Add Loop
- Add Custom Attributes
- Conditional Field

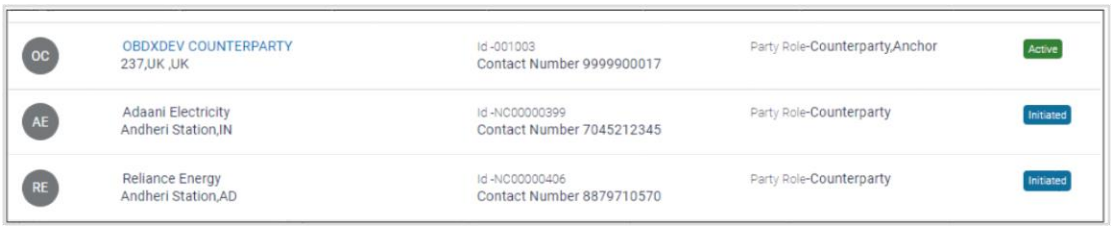
b) Layout:

- Grid

11) List View:

Usage: This component is used to add a view which groups several items and display s them in a vertical scrollable list

Example:



Supported attributes:

a) Basic:

- Label
- Source variable
- Id attribute

b) Advanced:

- Renderer ID
- Pagination
- Indexer
- Add Loop
- Add Custom Attributes
- Conditional Field

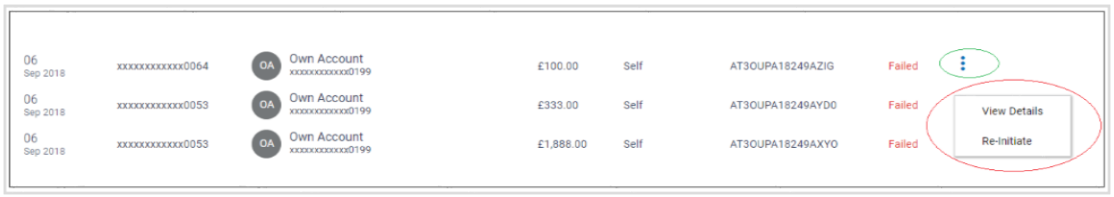
c) Layout:

- Grid

12) Menu:

Usage: This component is used to add an element, which displays popup menu with multiple options relevant to the particular row for easier navigation.

Example:



Supported attributes:

a) Basic:

- Label
- ID
- Options

b) Advanced:

- Enter Menu Launcher
- REST API chain
- Hook functions
- Add Loop
- Add Custom Attributes
- Conditional Field

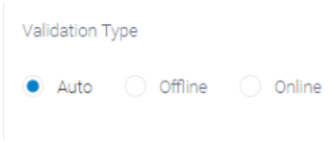
c) Layout:

- Grid

13) Radio Buttons:

Usage: This component is used to add buttons to let a user select an option from the available choices.

Example:



Supported attributes:

a) Basic:

- Label
- Hide Label
- Value:
- Options

b) Advanced:

- Value Change Handler
- Validations
- Required Field
- Add Loop
- Add Custom Attributes
- Conditional Field

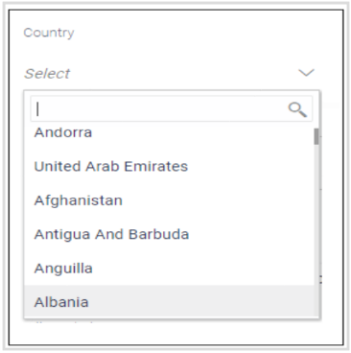
c) Layout:

- Grid:

14) Select:

Usage: This component is used to add a drop down, to allow user to make a choice from the predefined options.

Example:



Supported attributes:

a) Basic:

- Label
- Hide Label
- Value
- Select type
- Options

b) Advanced:

- Value Change Handler
- Validations
- Required Field
- Add Loop
- Add Custom Attributes
- Conditional Field

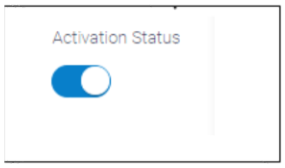
c) Layout:

- Grid

15) Switch:

Usage: This component is used to add a toggle button for binary status such as on/off.

Example:



Supported attributes:

a) Basic:

- Label
- Hide Label
- Value

b) Advanced:

- Value change handler

- Required field
- Add Loop
- Add Custom Attributes
- Conditional Field

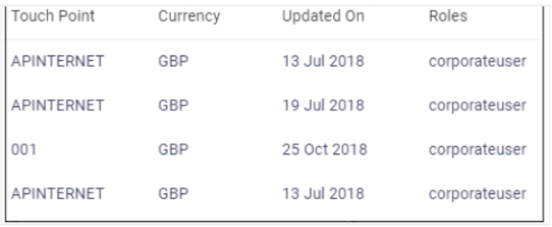
c) Layout:

- Grid

16) Table:

Usage: This component is used to add a view which groups several items and display s them in row- column fashion.

Example:



Touch Point	Currency	Updated On	Roles
APINTERNET	GBP	13 Jul 2018	corporateuser
APINTERNET	GBP	19 Jul 2018	corporateuser
001	GBP	25 Oct 2018	corporateuser
APINTERNET	GBP	13 Jul 2018	corporateuser

Supported attributes:

a) Basic:

- Label
- Source variable
- Id attribute
- Columns

b) Advanced:

- Row renderer
- Pagination
- Add Loop
- Add Custom Attributes
- Conditional Field

c) Layout:

- Grid

17) Text:

Usage: This component is used to add a simple text.

Example:

Supported attributes:

a) Basic:

b) Advanced:

- Aria label
- Tag type
- Binding Source
- Add formatter
- Add loop
- Add Custom Attributes
- Conditional Field

c) Layout:

- Grid

18) Text Area:

Usage: This component is used to add an input box to accept the multi-line text as an input from user.

Example:

Supported attributes:

a) Basic:

- Label

- Hide Label
- Value

b) Advanced:

- Enter rows
- Value Change Handler
- Validations
- Required Field
- Add Loop
- Add Custom Attributes
- Conditional Field

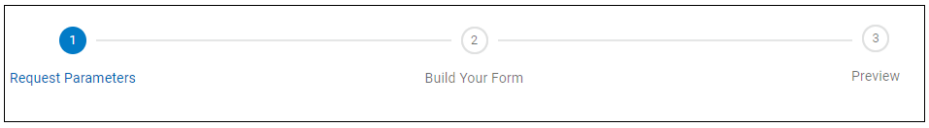
c) Layout:

- Grid

19) Train:

Usage: This component is used to add an element for navigation that allows a user to go between different "steps" i.e. different components. Each step can display information about the state of the step such as "visited", "unvisited", "disabled".

Example:



Supported attributes:

a) Basic:

- Label
- Value

b) Advanced:

- Selected step
- Value change handler
- Add Loop
- Add Custom Attributes:
- Conditional Field

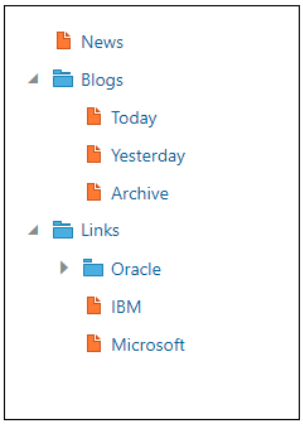
c) Layout:

l) Grid

20) Tree View:

Usage: This component is used to add an element to display the hierarchical relationship between the items of the tree.

Example:



Supported attributes:

a) Basic:

- Label
- Source variable

b) Advanced:

- Renderer ID
- Add Loop
- Add Custom Attributes
- Conditional Field

c) Layout:

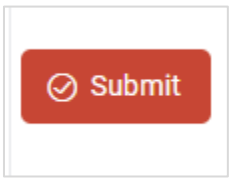
- Grid

B) Controls:

1) **Button:**

Usage: This component is used to add a call to action to the page which can be configured to perform various actions on its click.

Example:



Supported Attributes:

a) Basic:

- label

b) Advanced:

- Icons
- Select Class Type
- Select Type
- Rest API Chain
- Hook Function
- Add Loop
- Add Custom Attributes
- Conditional Field

c) Layout:

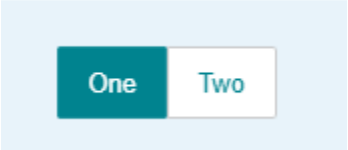
- Grid

2) Button Set:

Usage: Basically button set is a group of button which can be used as radio button or checkbox button. In case of single selection it act as radio button and in case of multiple selection it act as checkbox set button

Example:

a) Radio button set



b) Checkbox button set

Supported Attributes:

a) Basic:

- ID: It accepts the same value as Label
- Value
- Select Type:
  1) Buttons Set One: Choose this to use it as radio button set
  2) Button Set Many: Choose this to use it a checkbox button set
- Options

b) Advanced:

- Value Change Handler
- Add Loop
- Add Custom Attributes
- Conditional Field

c) Layout:

- Grid

3) Container:

Usage: This component is used to provide a wrapper or group for form elements to apply a common behavior to them. e.g. display / hide of buttons. An element is always placed inside the container start element and container end element.

1. On Dropping the container element at the desired position on form area, a panel will open on the right side
2. User can select the type, fill the required fields and click confirm.

Note: Container can also be used inside container.

Supported Attributes:

a) Basic:

- Select Type Of Container

b) Advanced:

- Add Loop
- Add Custom Attributes
- Conditional Field

c) Layout:

- Grid

4) Component Loader:

Usage: Component Loader is used to load already created component or partial in current component.

Supported Attributes:

a) Advanced:

- Select Type:

  a) Component

     1) Select Type

        i) Framework Elements: To use Framework Components.

        Refer to framework components section for more detail.

        ii) Transaction: To load transaction type of component. Refer to Transaction component section.

        III) Single: To load normal component.

  b) Partial:

- Add Loop:
- Add Custom Attributes:
- Conditional Field:

b) Layout:

- Grid:

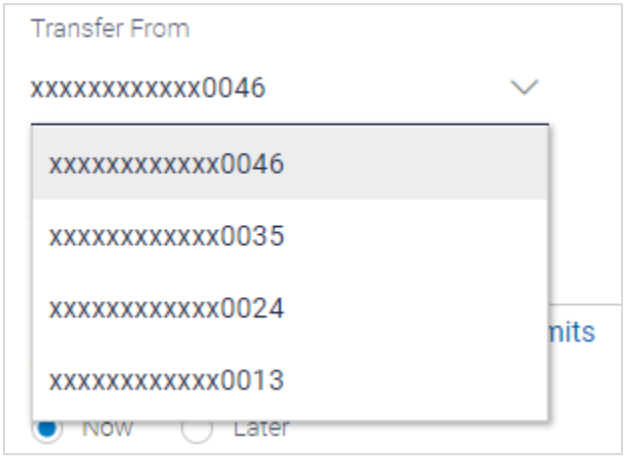
C) Framework Component:

These components are predefined component that can be plugged with the user's created component to give specific functionality. Different framework component:

1) Account Input:

Usage: Component which provides account selection that is fetched from current logged in user or custom URL

Example:



Supported Attributes:

a) Basic:

- Label
- Value

b) Advanced:

- Enter Type: These values will passed to 'type' key in params for account input.
    1. balance
    2. address
    3. nodeValue
    4. loans
        - Add Loop:
        - Add Custom Attributes:
        - Conditional Field:

c) Layout:

- Grid

2) Amount Input:

Usage: This component is used to accept amount in an input box. The component gives currency selection and entered amount is formatted with respect to selected currency.

Example:

Supported Attributes:

a) Basic:

- Label
- Value

b) Advanced:

- Enter Currency Variable: To store selected currency value.
- Required
- Currency List Required: Whether to display currency list. In case currency list is required, user has to provide URL to fetch currency list or provide a function called Currency Parser which returns the currency list.
- Currency URL: URL for Fetching currency.
- Currency Parser: Custom function which returns currency list that can be declared in init function
- Enter Root ID: Root id which will be passed in params of amount input.
- Enter Root Class: Root class which will be passed in params of amount input.
- Add Loop
- Add Custom Attributes
- Conditional Field

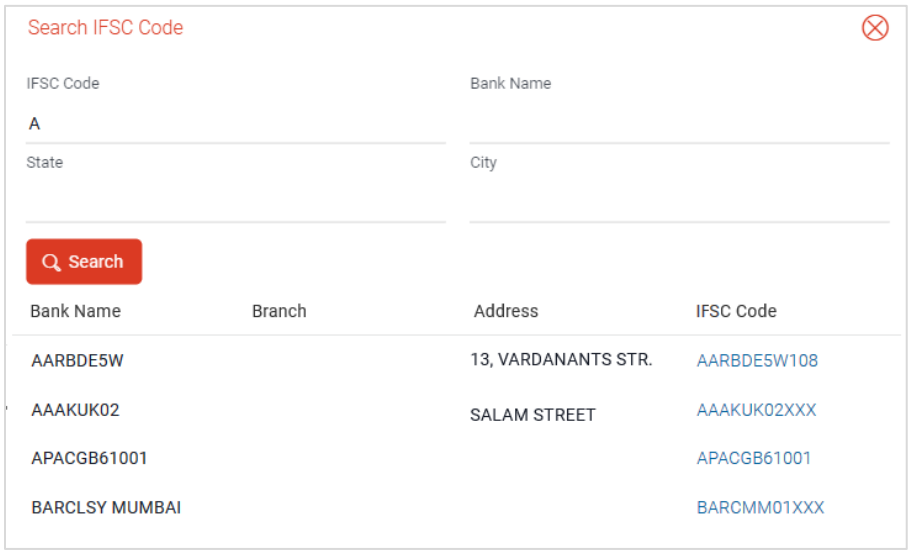
c) Layout:

- Grid

3) Bank Look up:

Usage: This component is used to provide bank look up on the basis of IFSC Code, state, city or branch name.

Example



Supported Attributes:

a) Basic:

- Label

b) Advanced:

- Clearing Code Type
- Account Type
- Region
- Network Code
- Add Loop
- Conditional Field

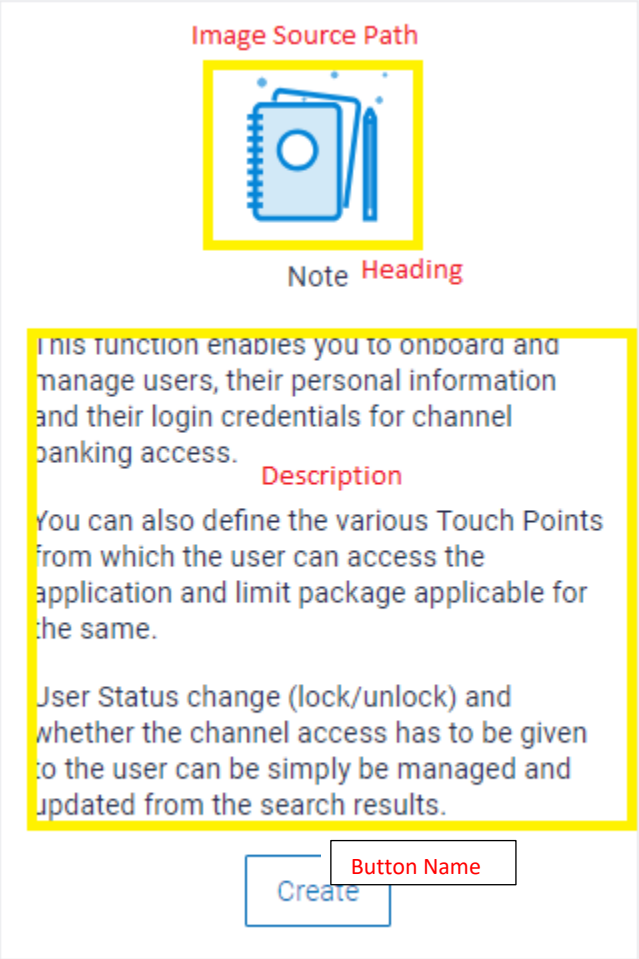
c) Layout:

- Grid

4) Help Panel:

Usage: This component is used to display related information about the current component .

Example:



Supported Attributes:

a) Basic:

- Partial Name: Name of the partial which will be created inside ChannelPath/partials/help/

b) Advanced:

- Enter Heading
- Image Source: Path of image inside channelPath/images/

- Enter Description
- Enter Button Name
- Rest API Chain: Rest to be fired on button click
- Hook Function: Write action to be performed on button click
- Add Loop
- Conditional Field

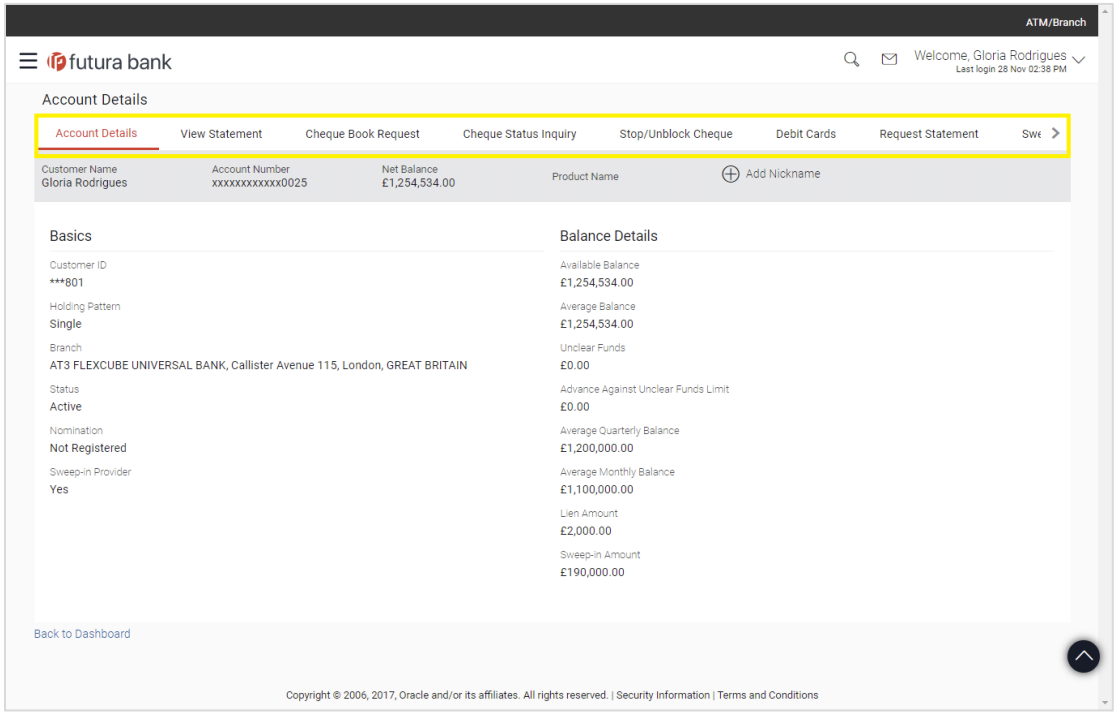
c) Layout:

- Grid

5) Navigation Bar:

Usage: This component is used to provide tab based page Navigation, where each tab item / name will display specific view.

Example:



Supported Attributes:

a) Basic:

- Label:
- Tabs: Can be provided externally or custom just like select box option. In case user is not familiar with how to provide option, refer Option section in Available attributes.

b) Advanced:

- UI Options:

  a) Icon Available

  b) Default option

  c) Menu Float

  d) Full Width

  E) Edge

- Navigation Bar Description
- Add Loop
- VI) Conditional Field

c) Layout:

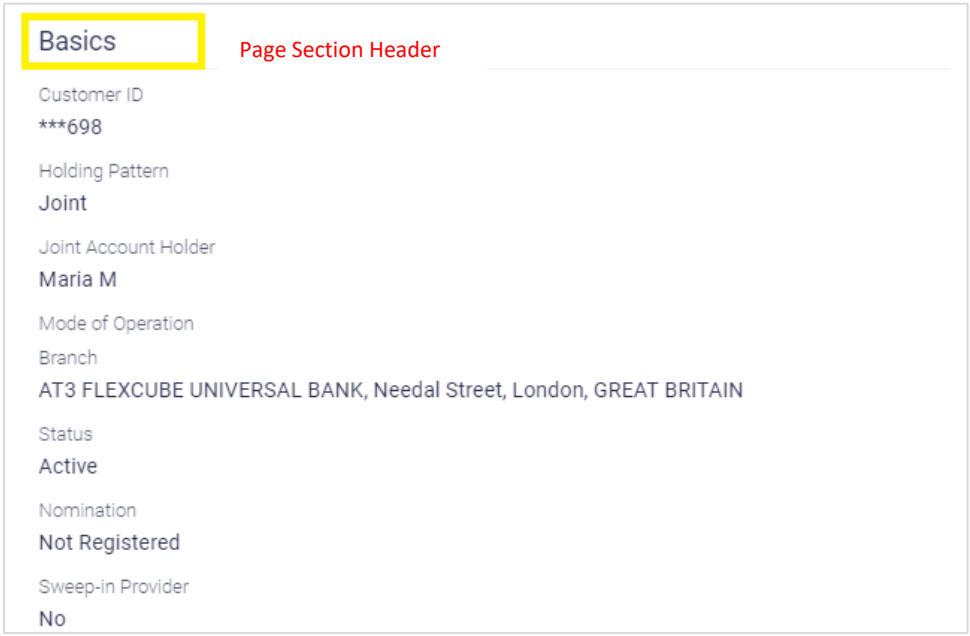
- Grid

6) Page Section:

Usage: This component is used to provide basic layout structure and styling for every page / screen.

Note: All the form elements must be present inside this container.

Example: In the example below, Page Section is added and header is optional. The other form elements can also be added via Partial.



Supported Attributes:

a) Basic:

- Label
- Format label
- Hide Page Heading: Switch to active mode to hide the page section heading.

b) Advanced:

- Add Loop
- Conditional Field

c) Layout:

- Grid

7) Row Control:

This component is used to display label and its corresponding value in structure supported by OBDX framework

Example:

Supported Attributes:

a) Basic:

- Label
- Format label
- Value

b) Advanced:

- Add Formatter
- Add Loop
- Conditional Field

c) Layout:

- Grid

D) Visualizations

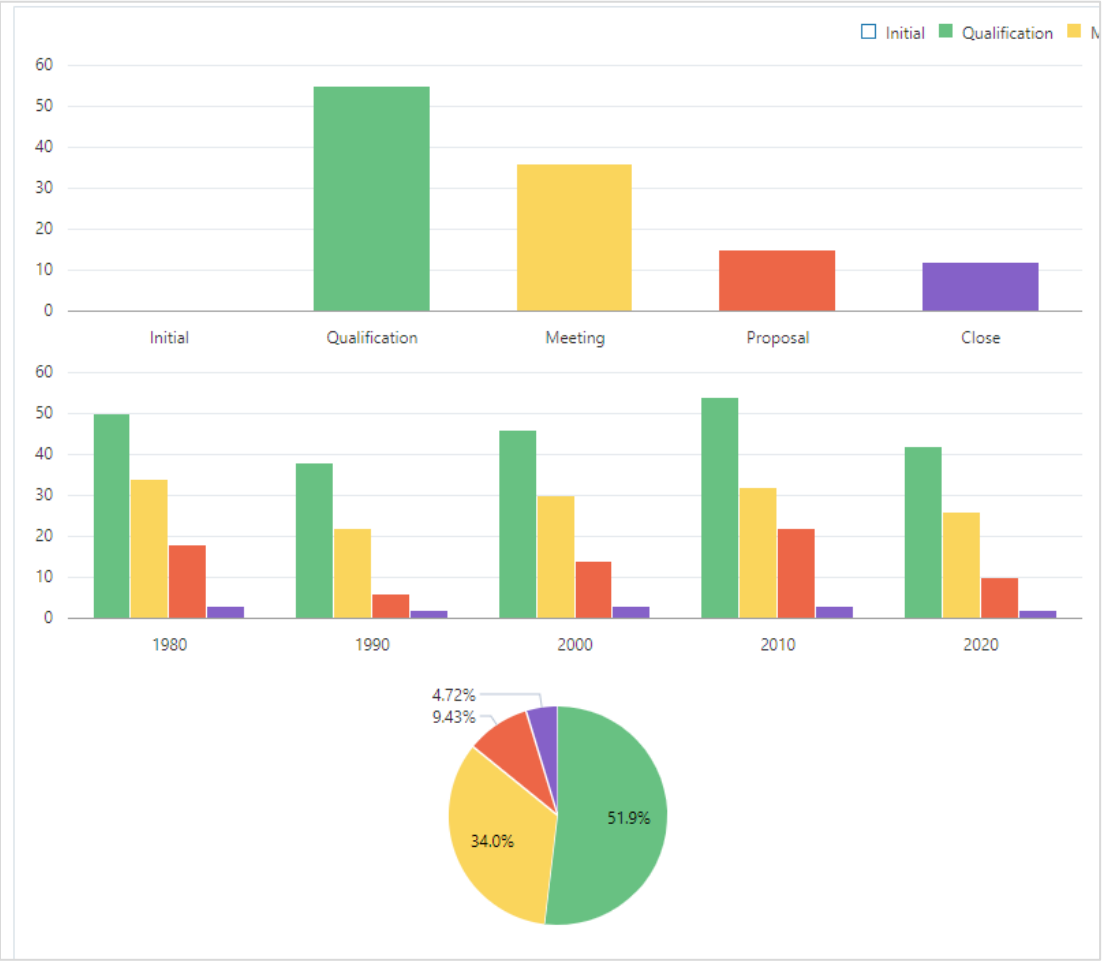
1) Chart: There are three types of chart that can be drawn with the help of tool:

a) Pie Chart,

b) Bar Chart,

c) Line Chart. For more information about each chart type, visit jet.us.oracle.com.

Example:



Supported Attributes:

a) Advanced:

- Series
- Groups
- Add loop
- Add Custom Attributes
- Conditional Field

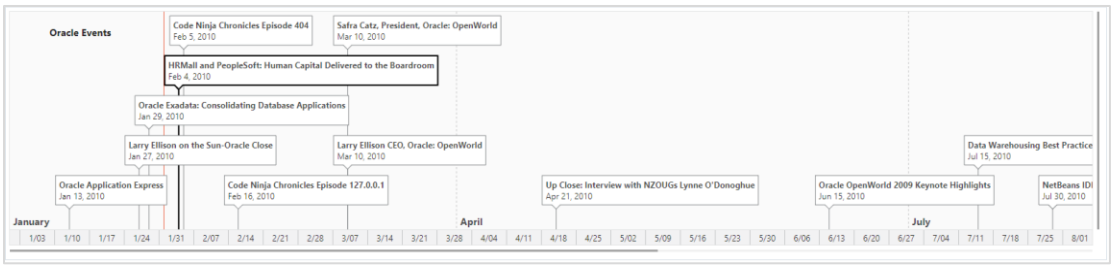
b) Layout:

- Grid

2) Timeline:

Usage: This component is used to display the timeline component. To provide values to timeline component use Add Custom Attribute (refer section **Add Custom attribute in Available Attributes**).

Example:



Supported Attributes:

a) Basic:

- Label

b) Advanced:

- Add Loop
- Add Custom Attributes
- Conditional Field

c) Layout:

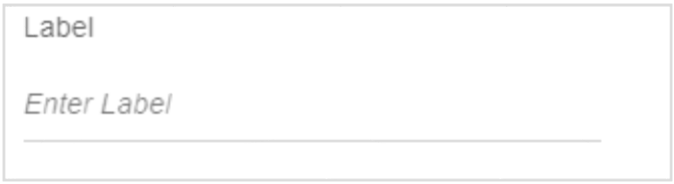
- Grid

11. Available Attributes

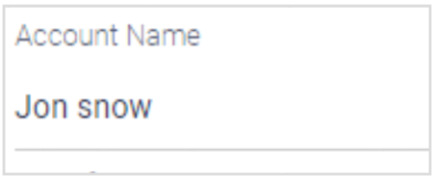
11.1 Label:

Description: This attribute is a mandatory element and is used to display the label of the element on the screen.

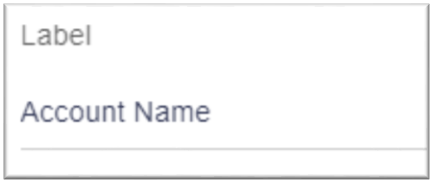
Component to accept the input: Input Text



Example: User has an input field to provide account name as shown in the image.



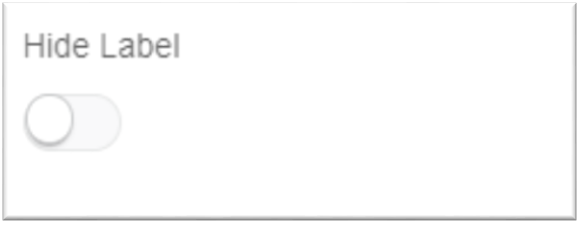
Usage: Enter the label as “Account Name” in the input text field as shown in the image.



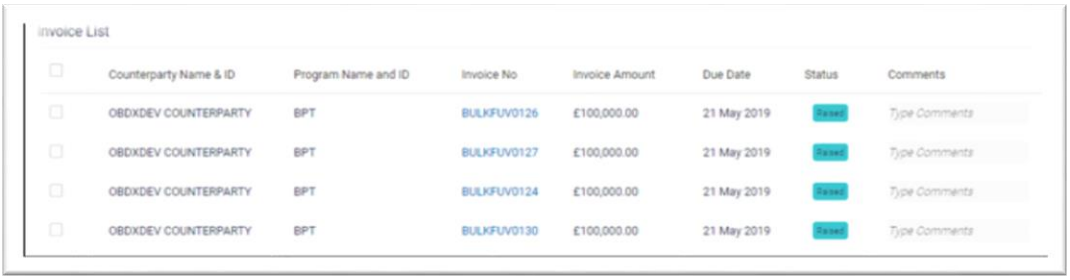
11.1.1 Hide Label:

Description: This attribute is used to hide the label of the element on the screen.

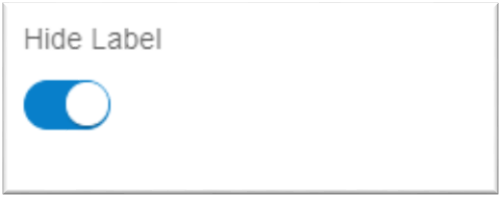
Component to accept the input: Switch



Example: In the below example, User can **apply** this component to use checkboxes that selects rows in a table without displaying the label.



Usage: Enable the switch on as shown in the image. It will disable the label of the element.



11.1.2 Format Label:

Description: This attribute is used when user wants to change the label or text dynamically.

Components to accept the input: Switch and Input text

If user enabled the switch, it will display two input texts.

1. Add variables to label
2. Enter variable's mapping

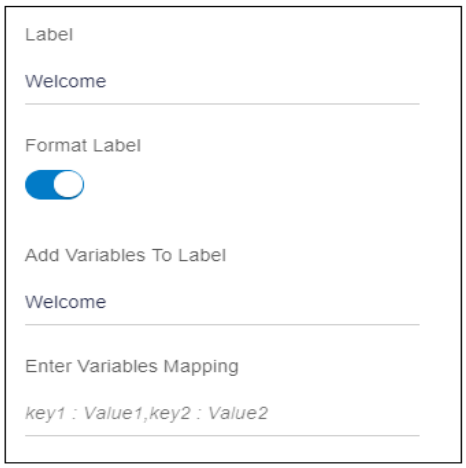
Example: In the example below, user name of logged in user and other login details are display ed. User name and Login details will be dynamic. (“Welcome <user name>” and “Last Login <login Details>”) and will change based on the current user and time.

Usage: Enter the label of an element which is static (e.g. “Welcome") in Label attribute,

Then enable the “Format Label” switch. It will display two input texts.

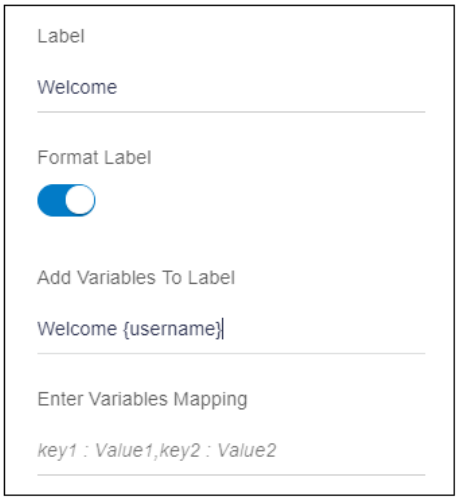
11.1.3 Add variables to label:

When user enables the switch, the label user entered in Label attribute will be pre-filled in this input field as shown in the image.



Now user needs a key to hold the value, which will change dynamically. In the example below, key name is user name. User will have to add this key name (user name) after the label (Welcome) in this field.

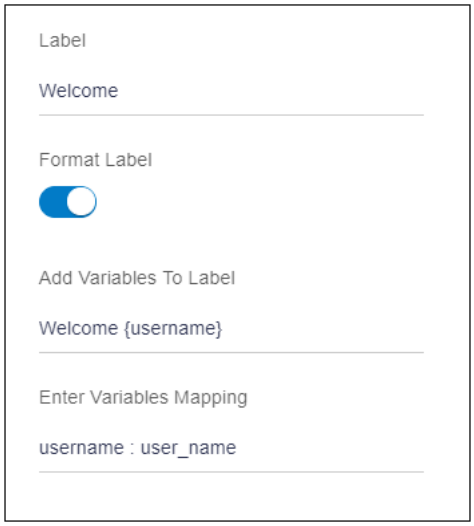
To add a key, write the key name surrounded by curly braces.



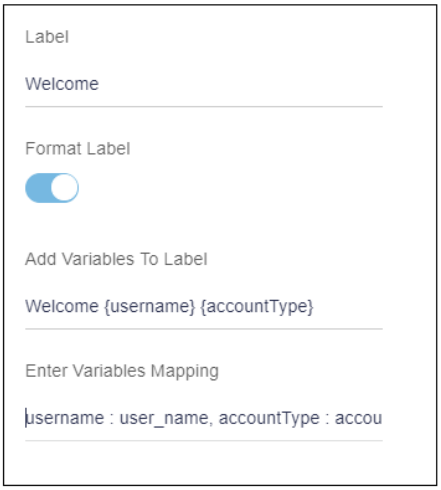
11.1.4 Enter variables mapping:

User will have to map this key to a variable, which will hold the dynamic value. In the example below, the value is stored in a variable called as user _name.

To map the key to the variable, write the key name, colon (:) and variable name.



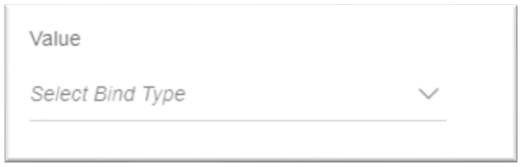
User can have multiple keys and values as shown in the image.



11.2 Value:

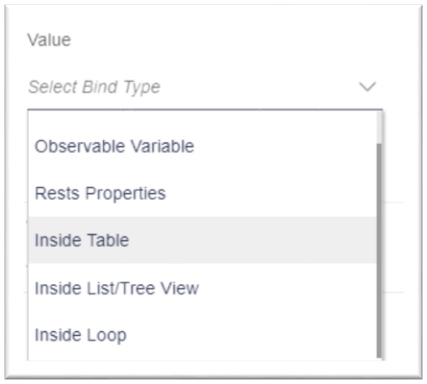
Description: This attribute is used to store the value of an element. This value can be displayed on the screen or used for further processing, such as sending to server.

Components to accept the input: Select Box

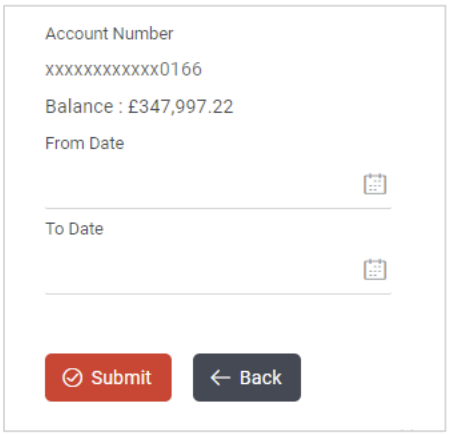


This select box has five options

a. Observable variable
b. Rest properties
c. Inside table
d. Inside List/tree view
e. Inside loop



Example: In the example below, User has a form to request a statement account for a given time period. In the form, account number component and two date pickers, **From date** and **To date** are available. User will send the values of **From date** and **To date** field to the server to fetch the details.



Usage:

User can store the values of these fields using five different ways. By selecting the type based on requirement.

a. Observable variable

In some scenarios, the data fetched from the REST API cannot be directly shown on the screen. Similarly the data submitted by the user cannot be directly saved on the server. It needs processing. To process this data, sometimes the user needs to store it in another variable. Therefore, if user wants to store the value of an element into a variable, which is defined by you, this option should be selected.

In the example below, user has a variable From Date, storing the value of From Date field.

Select the option as “Observable variable”. After selection, it will display an input field named as variable name.

Enter this variable name (fromDate) into “variable name” field as displayed in the image below.

b. Rest properties:

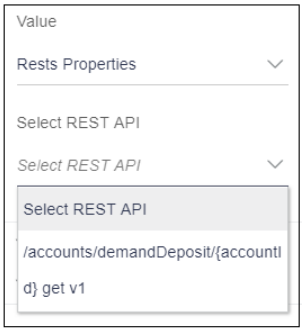
Rest API can be used for multiple requirements.

1. GET: To fetch data from the server.
2. POST: To save data on the server.
3. PUT: To modify data on the server

1. DELETE: To delete data from the server.

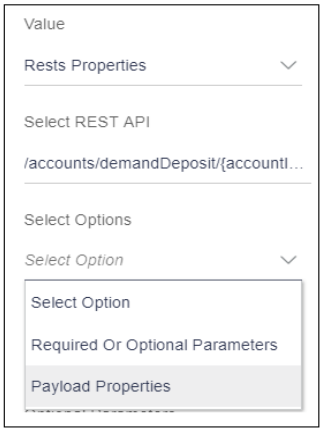
When user selects the option “Rest properties”, it will display a select box named as “Select REST API”.

Select REST API: This select box display s all the REST APIs that user has selected in Step 3 i.e. REST API Selection as shown in the below image.



When user selects a REST API from this select box, it will display another select box named as “Select Options”.

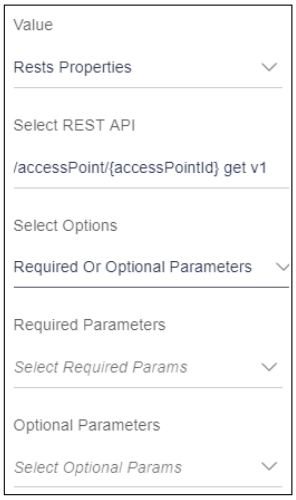
Select Options: This select box has two options as shown in the below image.



- Required and Optional Parameters :

To understand what is “Required or Optional Parameters” refer to Required and optional parameters section.

When user will select” Required or Optional Parameters” option, it will display two different select boxes.



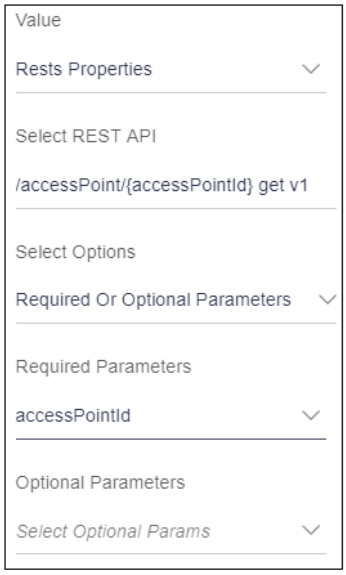
a. Required Parameters: This select box will display all the required parameters of the selected REST API.

b. Optional Parameters: This select box will display all the optional parameters of the selected REST API.

   Example: User has a REST API “/accessPoint/{accessPointId} get v1”. In this REST API, accessPointId is a required parameter and user wants this parameter to be filled as in Input and wants to map this property to input text.

   Usage:

   a. Select “REST Properties”.
   b. Select REST API i.e. / accessPoint/{accessPointId} get v1
   c. Select Required or Optional Parameters
   d. Map the accessPointId to the input text.



- **Payload Properties:**

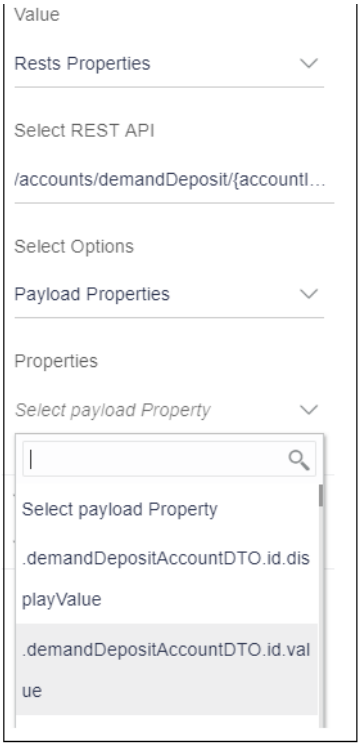
  When user select “Payload Properties”, it will display a select box named as “Properties”.

  Properties: This select box will display all the payload properties of the selected REST API.

  As discussed above, REST API can be used for multiple request types like GET, POST OR PUT.

  In the case of GET request, server sends the data to the user . In this case, this response data is considered as payload. All the payload properties listed in this select box will store the server response of that selected REST API. User can map this property to the elements, which display s the data. For example, Row control, Text.

  In the of POST or PUT request, user sends the data to the server. In this case, this request data is considered as payload. All the payload properties listed in this select box will store the data entered by the user for that selected REST API. User can map these payload properties to the elements, which accepts the input, and send it to the server.



In the above example, there are two components.

1. Row Control to display the account number

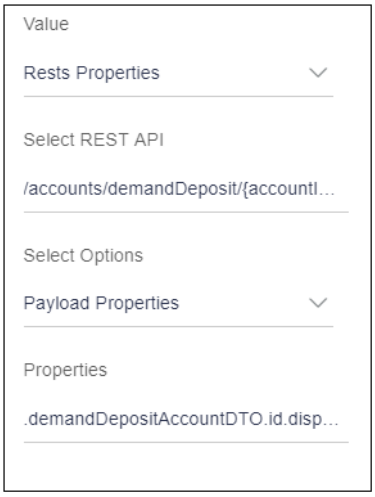
   In this case, account number is fetched from the server.

   a. Select “REST Properties”.

   b. Select REST API i.e. /accounts/demandDeposits/{accoundId} get v1

   c. Select Payload Property

   d. This is the GET request Payload Property will store the response came from the server, hence select the required property from payload i.e. demandDepositeAccountDTO.id.display value and map it to the row control.



2. Two date pickers to store the dates.

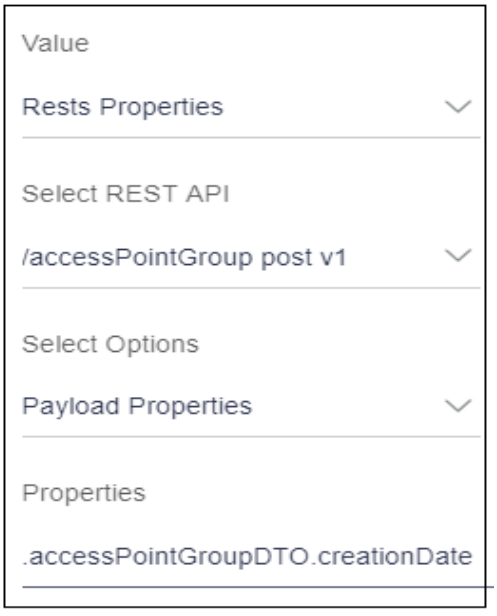
   In this case, dates need to be sent to the server.

   a. Select “REST Properties”.

   b. Select REST API i.e. /accounts/demandDeposits/{accoundId} post v1

   c. Select Payload Property

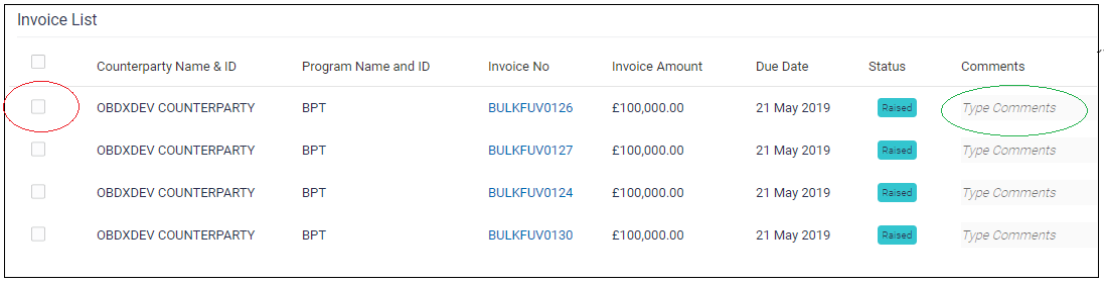
   d. This is the POST request Payload Property will store the input and this payload will be sent to the sever, select the required property from payload i.e. demandDepositeAccountDTO.id.display value and map it to the date pickers.



- **Inside table:**

If element is inside the table, then select this option.

Example: In the below image, user has Invoice list table as display. All the components, which are used, are inside the table such as checkbox, input box etc. select this option.



When user will select “Inside table” option, it will display an input box to add a variable name as shown in the below image. This variable is nothing but one of the keys available in the data source object given to the table. This variable can be used to display the data or to store the data.

Example: The datasource object looks like this:

```
datasource[0]  {
        counterPartyName: “OBDX_COUNTER_PARTY”,
        invoiceNo: ”BULKFUV0126”(this is string)
        comment: user _comment(this is variable)
}
```

Note: Datasource is an array of objects. The object shown above is just one object for the first row of the table. Data source will have an object for every row. That is why it is written as datasource[0] i.e. first object in an array.

Let us consider two cases here:

1. Invoice No:

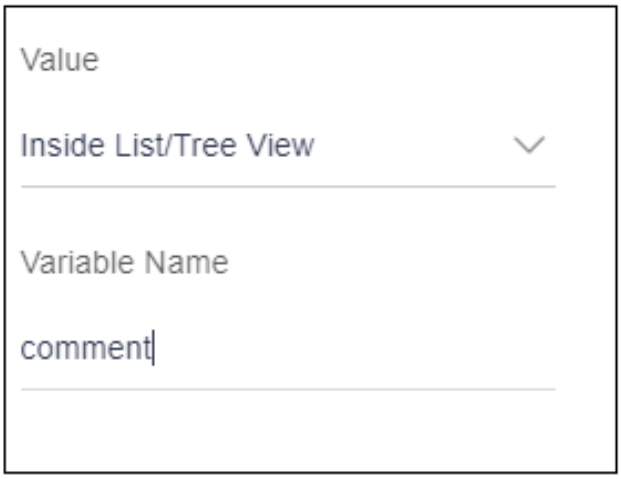
This field is displaying the data. Now user wants to display the invoice BULKFUV0126. As user can see in data source object written above, the key, which stores invoice number(BULKFUV0126), is “invoiceNo”. User will enter this key as a variable name for this field.

2. Comments:

This field is accepting the user comments, i.e. it is storing the data entered by the user. User will need a variable to store the data, which is user _comment variable. As user can see in data source object written above, the key, which has a reference to the variable(user _comment) is “comment”. User will enter this key as a variable name for this field.

- **Inside List/tree view :**

If element is inside the list or tree view, then select this option. This is similar to the table. User will have the data source having all the objects. User will write the key name into the variable name for their respective field.



- **Inside loop:**

Refer **"Add loop"** attribute to understand what is loop.

If the element is inside the loop, user will select this option.

referring to the same example of add loop attribute.

In this example, there are three fields: name, city and mobile number. They are inside the loop. for these elements user will select "Inside loop" as value.

The objective is as follows:

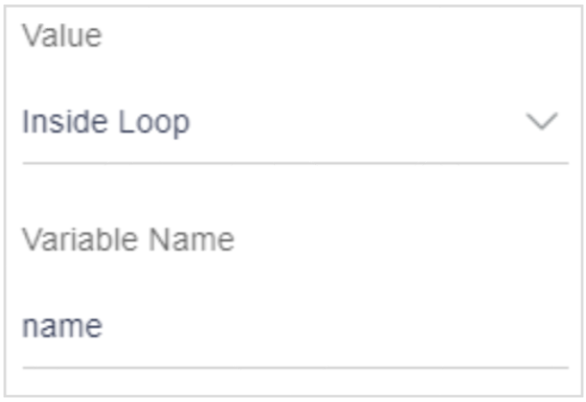
user _data = [{name: "James Smith", city: "New York", mobileNo: 3454654},

{name: "Christopher Robin", city: "Manhattan", mobileNo:4758945},

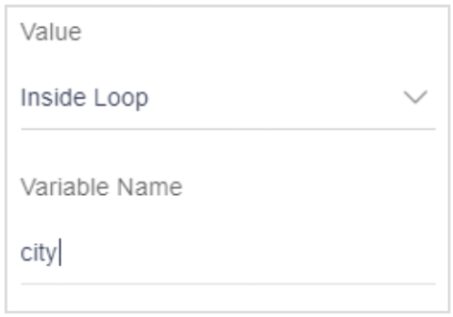
{name: "William Turner", city: "London", mobileNo:7857694}];

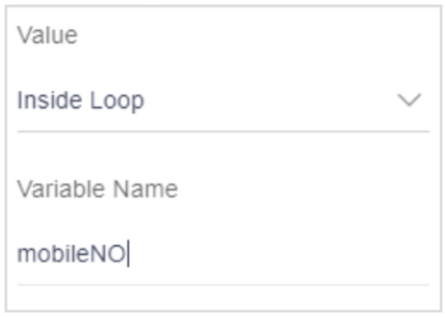
For example, user is adding the first text element, which displays the name. User will select an option as "Inside loop".

User will enter the key, which is storing the name (James Smith, Christopher Robin and William Turner) of every object. That key is "name". Enter this "name" key in variable name field as shown in the image below.



Similar for city and mobile number.





11.3 Options:

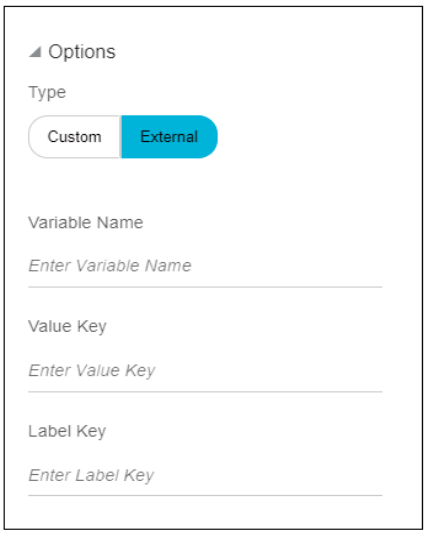
Description: This attribute is used for all the components, which enables the user to select one or more options. For example, select box, radio buttons, checkboxes etc.

Component used to accept the input: Collapsible element, Button set and Input texts.

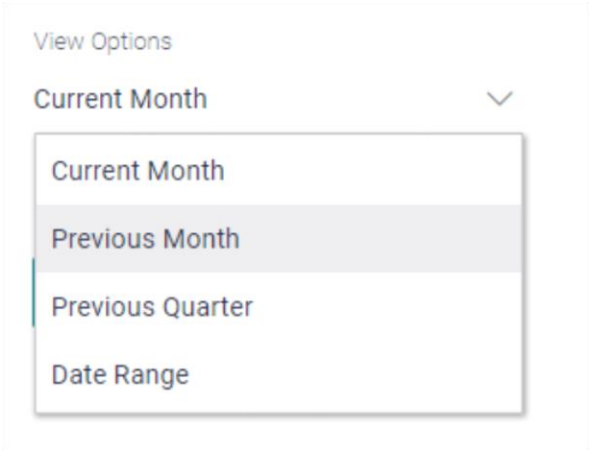
When user clicks on Options, it will expand. There are two ways to add the options.

a. External

b. Custom



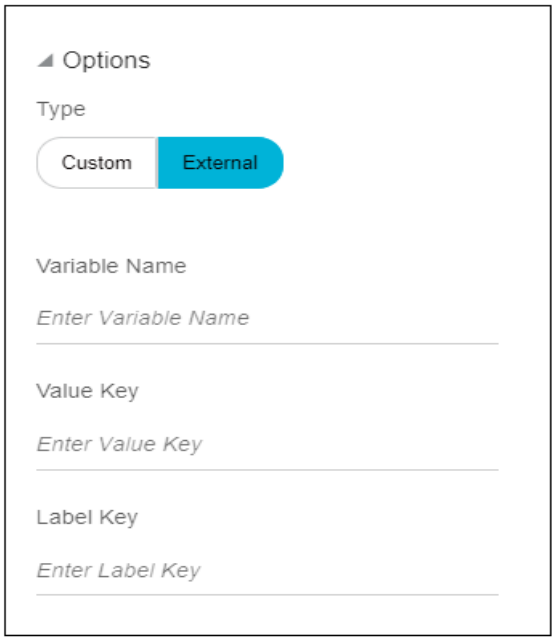
Example: User has a select box, which shows multiple options like Current month, Previous month as shown in the below image.



Usage: User can add these options using the following two ways

a. External

In this type, if user has options to be displayed on the screen which are getting fetched from a REST API, then select the type as external.



Under this type, there are three input fields.

1. Variable name: This field stores the data fetched from REST API. It is mandatory to have the data in this variable in **value=>labe**l format. In case the data is not in this format, then convert it to the required format and enter that variable name in this field.

   Example: The variable name is time_range.

   This variable has the following data which is fetched from REST API.

   time_range = [{ label : “Current Month”, value :”CM”}, { label : “Previous Month”, value :”PM”}, { label : “Previous Quarter”, value :”PQ”}, { label : “Date Range”, value :”DR”}]

2. Value key: As mentioned in point (1), value from variable name represents the actual data that will be used for further processing and is to be entered in this field.

   Here time_range is an array having multiple objects.

   In this object {label:”Current Month”, value:”CM”} , value “CM” is going to be processed.

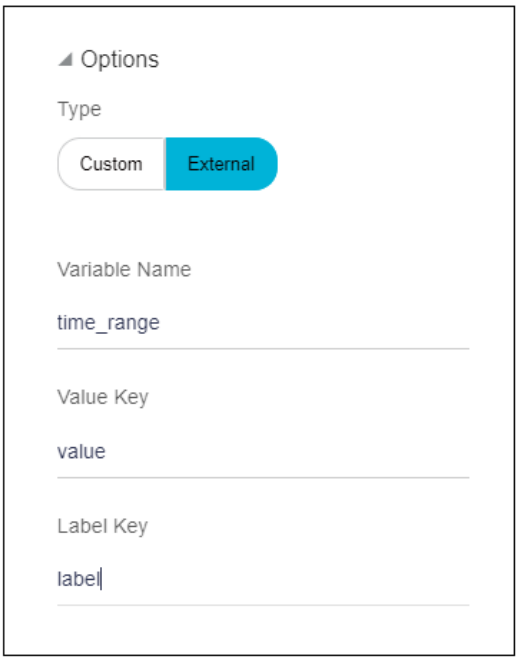
   Hence, in this case value key will be, “value”.

3. Label Key: As mentioned in point (1) above, label from variable name represents the description that needs to be displayed against an option of an input field such as select box, radio button set etc. This label is to be entered in this field.

In this object {label:”Current Month”, value:”CM”} , label “Current Month “is going to be displayed on screen,

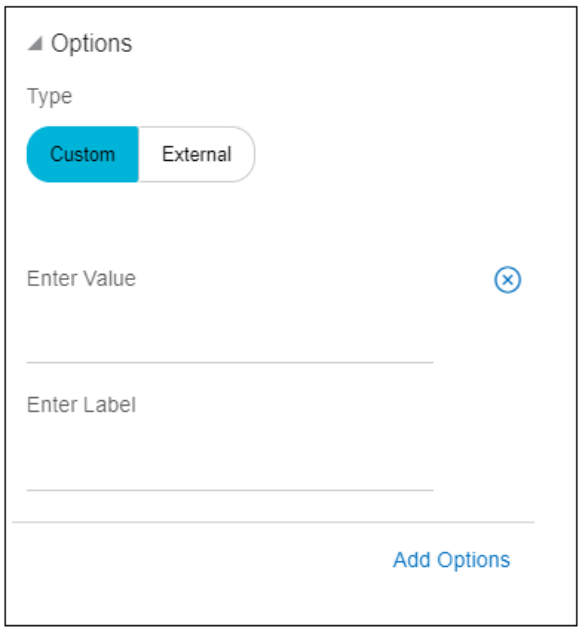
in this case label key will be, “label”.

Note: User can name this value key and label key as per their choice i.e. {label:”Current Month”, value:”CM”} can be written as {name:”Current Month”, id:”CM”}. In this case, value key will be “id”, and label key will be “name”.



a. Custom

In this type, if user wants to add the options manually, i.e. for data not fetched from any REST API, then select the type custom.



Under this type, there are two input fields.

1. Enter Value: This field represents the actual data that will be used for further processing. If user enters the value in double quotes, it will be processed as string. For special cases such as Boolean, where the value is true/ false do not enter the value in double quotes.

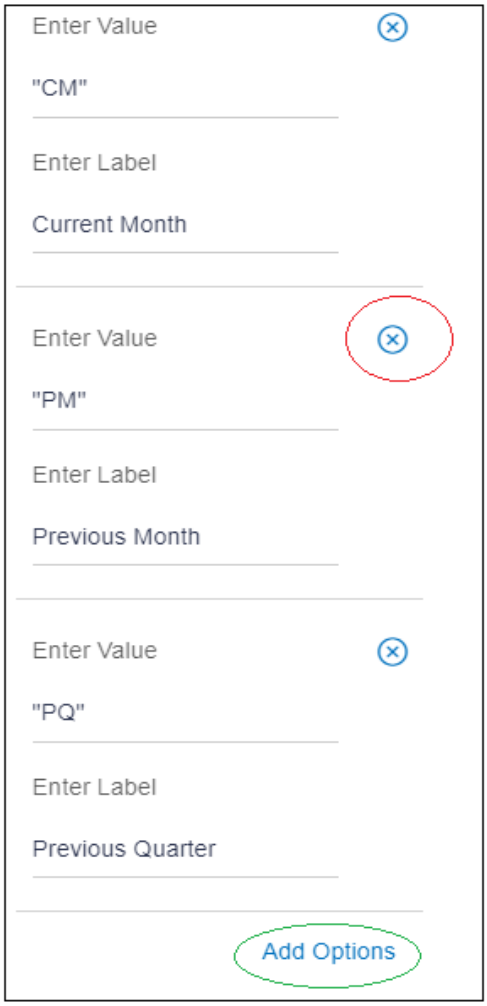
   In the below image, “CM”, “PM”,”PQ” are the values.

2. Enter Label: This field represents the description that needs to be displayed against an option of an input field such as select box, radio button set, etc.

   as shown in the image below, Current Month, Previous Month and Previous Quarter are the labels.

User can add more options using the “Add Options” link highlighted in green below.

User can delete the option using the icon highlighted in red below

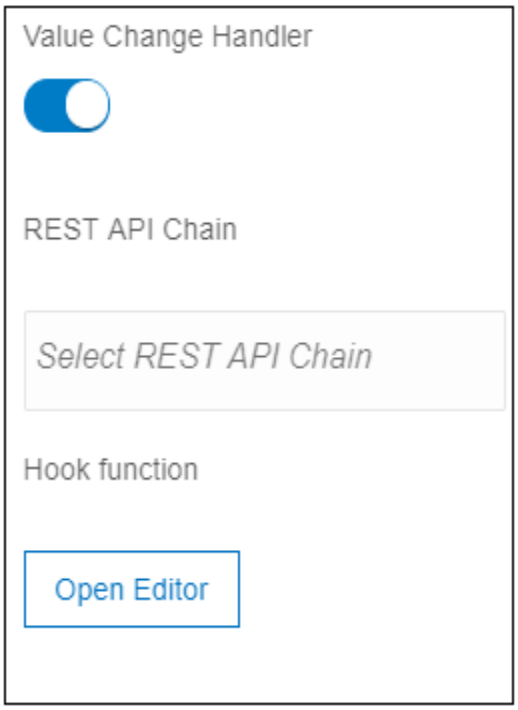


11.4 Value change handler:

Description: This attribute is used to handle the events, when the value of an element is changed.

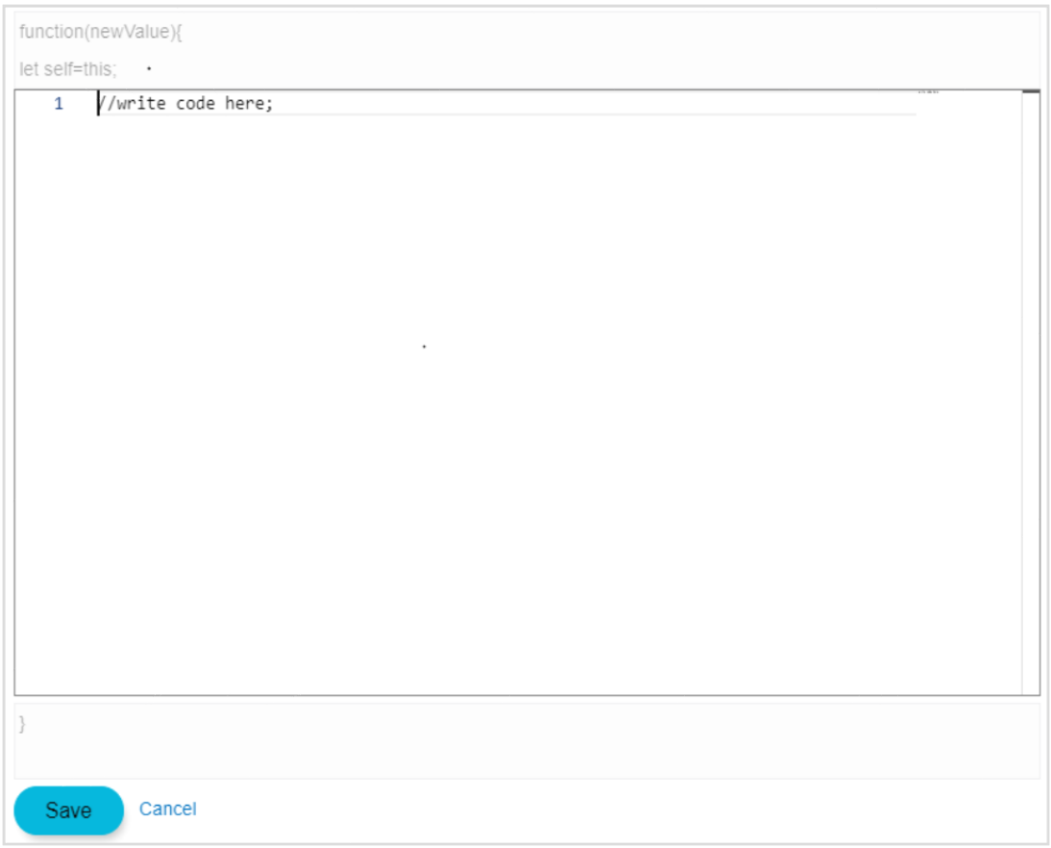
Components to accept the input: Switch, select box, and code editor

When user enables the switch, it will display an input box and a button as shown in the below image.



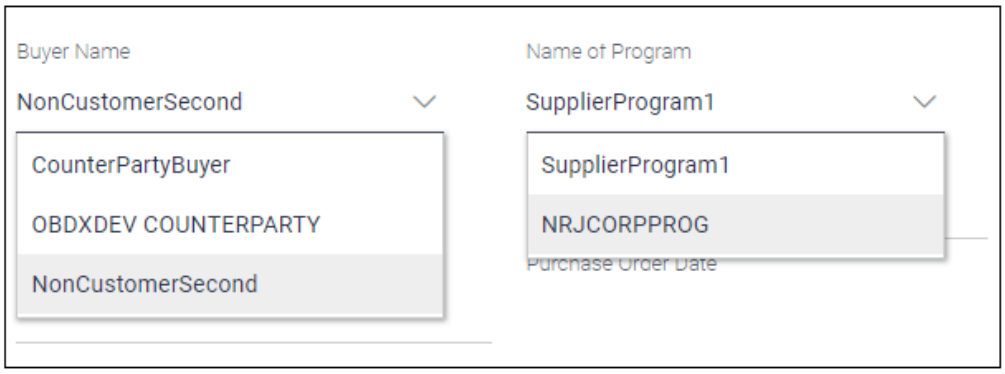
1. REST API Chain: To understand REST chaining, please refer **REST API Chaining** section. This select box lists all the REST API chaining that user has created in step four i.e. REST API Configuration. From this select box, user can select REST API chain that he wants to fire when the value of an element is changed.
2. Hook function: There is a button named as “Open Editor”. When user clicks on this button, it will open a code editor as shown in the below image. In this editor, user can write the code to be executed when the value of an element is changed.

To understand more about editor, **refer Open Editor of hook function** section.



Example:

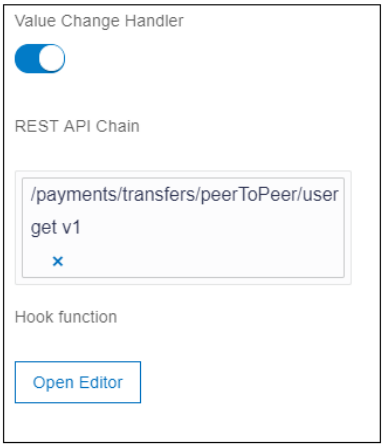
1. User has two select box options
   a. Buyer Name: This select box lists all available buyer names.
   b. Name of Program: When user selects a buyer name, this select box will lists available programs specific to the selected buyer.



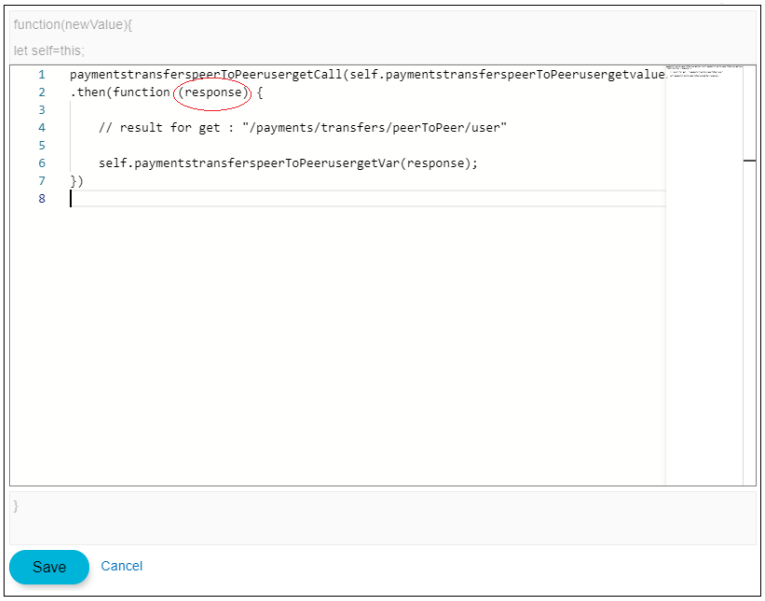
In above example, user will need to fire a REST API to fetch the available programs of the selected buyer. It means the user needs to perform an operation when the value of buyer field changes. In such scenarios, this attribute is used.

Usage: Enable the “value change handler” switch.

Select the REST API that user needs to fire when the value of **buyer** field changes as shown in the below image.



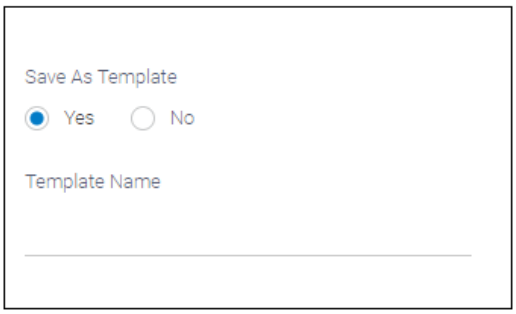
Now if user clicks on the “Open Editor” button, it will open the code editor.



A default code is already present in the window generated by the tool. When user selects a REST API, the tool generates the required code to execute that REST API. However, in some cases, there is a need to do some additional operation on the data fetched from the server. Therefore, for this purpose, there is a support for code editor. In such scenarios, use the response variable(highlighted in red circle in the above image) for further processing.

Whenever the value of buyer name field changes, it will execute the block of code written inside the editor.

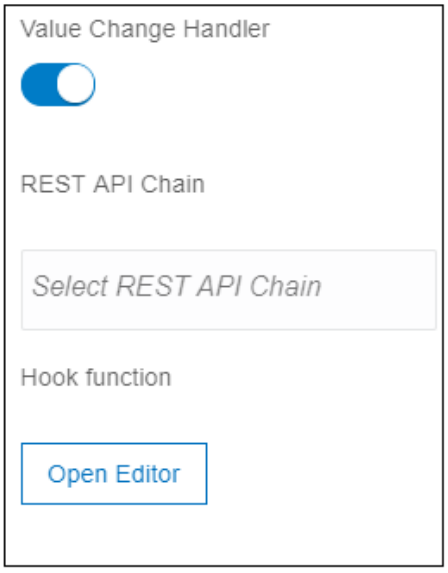
1) User has a form and wants to give an option to use this form as a template. If the user says yes, then an input text field is displayed to accept the template name as shown in the image.



Usage: In this example, user will need a variable to handle this case, say “isTemplate”. Initially this variable will have a value set as false (Boolean). it will hide “Template Name” input field. But when the user will select “Yes” option, its value will change to true (Boolean) and it will display the input field “Template Name”.

Enable the “value change handler” switch.

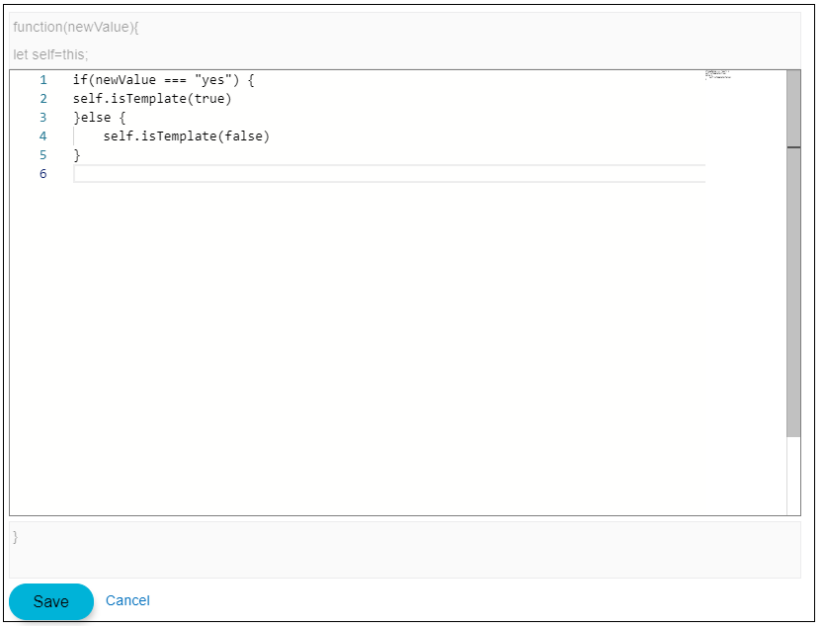
As user does not need to fire any REST API, no REST API is to be selected.



Click on the “Open Editor” button, and write code inside the editor as shown in the below image.

newValue represents the new value of the field. If user selects “Yes”, **newValue** will be “Yes” and variable will be true. If user selects “No”, **newValue** will be “No” and variable will be false. Whenever the value of radio button changes, it will execute this block of code.

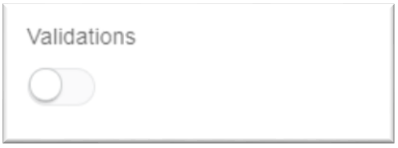
Note: As there is no REST API selected, there will be no auto-generated code.



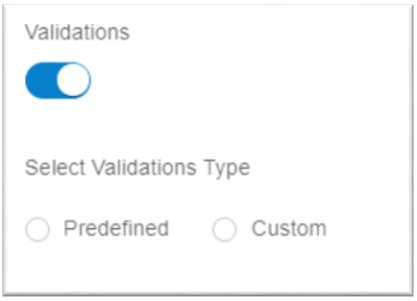
11.5 Validations:

Description: This attribute is used to add UI level validations for an element.

Components to accept the input: Switch, Radio buttons, Select box

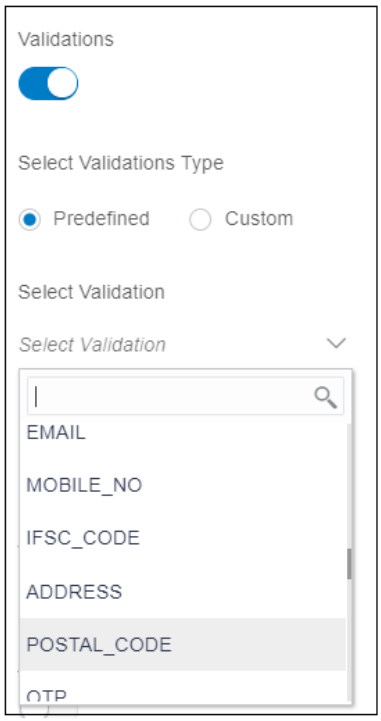


When user will enable this switch, it will display two radio buttons as shown in the below image.



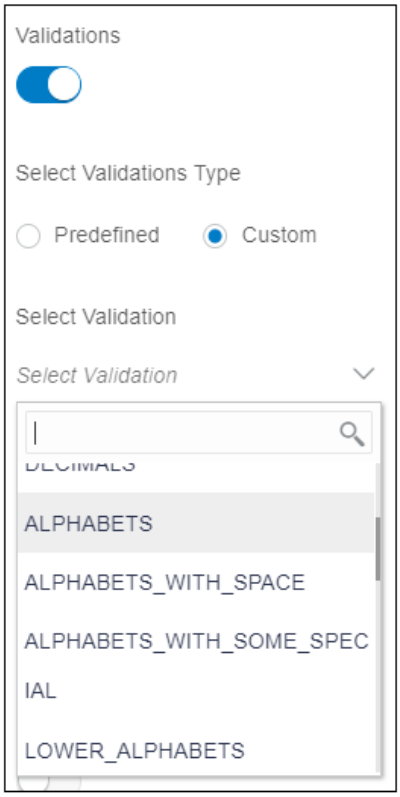
1. Predefined: There are some validations, which are common such as mobile number must have 10 digits. Such validations are already defined by OBDX framework.

   When user selects this option, it will display a select box named as “Select Validations” as shown in the below image. This select box lists all the predefined validations, which are available. For example, mobile number, address, email etc.

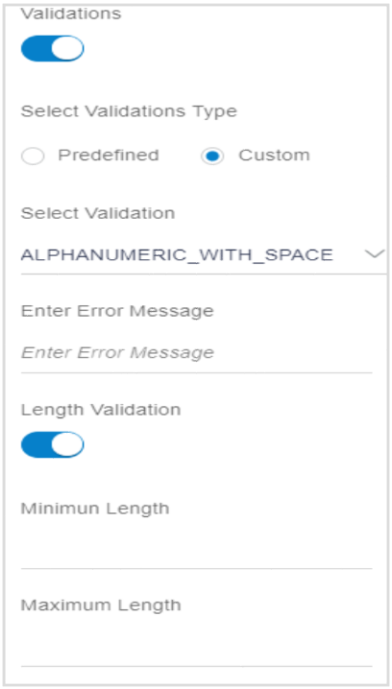


2. Custom: There are some validations which are element specific, for example, some element must have a specific length or special characters etc. Select this option to customize validations.

   When user selects this option, it displays a select box named as “Select Validations” as shown in the below image. This select box lists all the custom validation options such as “alphabets with space”, “alphabets with some special characters” etc.



Below the select box, there is an input box named as “Enter error message” and a switch for “Length validation” as shown in the below image.



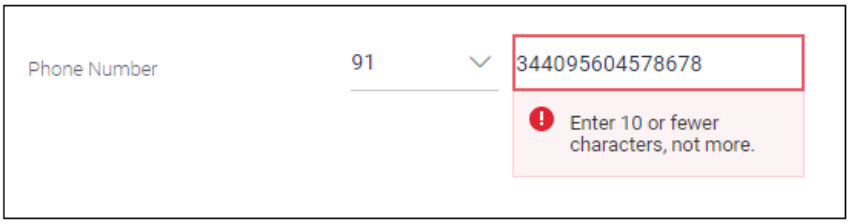
Enter error message: Enter the error message that should display on the screen when the validation fails.

Length validation: Enabled this switch, if there is a length validation. When user will enable this switch, it will display two input boxes for minimum length and maximum length.

Example: User has two fields, which needs some validations.

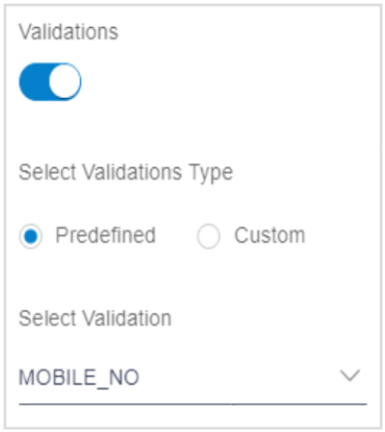
1. Input field to accept the phone number:

   Validation for phone number is “Enter 10 or fewer characters, not more.” As shown in the below image.



Usage: This validation is available in predefined validation.

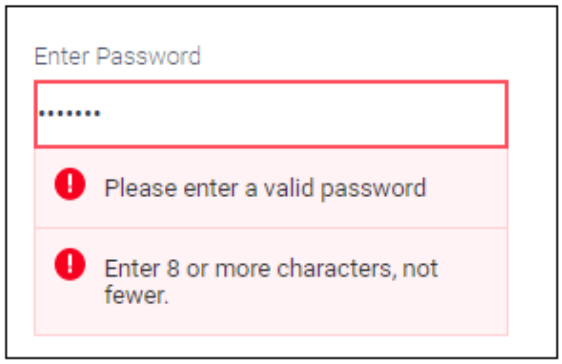
After selecting the validation, select the option, “mobile_no” as shown in the below image.



2. Input field to set the password:

   Validation for password is “Password must have minimum 8 and maximum 12 characters. It can have special characters”.

   Example of Error Message can be ‘Please enter a valid password.

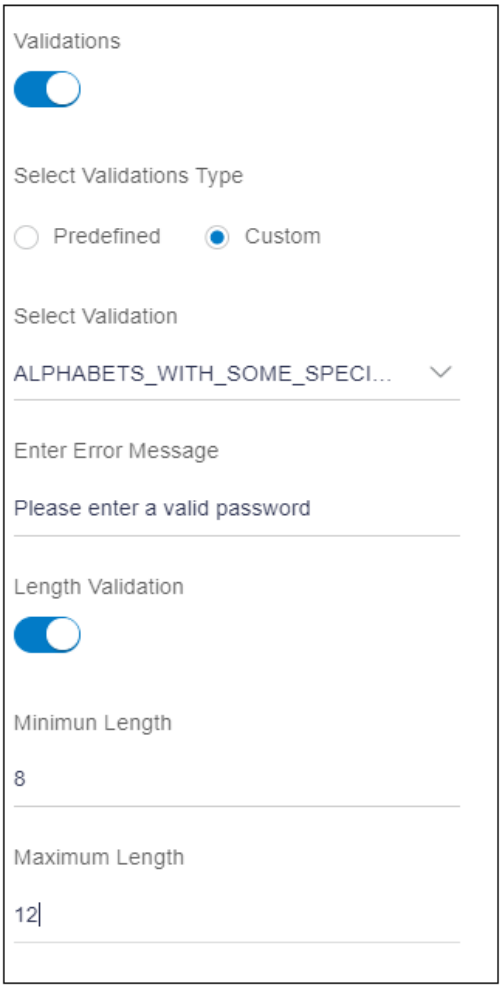


Usage: This validation is specific to password field, and hence is not available in predefined validation. The user needs to customize the validation. Therefore, user will enable the validation switch and select custom validation.

From the select box, user needs to select the option, “APHABETS_WITH_ME_SPECIAL”.

Enter the error message example ‘password is not valid’ into the input box.

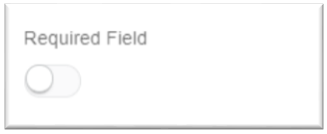
Enable the switch and enter the minimum length and maximum length in respective input fields as shown in the below image.



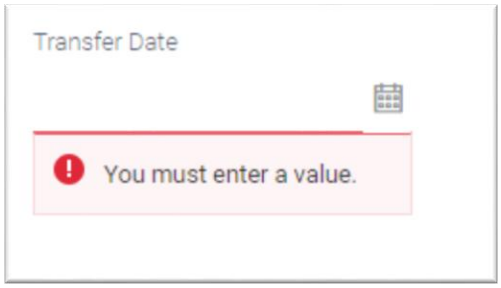
11.6 Required field:

Description: This attribute is used to make the field mandatory, i.e. user cannot leave the field empty.

Components to accept the input: Switch



In the below Example**,** User has a date picker and wants to make it mandatory i.e. user must select the date otherwise an error is will be displayed.



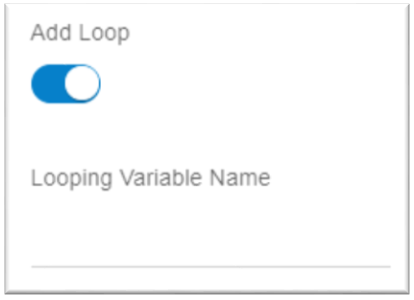
Usage: Enable the switch as shown in the below image. This will make the date picker field mandatory.

11.7 Add Loop:

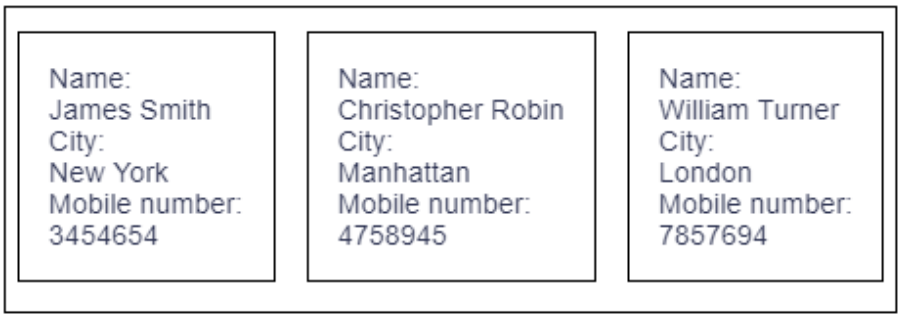
Description: This attribute is used to display the same element multiple times with different values.

Components to accept the input: Switch and input text.

When user enables the switch, it displays an input text named as “Looping variable name” as shown in the below image.



In the below example, User has a template, which displays the name, city and mobile number of the three user s.



There are two ways in which this can be achieved:

1. Writing the template 3 times for three users, which is quite cumbersome and will produce unnecessary redundancy in code.
2. Writing the template once, and using it 3 times for three user s which can be achieved using **add loop** attribute

Usage: Assume user has a normal container (For container refer **Container element**) and all the fields(name, city and mobile number) are wrapped inside this container.

The user details are stored in the variable “user _data”:

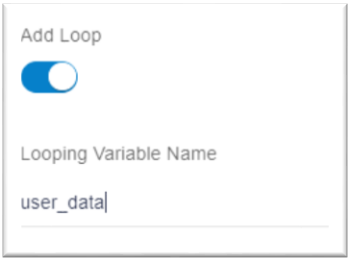
user _data = [{name: “James Smith”, city: “New York”, mobileNo: 3454654},

{name: “Christopher Robin”, city: “Manhattan”, mobileNo:4758945},

{name: “William Turner”, city: “London”, mobileNo:7857694}];

Note: user _data is an array of objects. Each object represents one user. In this array, there are three objects length of this variable is three. Hence, the container will repeat three times.

To enable the switch of add loop attribute, and user will enter user _data variable name in the “Looping variable name” input field as shown in the below image.



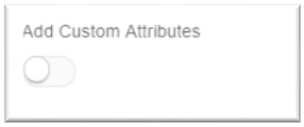
To know how to add the elements, which are inside the looping variable, (name, city, mobile number) refer **to Inside loop attribute** section.

Note: In this example, looping variable has been added to the container. But user can add the looping variable to any element using the same process (For example, input text, anchor tag etc.) In that case, input text and anchor tag will be repeated as many times as the length defined by the looping variable.

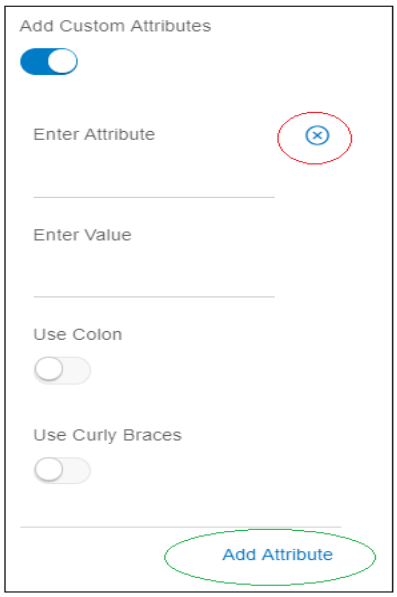
11.8 Add Custom Attributes:

Description: This attribute is used to add custom attribute, i.e. attribute not supported by the tool.

Components to accept the input: Switch, Input box



When user enables the switch, it will display two input boxes and two switches as shown in the image. User can add more attributes using the link “Add attribute” highlighted in green and can delete the attribute using the icon highlighted in red.



a. Input boxes :

   1. Enter attribute: This field represents the attribute name.
   2. Enter value: This field represents the attribute value. It supports both strings and variables. For strings use double quotes(“”).

b. Switch:

   Attributes support two types of bindings : one way binding and two way binding. To understand one way binding and two way binding refer: https://knockoutjs.com/documentation/value-binding.html

   1. Use colon: This option is only valid if the user is using one way binding to add attributes to the JET components, for example, oj-input-text, oj-select-one etc. Use this field as some of the JET attributes need colon before attribute name. If this switch is enabled, attribute name will start with colon. Refer JET site to know whether the attribute needs colon or not. http://jet.us.oracle.com/6.1.0/jetCookbook.html
   2. Use curly braces: If this switch is enabled, attribute value will be surrounded by curly braces.

For one way binding, use square brackets.

For two way binding, use curly brackets.

Example: User can use this attribute in two cases.

1. User has a list and wants to refresh it whenever the data changes. To refresh the list, its ID attribute is required. Tool generates ID for every element but it is random and will change every time the component is edited. Therefore, this ID cannot be used to refresh the list. Hence, user can add new id attribute. It will replace the id generated by the tool. In the below example, “listview” is taken as an attribute.

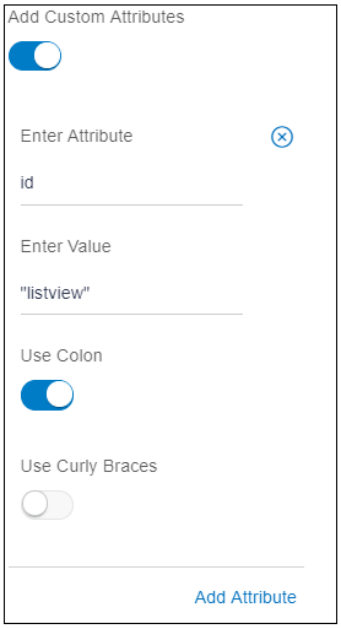
Usage: Enable the switch.

Enter the attribute name in “Enter Attribute” field. Attribute name is “id”.

Enter the attribute value in “Enter Value” field. Attribute value is “listview”. As it is a simple string, use double quotes as shown in the image below.

This id attribute needs colon. enable the switch.

This attribute needs one way binding, it needs square brackets. Hence, do not enable the switch as shown in the image below.

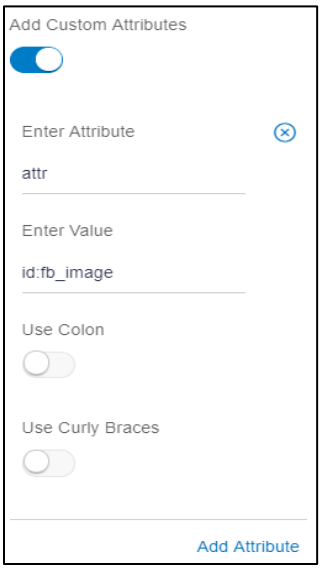


2. A single element can support many attributes and it is not feasible to add each attribute. Therefore, there will be some attributes, which will not be available inside the tool user can add such attributes using this custom attribute.

Note: When user add attributes to the Non-JET components, some attributes goes inside **attr** attribute. Refer https://knockoutjs.com/documentation/attr-binding.html

In that case, Attribute Name will be attr and Attribute value will be actual attribute user want to add.

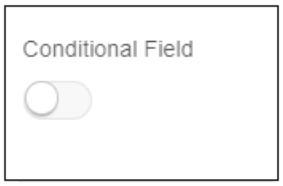
Assume user want to add id attribute to an Image, i.e. <img> html tag. In this case, id attribute goes inside attr attribute. Therefore, Attribute Name will be attr and Attribute value will be id: fb_image as shown in the image below.



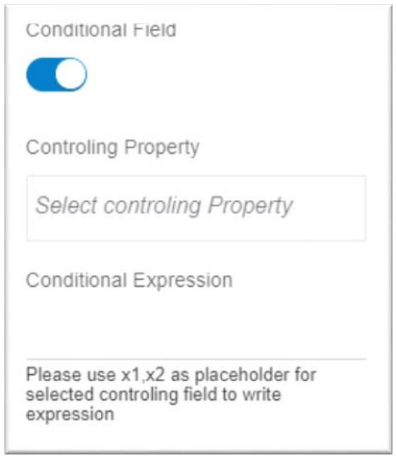
11.9 Conditional Field:

Description: This attribute is used to display an element only if given condition gets satisfied.

Components to accept the input: switch, select box, and input text.

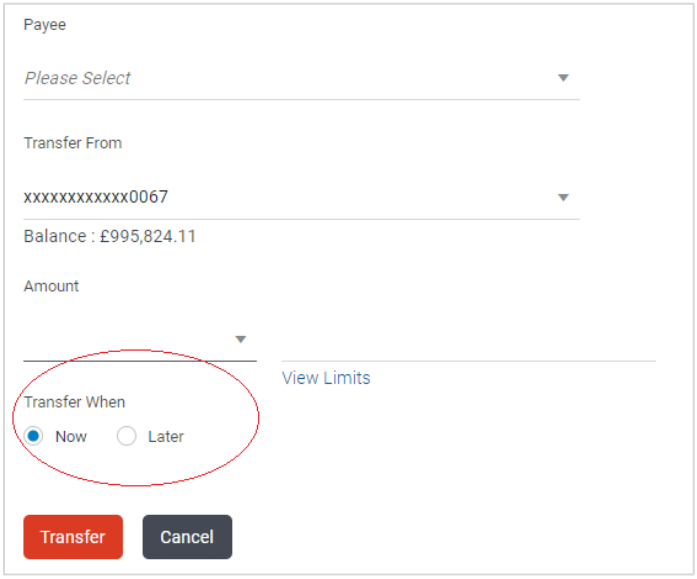


When user enable the switch, it will display a select box and an input text as shown in the image below.



1. Controlling property: This select box lists name of the all the elements which user have added. If user want to display element based on the value of a different element, then user can use all these elements available in the select box to write a condition.
2. Conditional Expression: This field accepts the condition that needs to be satisfied.

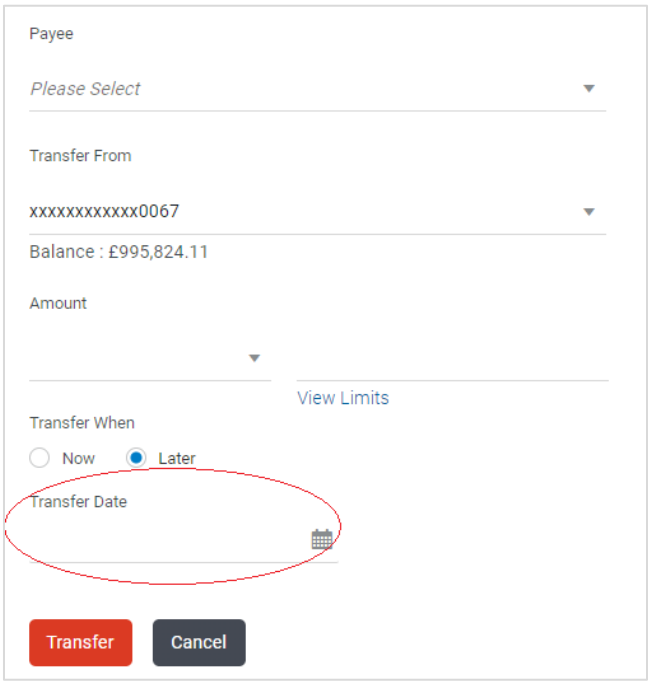
   Example: User have a form to transfer money from account to another account. That form has select box to select a payee, account input to select an account from which user want to transfer the money, input field which accepts the amount and radio buttons which accepts the date i.e. when to transfer the money as shown in the image.



   This radio button (Transfer when) has two options.

   a. Now : If user wants to transfer the money today only, he will select this option.
   b. Later: If user wants to transfer the money on a different date, he will select this option.

   Now when user selects “Later” option, there should be a date picker to accept the date as shown in the image below.



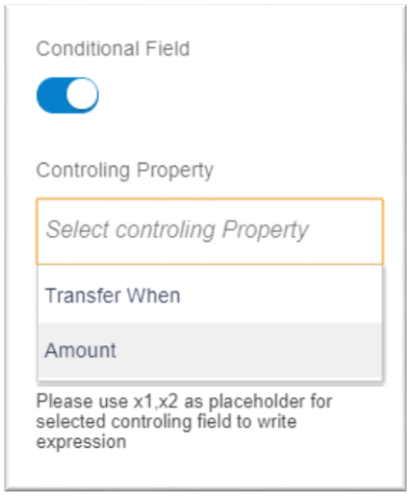
In this case user have to display the date picker “Transfer date”, if user selects “Later” option. This is condition.

Usage:

Assuming user has already added the radio buttons, and amount field.

Now when user will add “Transfer date” date picker, user will enable the conditional field switch. It will display a select box (Controlling property) and input text (Conditional Expression).

3. Controlling property: This select box will list name of all the elements. Therefore, it will display name of radio buttons (“Transfer when”)and amount(“Amount”) field as shown in the image below.



If date picker based on the value of radio button is to be displayed, user will select the “Transfer When”.

4. Conditional Expression: User will have to add the condition. There is a help text written just below this input field.

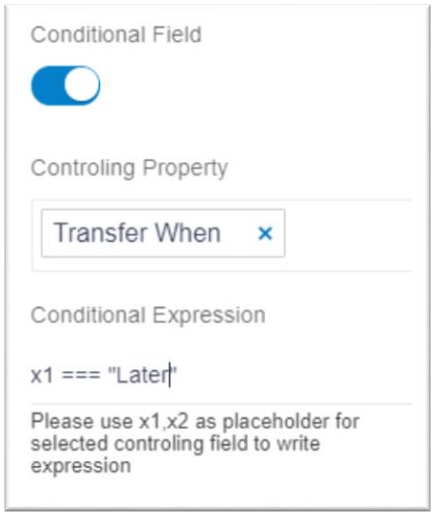
   “Please use x1, x2 as placeholder for selected controlling field to write expression”.

   It implies when the condition is written, the name of an element that user have selected from Controlling Property, instead refer them as x1,x2 and on is not mentioned.

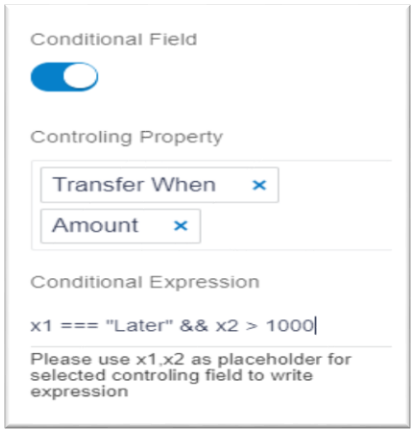
   Therefore, in this example, user will write “Transfer When” as x1.

   condition will be x1 === “Later”.

   Write this condition in the input field as shown in the image below.

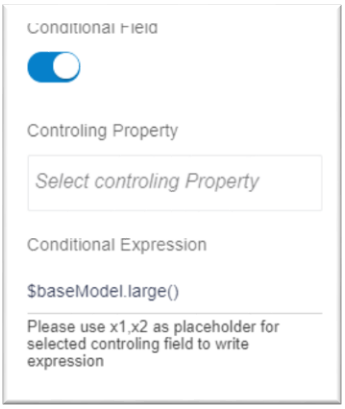


Note: Sometimes, condition is a combination of multiple elements. In that case, user can select multiple elements from Controlling property. For example, user wants to display the date picker, only if user selects later and amount entered is greater than 100INR.Then user can select both “Transfer when” and “Amount”. And condition will be x1===”Later” && x2 > 100 as shown in the image below.



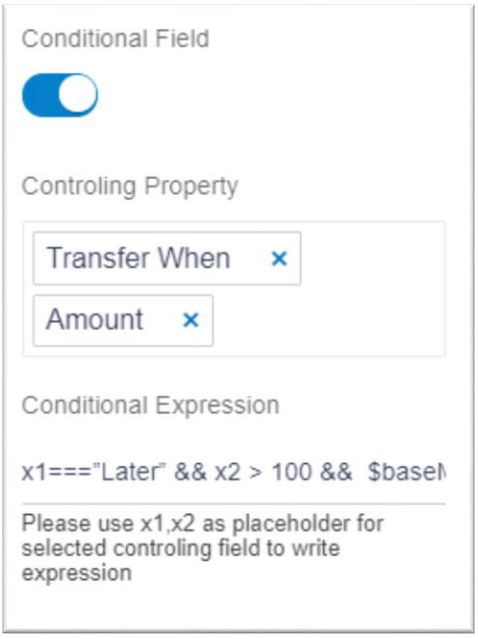
5. Condition cannot always be depend on element's value that user have added. For example, user has only one condition, display the date picker in case of large screen only. This condition is not based on any element that user have added. In such cases user can directly write condition in Conditional expression field. As there is no controlling property, there will be no x1, x2 placeholder.

   Condition will be "$baseModel.large()" ($baseModel.large() returns true if it is large screen) as shown in the image below.



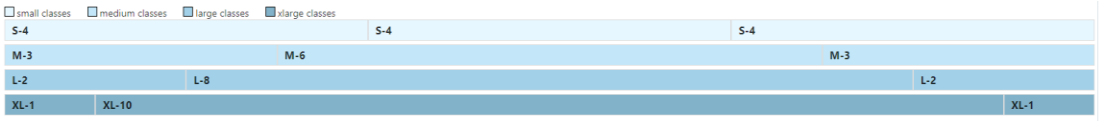
6. With controlling properties, user can add me different condition al. For example, user has a condition, which is a combination of all the three conditions, which are discussed above.

   condition will be x1==="Later" && x2 > 100 && $baseModel.large() as shown in the image below.



11.10 Grid:

The grid section is used to design grid structure supported by Oracle JET. Refer http://jet.us.oracle.com/jetCookbook.html?component=grid&demo=gridresponsive to learn how to use grid structure.



Usage: By default tools support Oracle JET oj-form-layout (refer http://jet.us.oracle.com/jetCookbook.html?component=ojFormLayout&demo=formverticalofl).

The default layout will be the one which user selected on Layout Selection page (refer **Layout Selection** Page). But there are limitations of oj-form-layout. Not every element is supported by oj-form-layout and not every structure can be created by oj-form-layout, hence grid layout can be used to resolve this problem. Use of grid is pretty simple.

Creating grid Layout: Select grid option to enable grid layout.

a. Flex :

This property provide new flex to the selected element. And the current element will be wrapped as flex-item in this Flex. But in case user does not select this option and previous element has Flex true than this element will be added as flex item in the previous element. It is okay if this all seems little complex, we will clear this in upcoming examples.

b. Flex Item Label Class:

In case of form element like input box, select box, etc. this property is used to decide the width and position of label. In case form element does not have label or user does not want to display the label, keep the switch off.

c. Flex item label Class:

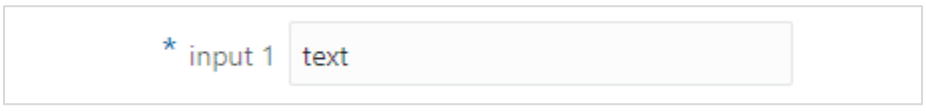
This field is used to get configuration for width and position of current element.

Examples:

Case:

a) Grid: true, flex : true; Flex item label class: true,

Label Class: ‘oj-lg-3 oj-md-4 oj-sm-12’, Flex Item Class: ‘oj-lg-4 oj-md-5 oj-sm-12’



b) Grid: true, flex: false, previous element flex: true, Flex item label class: true

Label Class: ‘oj-lg-2 oj-md-4 oj-sm-12’, Flex Item Class: ‘oj-lg-3 oj-md-5 oj-sm-12’

c) Grid: true, flex: true, Flex item label class: false

Flex Item Class: ‘oj-lg-3 oj-md-5 oj-sm-12’

Open Editor of hook function: Whatever is written inside this code editor goes inside the function.

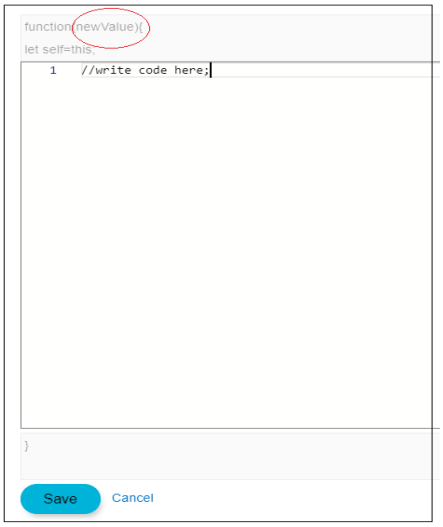
Function has three parts:

function function_name(function_parameters) {

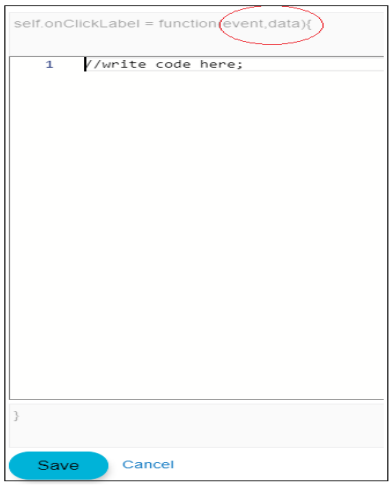
Function_body

}

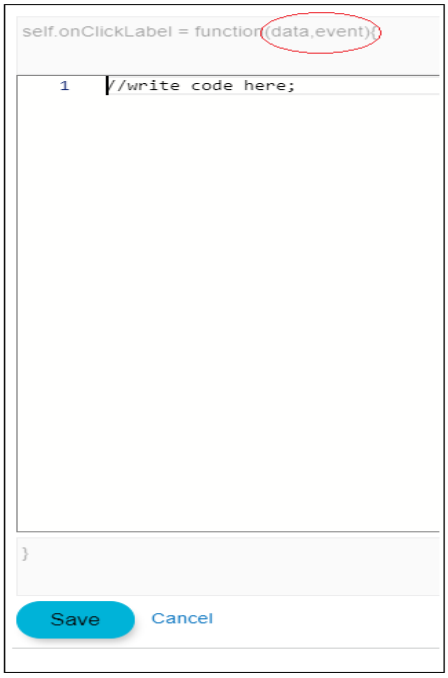
1. function_name: Function name will be generated by the tool
2. function_parameters:
   a. In case of JET components, which support value attribute for example, oj-input-text, oj-select-one etc. function parameter will be newValue as shown in the image below. newValue stores a new value of the field. To know more about JET components refer http://jet.us.oracle.com/6.1.0/jetCookbook.html.



   b. In case of JET components, which do not support value attribute, for example, oj-button, oj-menu, oj-file-picker etc. function parameter will be event and data as shown in the image below.



c. In case of Non JET components i.e. pure html components, for example, anchor tag function parameters will be data and event as shown in the image below.



1. function_body: User can write own code inside this body. If user select any REST API from REST API Chain select box, it will have some auto-generated code. Otherwise, it will be an empty body.

There are two buttons.

1. Save Button: It will save all the code user has written.
2. Cancel Button: It will close the code editor without saving anything user has written after opening the code editor.

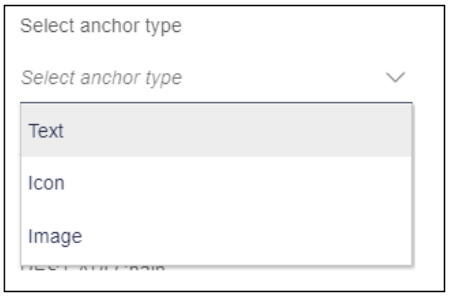
11.11 Select anchor type:

Description: This attribute is used to determine the type of anchor tag.

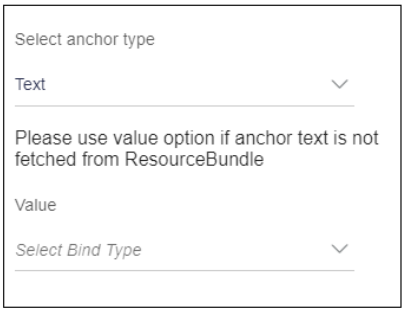
Components to accept the input: Select box

This select box has three options as shown in the image below.

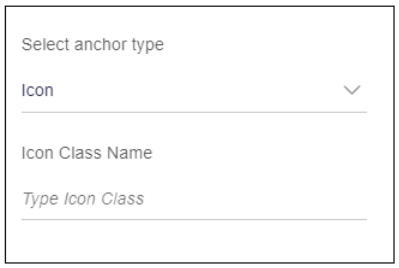
1. Text
2. Icon
3. Image



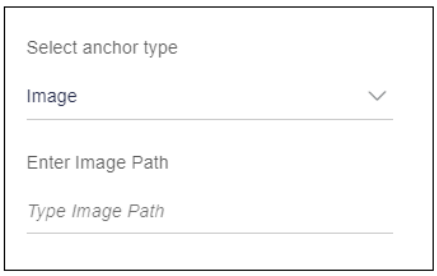
1. Text: This option is used for simple text. When user select “Text” option, it shows “Value” select box as shown in the image below.



2. Icon: This option is used for icons. When user select “Icon” option, it shows an input box named as “Icon class Name” as shown in the image below. This input text accepts an icon class.

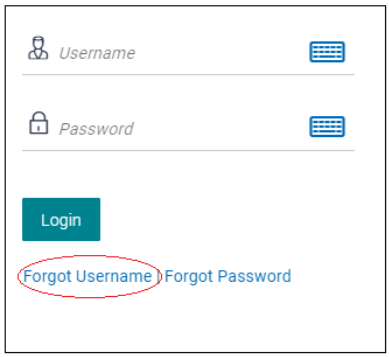


3. Image: This option is used for image. When user select “Image” option, it shows an input box named as “Enter Image Path” as shown in the image below. This input text accepts image path where it is located.

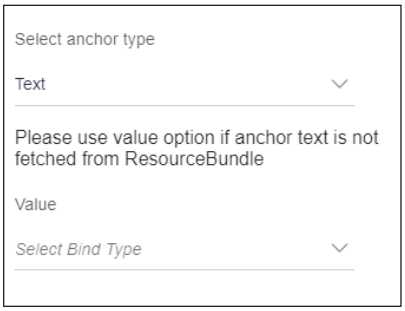


Example: Now anchor tag can be used with simple text, icon or image.

1. Text: User have a simple text “Forgot User name” that user want to use as a link, as shown in the image below highlighted in red oval.



Usage: In this type, select this option as shown in the image below.



As user can see in above image, there is small information text, “Please use value option if anchor text is not fetched from ResourceBundle”.

It means, if text is not a simple string, i.e. it is fetched from the server or stored in variable then user can use value option available just below the text.

To understand value attribute refer value attribute section.

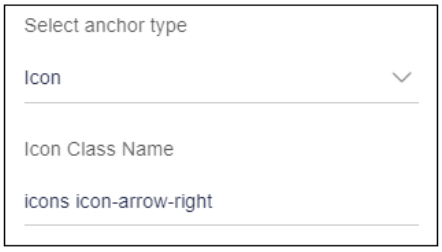
In this example, “Forgot User name” it is a simple string, do not select anything from “Value” select box.

2. Icon: User have an icon (>) that user want to use as a link, as shown in the image below highlighted in red oval.

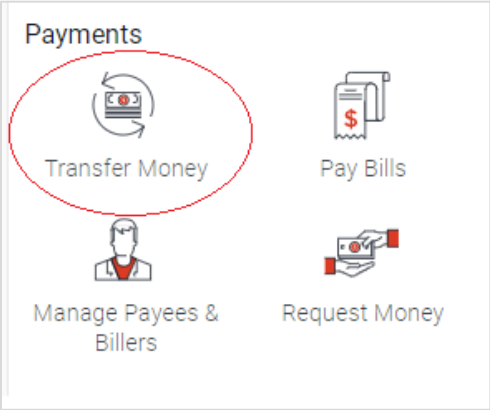
Usage: In this type, select this option as shown in the image below.

It will display an input box to enter the class name for an icon.

For this icon, class name is “icons icon-arrow-right”, enter it in the input box as shown in the image below.



3. Image: User have an image that user want to use as a link, as shown in the image below highlighted in red oval.

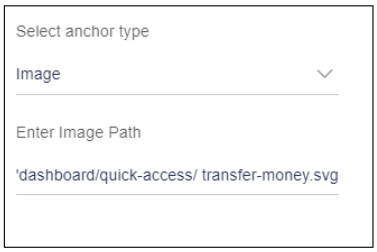


Usage: In this type, select this option as shown in the image below.

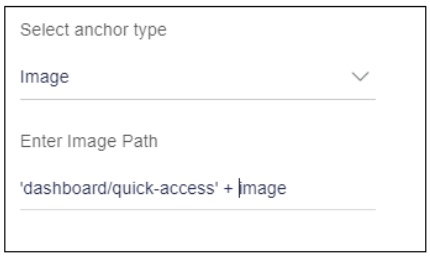
It will display an input box to enter the path for an image.

For this image, path is "dashboard/quick-access/ transfer-money", enter it in the input box as shown in the image below.

Note: Use single quotes if path is of type string.



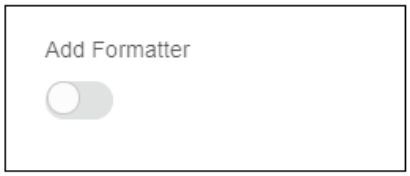
If "dashboard/quick-access" is string but "transfer-money" is stored in image variable. Then path will be 'dashboard/quick-access' + image



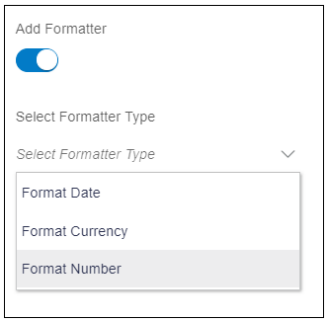
11.12 Add formatter:

Description: This attribute is used to add formatter to format a date, currency or number.

Components to accept the input: Switch, Select box and Input box

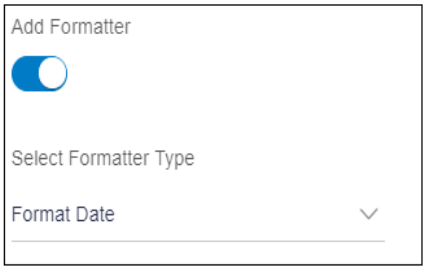


When user enables this switch, it will display a select box named as “Select Formatter type”.

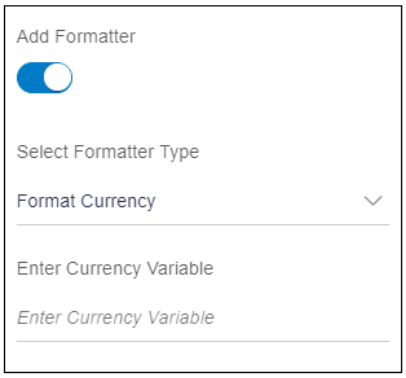


This select box has three options:

a) Format Date: To format a date

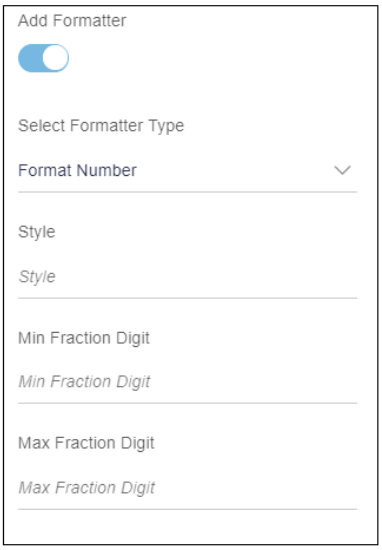


b) Format Currency: To format currency as per the standard format for a particular currency.



When user selects this option, it will display an input box named as “Enter Currency Variable”. Add variable name which has currency value.

c) Format Number: To format a number into percent value.
When user selects this option, it will display three input boxes as shown in the image below



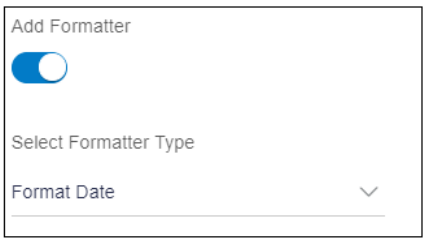
a) Style: This field is used to specify the style which should be used for formatting the number, like percent or decimal
b) Min Fraction Digit: This field is used to specify the minimum digits permissible after the decimal.

c) Max Fraction Digit: This field is used to specify the maximum digits permissible after the decimal.

Example:

1) Format Date: User has a date like “2017-10-03T19:43:45.695Z”. And user wants to format it using dateTimeStampFormat, which will be “04 Oct 2017 01:13:45 AM”.

   Usage: Enable the switch, and select type “Format Date”



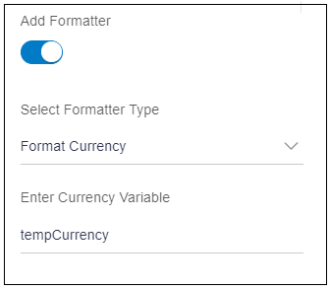
2) Format Currency: User has an amount “2,502.25” and user want to format it using currency “GBP”, which will be “£2,502.25”.

Usage: Enable the switch.

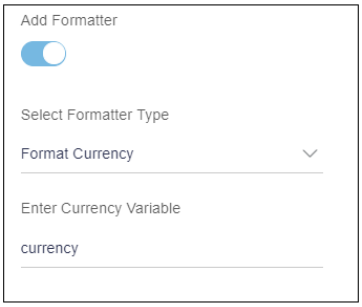
Here what user write in “Enter Currency Variable” input box depends on what user select in value attribute. Refer **value attribute** section

If user selects “Observable variable / Inside Table/ Inside List/TreeView”, then user will write variable name in which this currency has been stored.

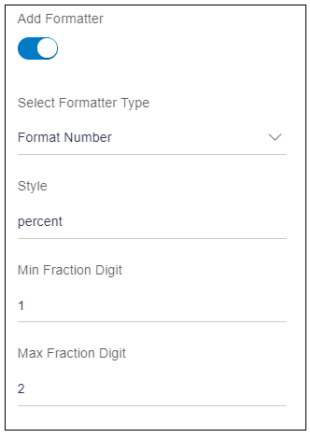
Assume variable name is “tempCurrency”. Its value is “GBP”.



If user select “Rest Properties”, then user will write “currency” word as shown in the image below.



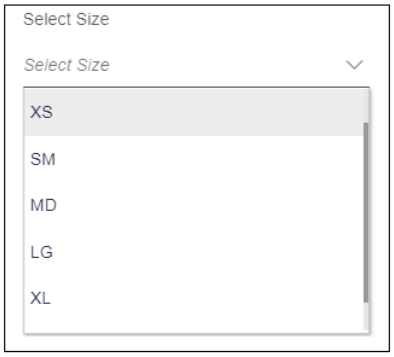
3) Format Number: User has a number “63.2512” and want to format it with percentage. Minimum fraction digit is 1, and maximum is 2. It will be 63.25%.
Usage: Enable the switch, select “Format Number” type. Enter style as percent (do not use single quotes), and fractions digits as shown in the image below.


Add Formatter
Select Formatter Type
Format Number
Style
percent
Min Fraction Digit
1
Max Fraction Digit
2

11.13 Select Size:

Description: This attribute is used to specify the size of the avatar.

Components to accept the input: Select box

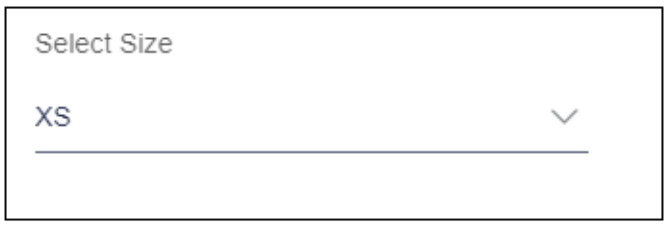


This select box has seven options:

1. XXS : Double extra small
2. XS: Extra small
3. SM: Small
4. MD: Medium
5. LG: Large
6. XL: Extra large
7. XXL: Double extra large

Example: User has an avatar, and user wants to specify the size as extra small. To know more about avatar size refer http://jet.us.oracle.com/6.1.0/jetCookbook.html?component=avatar&demo=basic

Usage: Select XS option as shown in the image below.



11.14 Enter Image Path:

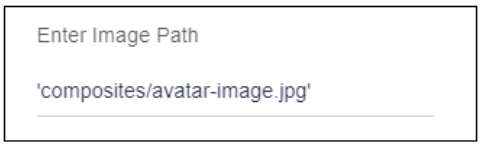
Description: This attribute is used to specify the path for the image of the avatar. Image will be rendered as a background image.

Components to accept the input: Input Text

Example: User wants to add an image in avatar located at “composites/avatar-image.jpg”.

Usage: Enter path “composites/avatar-image.jpg” in an input box as shown in the image below.

Note: Use single quotes if path is of type string

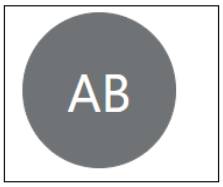


11.15 Enter Initials:

Description: This attribute is used to specify the initials of the avatar. It will only be displayed if the source (src) attribute i.e. path of an image is null or not specified.

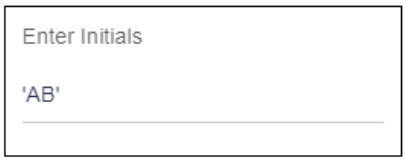
Components to accept the input: Input Text

Example: User wants the initials as “AB” in avatar.



Usage: Enter the initials “AB” in an input box as shown in the image below.

Note: Use single quotes if initials are of type string.

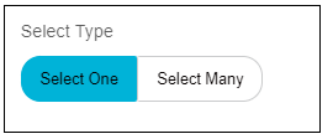


11.16 Select Type:

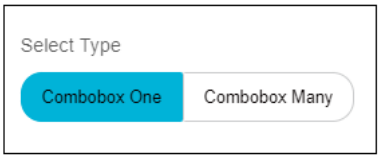
Description: This attribute is used to specify whether to allow single or multiple option selection. By default, it will be single selection.

Components to accept the input: Buttonset

For Select:



For ComboBox:



There are two types of button available.

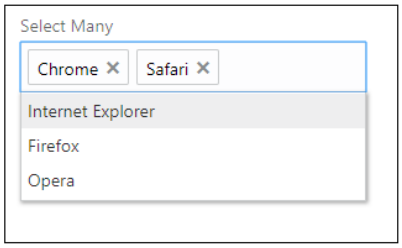
1. Select/ComboBox One : For single select
2. Select/ComboBox Many : For multi select

By default, “Select/ComboBox One” will be selected.

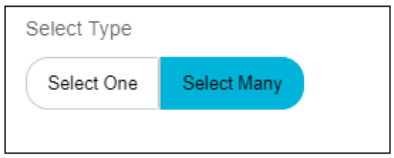
Example:

1. For Select Component:

User wants multi select drop down as shown in the image below.

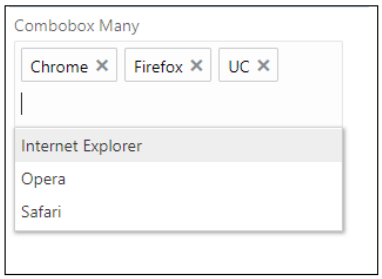


Usage: Select type “Select Many” as shown in the image below.

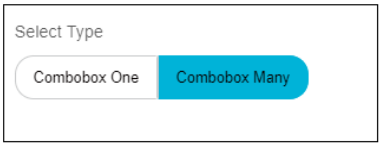


2. For ComboBox Component:

User wants multi select combobox as shown in the image below.



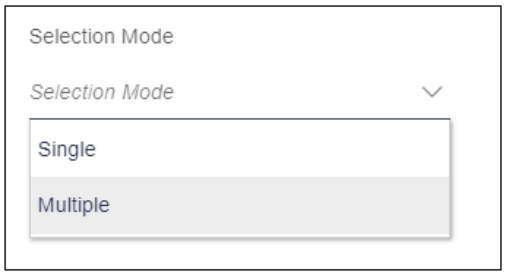
Usage: Select type “ComboBox Many” as shown in the image below.



11.17 Selection Mode:

Description: This attribute is used to specify whether to allow single or multiple file selection.

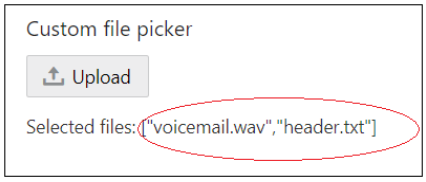
Components to accept the input: Select box



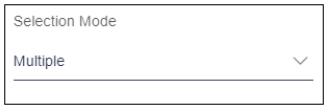
This selection box has two options:

1. Single
2. Multiple

Example: User wants to allow multiple file selection as shown in the image below highlighted in red oval.



Usage: Select type “Multiple” as shown in the image below.



11.18 Enter Allowed File Extensions:

Description: This attribute is used to specify the file extensions that can be uploaded. If not specified, accept all file types.

Component to accept the input: Input text

In the image above, there is information text written under the input text. Example[‘.jpg’, ‘.png’]

All extensions must be in an array, separated by a comma (,), surrounded by single quotes(‘ ‘) and start with full stop (.).

Example: User wants to accept only the following file extension.

“jpg”, “png”, “gif”

Usage: array will be [‘.jpg’,’.png’,’.gif’]

Enter this array in an input box as shown in the image below.

11.19 Image source:

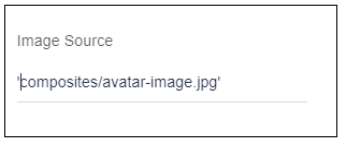
Description: This attribute is used to specify the path or source of an image.

Components to accept the input: Input Text

Example: User wants to add an image, available at “composites/avatar-image.jpg” location.

Usage: Enter this path “composites/avatar-image.jpg” in the input text as shown in the image below.

Note: Use single quotes if path is of type string.



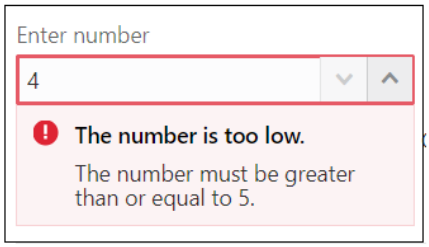
If path is stored in a variable, write that variable name as shown in the image below. Example variable name is “image”.

11.20 Enter Minimum Length:

Description: This attribute is used to specify the minimum allowed value. This number is used in the range validator; if the value is less than the minimum value then the range validator flags an error to the user. The down arrow is disabled when the minimum value is reached.

Component to accept the input: Input Text

Example: User wants to set the minimum value as 5 as shown in the image below.



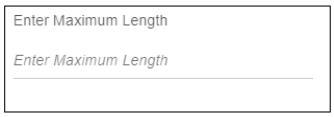
Usage: Enter 5 in the input text as shown in the image below.

Note: Do not write number in quotes.

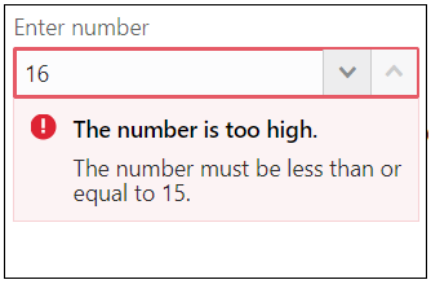
11.21 Enter Maximum Length:

Description: This attribute is used to specify the maximum allowed value. This number is used in the range validator; if the value is greater than the maximum value then the range validator flags an error to the user. The up arrow is disabled when the maximum value is reached.

Component to accept the input: Input Text



Example: User wants to set the maximum value as 15 as shown in the image below.



Usage: Enter 15 in the input text as shown in the image below.

Note: Do not write number in quotes.

11.22 Enter Step:

Description: This attribute is used to specify the size of the step to take when spinning via buttons. Step must be a number greater than 0, otherwise an exception is thrown. It defaults to 1. To understand more about step attribute refer
http://jet.us.oracle.com/6.1.0/jsdocs/oj.ojInputNumber.html#step

Component to accept the input: Input Text

Example: User wants to set the step value as 2.

Value of input number is 5. If user clicks the up arrow, value will increment by 2 and it will be 7 as shown in the image below.

Now value of input number is 7. If user clicks the down arrow, value will decrement by 2 and it will be 5 as shown in the image below.

Usage: Enter 2 in the input text as shown in the image below.

Note: Do not write number in quotes.

11.23 Source variable:

Description: This attribute is used to specify the data source for the list/table/tree. A Data source is a variable, which stores the data that needs to be displayed on the screen in the form of a list/table/tree.

Input Text

Example: User has stored data in a variable named as “deptArray”.

Note: In case of Tree element, this variable must be in JN format.

Usage: Enter this variable in an input box as shown in the image below.

11.24 Id attribute:

Description: This attribute is used to specify the column name that contains the unique key from the data source of the list/table. A Unique key i.e. a column is used to identify an item of list/table uniquely.

Component to accept the input: Input Text

Example: User has a data source deptArray, which has the following data.

Var deptArray= [

{DepartmentId: 10, DepartmentName: 'Administration', LocationId: 200},

{DepartmentId: 20, DepartmentName: 'Marketing', LocationId: 200},

{DepartmentId: 30, DepartmentName: 'Purchasing', LocationId: 200}];

Usage: In this example, the column, which is a unique key, is “DepartmentId”. Because value of every “DepartmentId” is unique.

Enter this column name in an input box as shown in the image below.

11.25 Renderer ID:

Description: This attribute is used to specify the id of a renderer that user have created for list/tree. To understand what is renderer and how to create it refer (link to renderer)

Components to accept the input: Input text

Example: User has created a renderer with the id as “item_template”.

Usage: Enter “item_template” in an input box as shown in the image below.

11.26 **Pagination:**

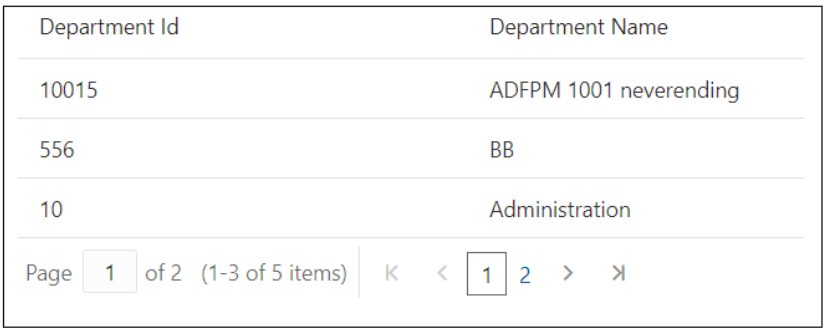
Description: This attribute is used to apply pagination to list/table.

Components to accept the input: Switch

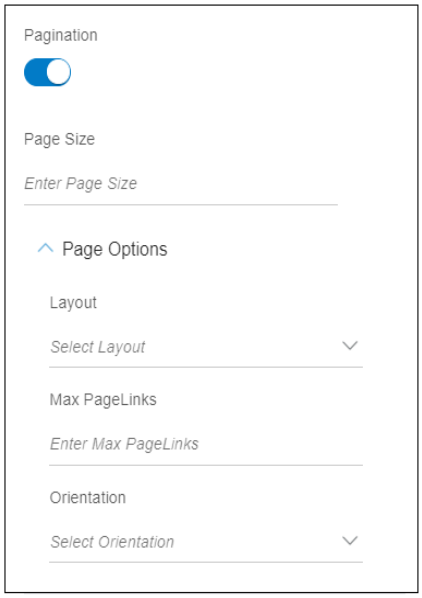
When user enables the switch, it will display an input text and accordion as shown in the image below.

1) Page Size: This is used to specify the page size i.e. how many records should be visible to the user on the first page.

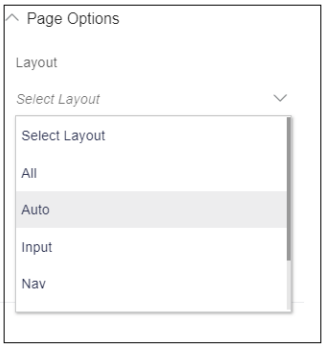
For example, user has given page size as 3, only first three records will be visible to the user as shown in the image below.



2) Page Options: When user expands this accordion, it shows multiple page options as shown in the image below.



3) Layout: This option is used to specify how the paging navigation controls should be displayed.



It has following six options:

a) All: Display all controls

b) Auto: The Paging Control decides which controls to display.

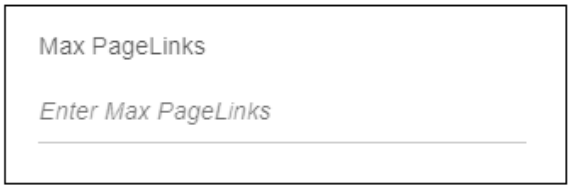
c) Input: Display the page input control

d) Navigation: Display the navigation arrows

e) Pages: Display the page links

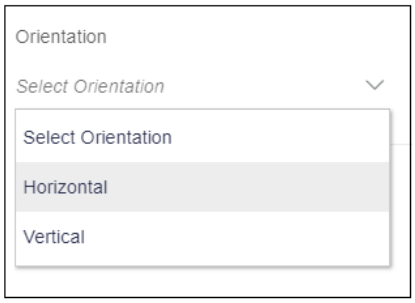
f) RangeText: Display the page range text control

1) Maximum PageLinks: This option is used to give the maximum number of page links to display.



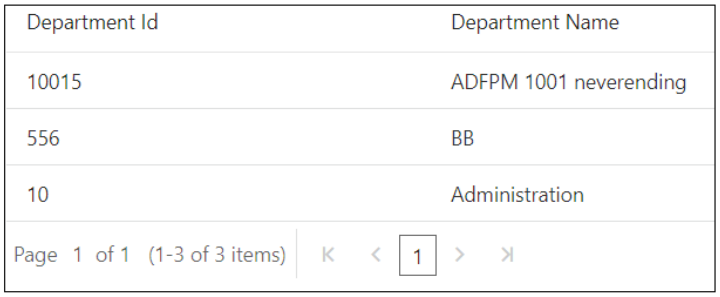
An ellipsis '...' will be displayed for pages, which exceed the maximum number as shown in the image below. maxPageLinks must be greater than 4.

2) Orientation: This option is used to give the orientation of the page links.



It has following options:

a) Horizontal: This option is used to align the page links horizontally.



b) Vertical: This option is used to align the page links vertically.

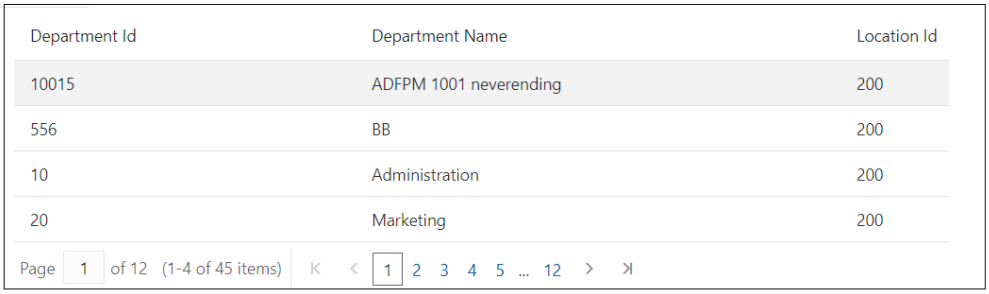
Example: User wants a pagination with following options.

Page size: 4

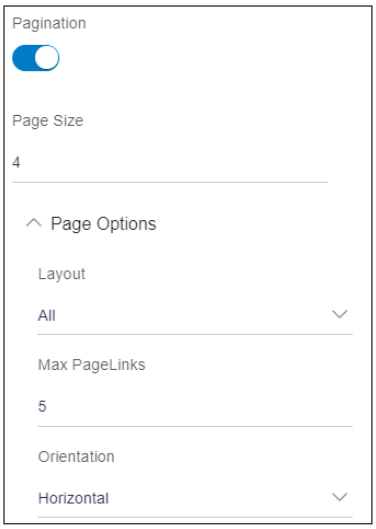
Layout : 'All'

Max Page Links: "5"

Orientation: "Horizontal"



Usage: Enter all the options as shown in the image below.

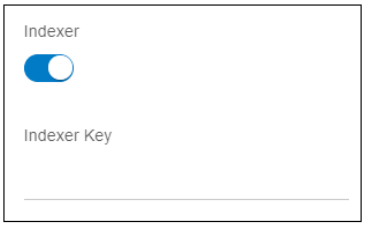


11.27 Indexer:

Description: The JET Indexer is usually associated with a scrollable JET ListView. It provides a list of sections that corresponds to group headers in ListView. When a section is selected, the corresponding group header will be scroll to the top of the ListView.

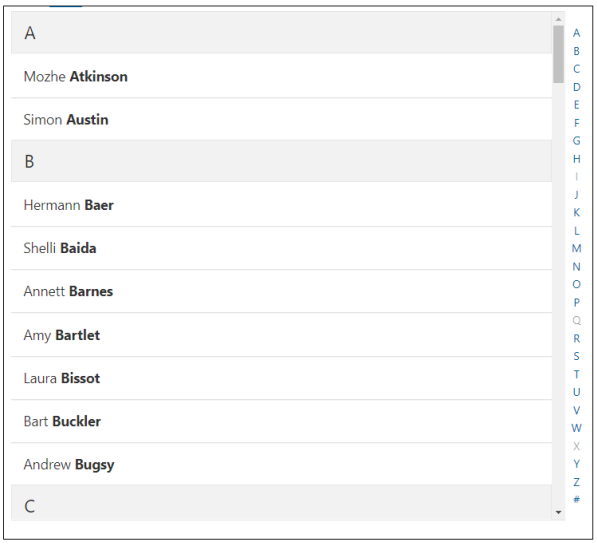
Components to accept the input: Switch

When user enables this switch, it will display an input box as shown in the image below.



Indexer key: This field accepts the key of the data on which grouping is based on.

Example: User wants a list with indexer and user want to group all list items based on the surname as shown in the image below.



Usage: Assume user have following data source of the list.

datasource= [{id: "1", first_name: "", last_name: "Dunphy"},

{id: "100", first_name: "Mozhe", last_name: "Atkinn"},

{id: "101", first_name: "Simon", last_name: "Austin"},

{id: "200", first_name: "Hermann", last_name: "Baer"},

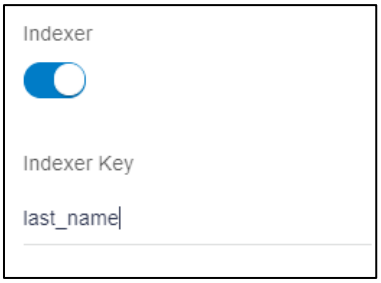
{id: "201", first_name: "Shelli", last_name: "Baida"},

……

{id: "2300", first_name: "Eleni", last_name: "Zlotkey"}]

Here surname is stored in last_name key.

Enable the switch and enter the indexer key as “last_name” as shown in the image below.



11.28 **ID:**

Description: This attribute is used to specify the id attribute for menu. This id attribute is used to launch the menu.

Components to accept the input: Input text

Example: User wants to give the id as “actionMenu”.

Usage: Enter “actionMenu” in an input box as shown in the image below.

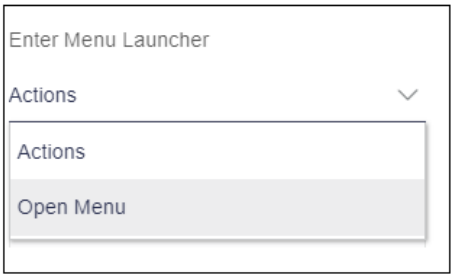
Note: Use single quotes if it is a simple string.

If it is a combination of a string and a variable, for example string is “actionMenu” and variable is “$data.index” then ID will be “ ‘actionMenu’ + $data.index ” as shown in the image below

11.29 **Enter menu launcher:**

Description: This attribute is used to specify the DOM Element, which may or may not be a JET element that launches the menu. For example anchor tag (<a>) or oj-button.

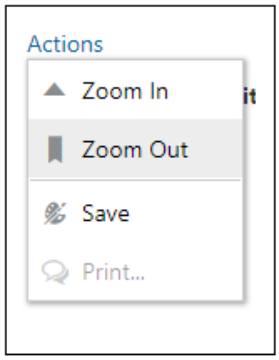
Components to accept the input: Select box



This select box lists the name of all clickable elements that user has added in page. For example, anchor tag and button.

First option “Actions” is a name of the anchor tag and second option “Open Menu” is a name of the button.

Example: User wants to open menu on the click of an anchor tag “Actions” as shown in the image below.



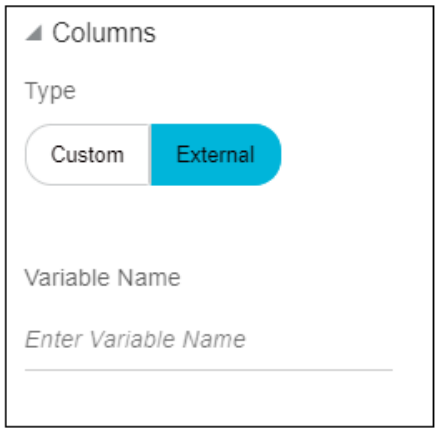
Usage: Select the “Actions” anchor tag option as shown in the image below.

11.30 Columns:

Description: This attribute is used to add the columns of a table.

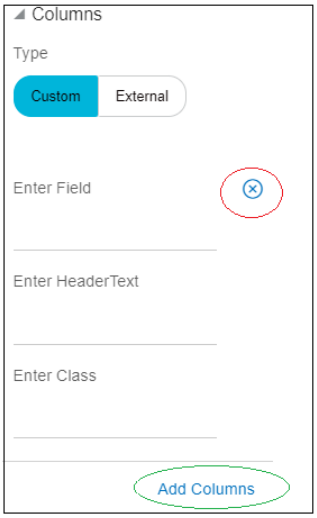
Component to accept the input: Accordion, Buttonset and Input text

When user expands the “Columns” accordion, it shows a buttonset, with custom and external options as shown in the image below.



These are two ways to add columns of the table.

1) Custom: When user clicks this option, it shows three input text as shown in the image below.

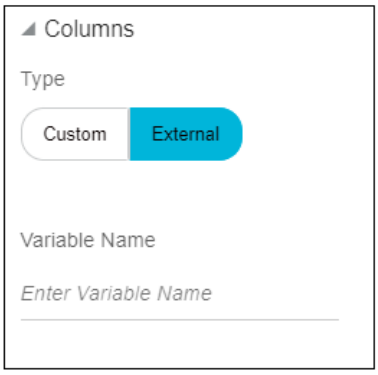


2) Enter Field: This field is used to specify the data field the column refers to i.e. the data of the column.
3) Enter HeaderText: This field is used to specify the text to display in the header of the column i.e. name of the column
4) Enter Class: This field is used to specify the CSS class to apply to the column cells.

User can add more columns using “Add Columns” link highlighted in green oval in the above image.

User can delete the columns using an icon highlighted in red oval in the above image.

5) External: By default, this option will be selected. When user click this option, it shows an input text as shown in the image below.



6) Variable name: This accepts the variable name in which columns information is stored.

Example:

1) Custom: User has a table, which display s Department name and manager ID as shown in the image below.

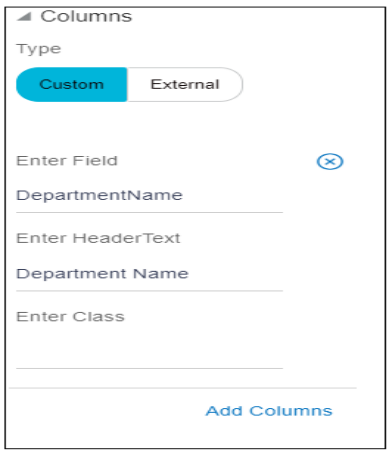
Department Name	Manager Id
ADFPM 1001 neverending	300
BB	300
Administration	300
Marketing	300

Data source of the table is as following.

```
var deptArray = [
 {DepartmentName: 'ADFPM 1001 neverending',ManagerId: 300},
 {DepartmentName: 'BB', ManagerId: 300},
 {DepartmentName: 'Administration', ManagerId: 300},
{DepartmentName: 'Marketing', , ManagerId: 300];
```

a) For the first column i.e. “Department Name”
   - Name of the column is Department Name, headerText is “Department Name”.
   - Data of the column is stored in the field DepartmentName (refer the deptArray), field is “DepartmentName”.
   - No class is needed.

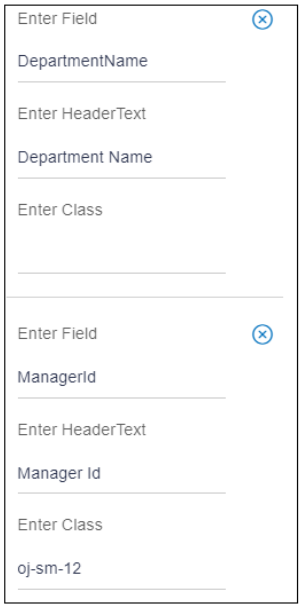
Usage: Enter all these information as shown in the image below.



b) For the second column i.e. “Manager ID”.

- Name of the column is Manager ID, headerText is “Manager ID”.
- Data of the column is stored in the field ManagerId (refer the deptArray), field is “ManagerId”.
- Class is “oj-sm-12”

Usage: To add this column, click “Add columns”. It will add one more column and add all the details as shown in the image below.



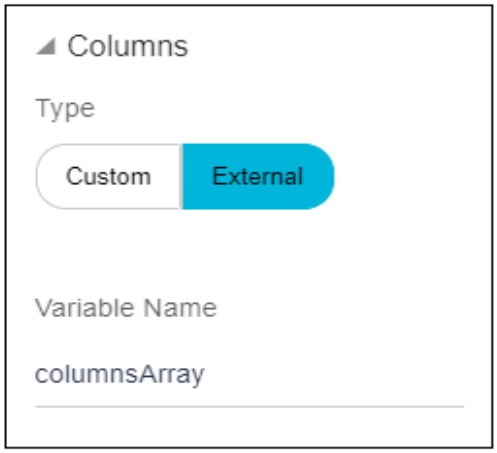
2) External: If user wants to display the same table used in above example.
User wants to make first column resizable, which can be done by adding resizable: enabled configuration to the column. But in custom type there is no option to add this configuration because it accepts only headerText, field and class.
In such cases, where column has more configuration, use “External” type.

Create one variable “columnsArray”. Add two objects for two columns. In those objects, user can add any configuration needed for the column.

```
columnsArray = [ {"headerText": "Department Name",
                  "field": "DepartmentName",
                  "resizable": "enabled"},
                 {"headerText": "Manager Id",
                  "field": "ManagerId",
                  "class": "oj-sm-2"}]
```

As user can see for the first column, “Department Name”, resizable: enabled configuration is added.

Usage: Enter this variable name in an input box as shown in the image below.



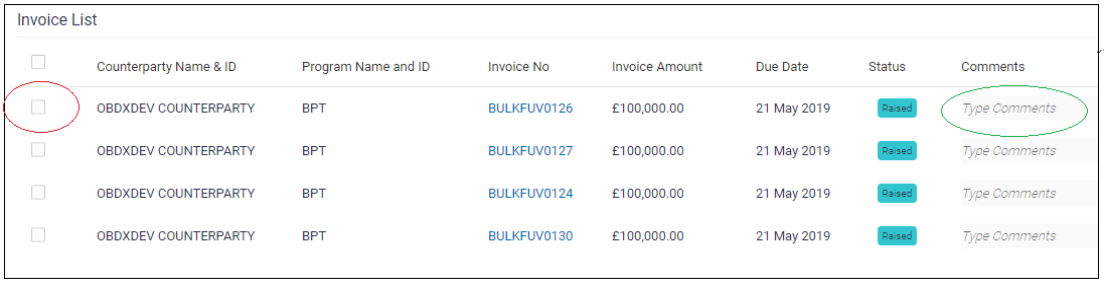
11.31 **Row renderer:**

Description: This attribute is used to specify the id of a renderer for table. This is only valid if user have added any row renderer for the table.

If user has a simple data to display i.e. every column of the table, display some text. In that case, user do not need any row renderer. Refer the image below.

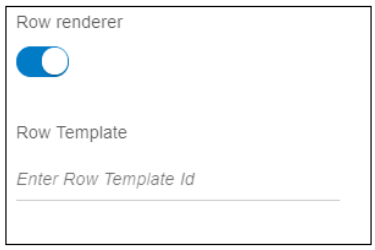
Date	Description	Reference No	Amount	Balance
06 Sep 2018	AT30028200778 NEW DEPOSIT	AT3DEBK1824915AE	£121.00 Dr	£6,160.45
06 Sep 2018	AT30028200745 NEW DEPOSIT	AT3DEBK1824915AA	£221.00 Dr	£6,281.45
06 Sep 2018	Payments and Collections Transaction code	AT3OUPA18249B0QN	£550.00 Dr	£6,502.45
06 Sep 2018	Payments and Collections Transaction code	AT3OUPA18249B0QL	£550.00 Dr	£7,052.45

However, in the table, input fields such as input text, checkboxes or anchor tag have to be added. In that case, user needs a row renderer. For example in the following image, there is checkbox highlighted in red oval and input text highlighted in green oval inside the table. To understand what is renderer and how to create it refer **renderer** section.



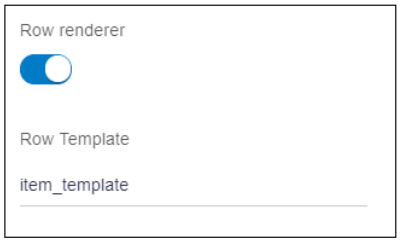
Components to accept the input: Switch, Input box

When user enables the switch, it shows an input box named as “Row template”.



Example: User has created a renderer with the id as “item_template”.

Usage: Enter “item_template” in an input box as shown in the image below.



11.32 Aria label

Description: This attribute is used to specify the aria label of the tag.

Components to accept the input: Input text

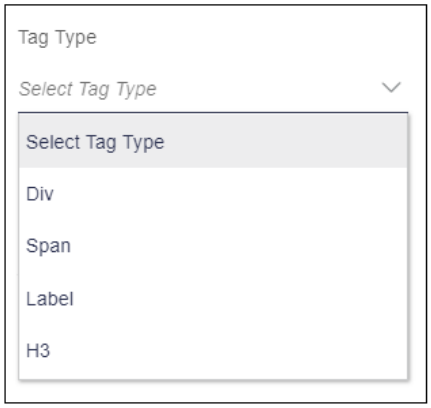
Example: User wants aria label as “Accounts”.

Usage: Enter “Accounts” in an input box as shown in the image below.

11.33 Tag type:

Description: This attribute is used to specify the type of tag.

Components to accept the input: Select box

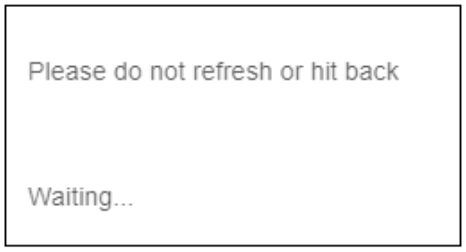


This select box has five options:

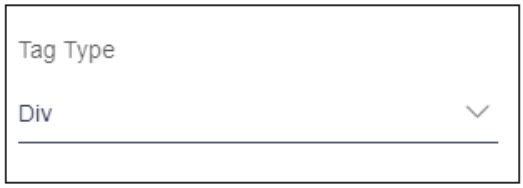
1. Div : It is a block-level element. A block-level element always starts on a new line and takes up the full width available.
2. Span: It is an inline element. An inline element does not start on a new line and only takes up as much width as necessary.
3. Label: It is a label tag of HTML. This element is used to associate a text label with a form input field
4. H3: It is h3 tag of HTML. It represents a level 3 heading in an HTML.

As per requirement, select the type.

Example: User wants do display “Please do not refresh or hit back” text. And user wants other text “Waiting” on the next line as shown in the image below.



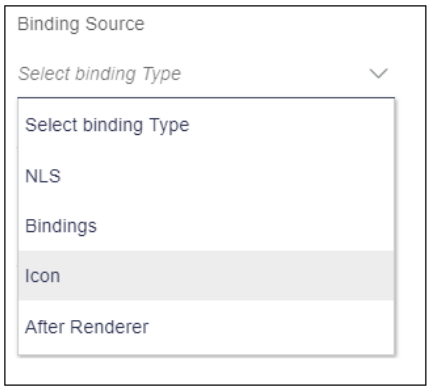
Usage: Now in this example, our first text should occupy the whole line and second text should start on the next line. As Div is a block element, it satisfies the requirement. When user adds “Waiting” text and selects Div type as shown in the image below. “Waiting” text will start on a new line.



11.34 Binding source:

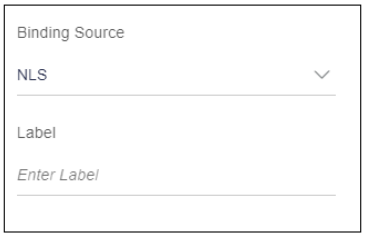
Description: This attribute is used to specify the use of the tag. For example, it can be used to display text, or some value or an icon etc.

Components to accept the input: Select box

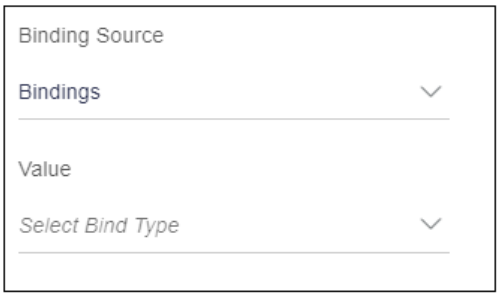


This select box has five options:

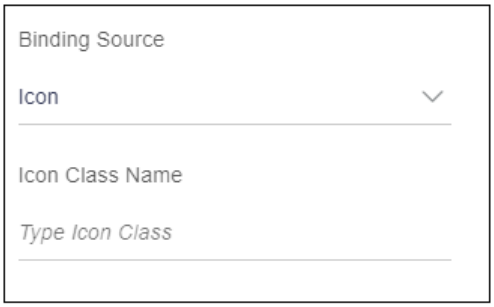
1) NLS: This option is used to display simple text. When user selects this option it will display an input box named as “Label” as shown in the image below.



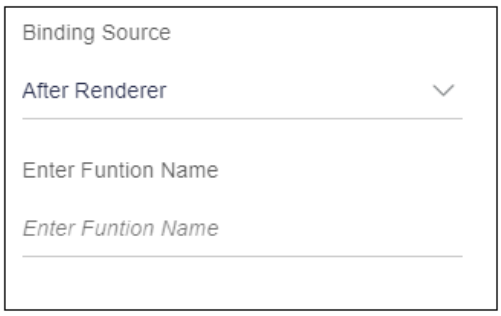
2) Bindings: This option is used to display value of some variable. When user selects this option it will display a select box named as “Value” as shown in the image below. To know more about value refer **Value attribute** section.



3) Icon: This option is used to display an icon. When user selects this option, it will display an input box named as “Icon class name” as shown in the image below.



4) After Renderer: This option is used to call a function after the DOM (Document Object Model) has rendered the specific HTML code. When user selects this option, it will display an input box named as “Enter Function Name” as shown in the image below. To know more about this refer https://knockoutjs.com/documentation/template-binding.html



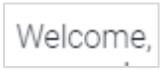
Example:

1) User wants to display simple text “Welcome” as shown in the image below.

Usage: As this is a simple text coming from Resource Bundle, select Type “NLS”.

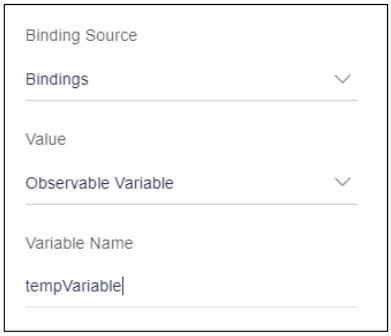
Enter “Welcome” text in the “Label” input box as shown in the image below.

2) User wants to display “Welcome” which is stored in a variable.

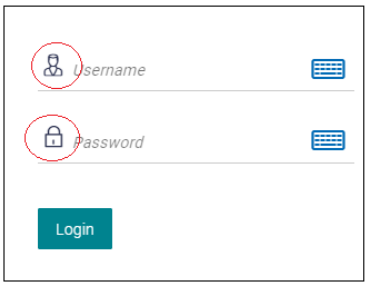


Usage: Assume variable name is “tempVariable”. As this is stored in a variable and not coming from Resource Bundle, select type “Binding Source”.

How to add a variable using Value attribute refer Value attribute section.



3) User wants to display the user icon and password icon, highlighted in red oval in the image below.



Usage: User icon class is “icons icon-user”.

Select type “Icon”

Enter the user icon class “icons icon-user” in the “Icon class name” input box as shown in the image below.

4) User wants to execute some function once the specific elements are rendered on the screen.

Usage: Assume function name is "display Success".

Select type “After renderer”

Enter function name “display Success” in the “Enter Function Name” input box as shown in the image below.

11.35 Enter rows

Description: This attribute is used to specify the number of visible text lines in the text area. It can be used to give specific height to the text area.

Components to accept the input: Input text

Example: User want to add text area with 3 rows as shown in the image below.

In the image above, only first three rows are visible to the user because the number of rows are 3. There is a scroll for other rows as shown in the image below.

Usage: Enter 3 in the input text as shown in the image below.

11.36 Selected step:

Description: This attribute is used to specify the ID of the current selected step. Default is the first step in the steps array.

Components to accept the input: Input text

Example: User wants to set step 2 as a selected step as shown in the image below.

Usage: Assume step array has following data:

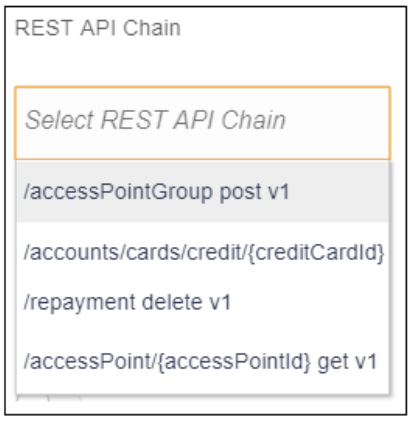
```
Step_array = [{label:'Step One', id:'stp1'},
              {label:'Step Two', id:'stp2'},
```

```
{label:'Step Three', id:'stp3'},
{label:'Step Four', id:'stp4'},
{label:'Step Five', id:'stp5'}];
```

Therefore, ID of second step is “stp2”. Enter this id in the input text as shown in the image below.

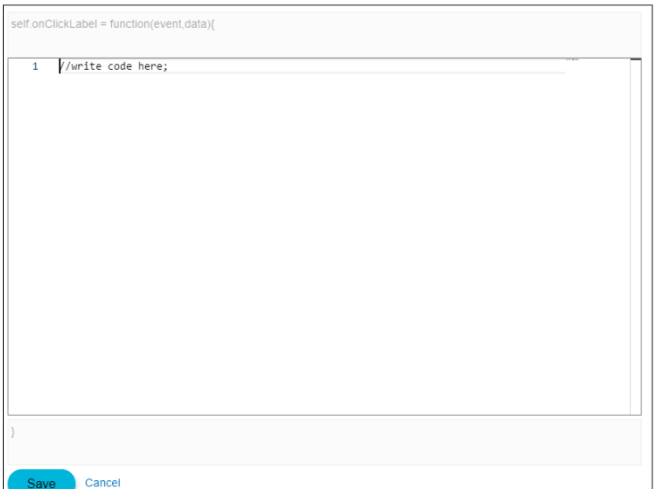
11.37 REST API Chain and Hook Function

- REST API Chain: To understand what is chaining refer **REST API Configuration** section. This select box lists all the REST API chaining that user have created in step four i.e. REST API Configuration. From this select box, user can select REST API chain that user wants to fire when an element is clicked.



- Hook function: There is a button named as “Open Editor”. When user clicks this button, it will open a code editor as shown in the below image. In this editor, user can write the code user wants to execute when the value of an element is changed.

  To understand more about editor refer **Open Editor of hook function** section



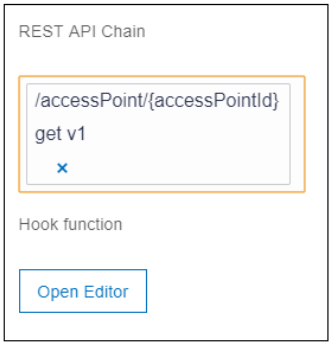
  Example:

1. User want to give an option to use a form as a template. It has an input text to accept the template name as shown in the image below. And it has an anchor tag that checks for the availability of the name entered by the user highlighted in a green oval as shown in the image below.

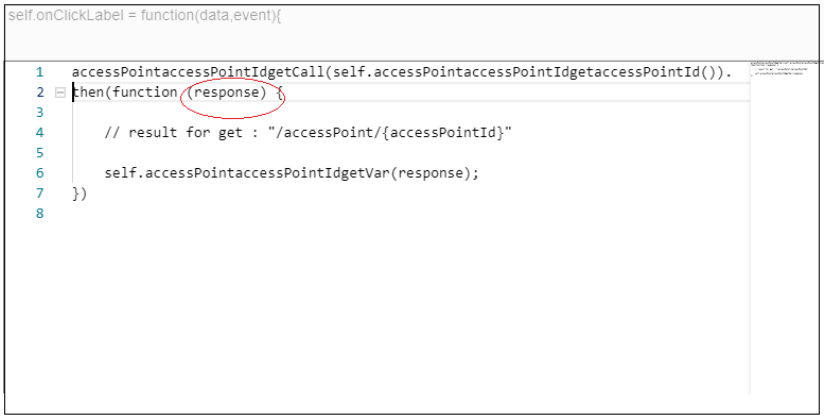
In this example, user needs to fire a rest to check the availability of the name.

Usage:

Select the REST API, user needs to fire when user clicks the Check Availability option as shown in the image below.



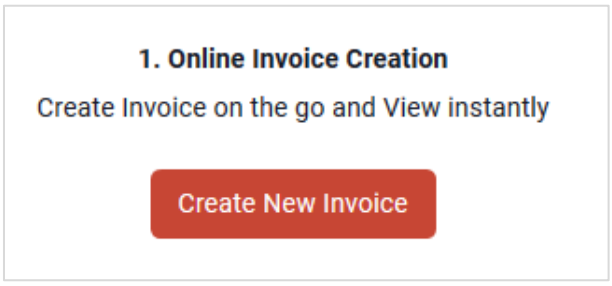
If user click the “Open Editor” button, it will open the code editor.



There is some code already written inside the editor, the tool has generated this code. When user selects any REST API, the tool generates the required code to execute that REST API. However, in some cases, there is need to do some additional operation on the data, fetched from the server. Therefore, for this purpose, there is a support for code editor, to let user write its own code. In such scenarios, use the response (highlighted in red circle in the image above) for further processing.

Whenever user clicks this option, it will execute the block of code written inside the editor.

2. User want to create an invoice. On clicking Create New Invoice button, navigate should be to the form of creating an invoice.



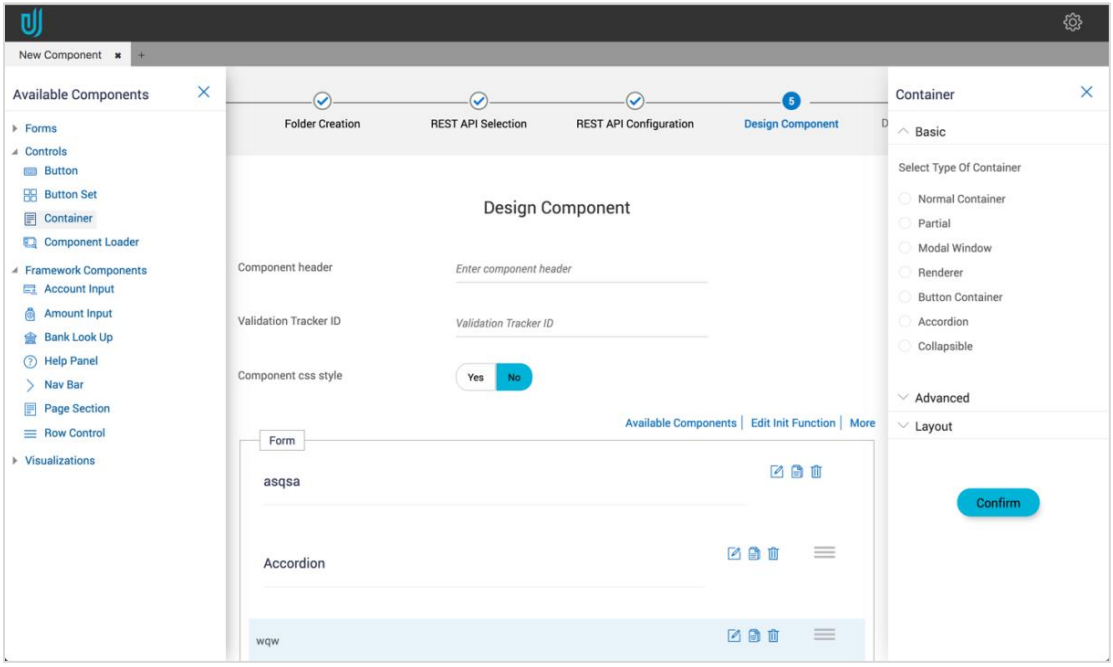
Usage: In this example user do not need to fire any REST API, user will not select any REST API.

Click the “Open Editor” button, and write code inside the editor as shown in the image below.

Function has two parameters event and data. User can use these parameters in code if required.

Note: As there is no REST API selected, there will be no auto-generated code.

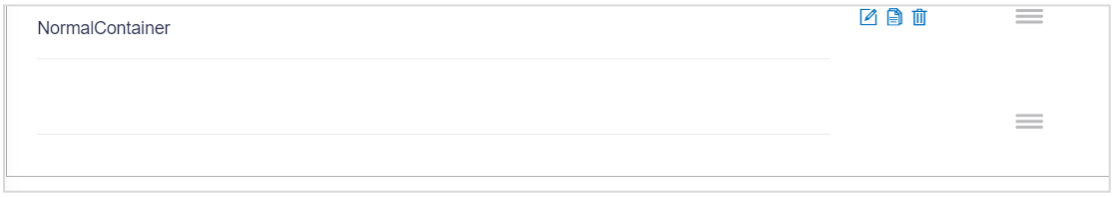
11.38 Select Type of Container



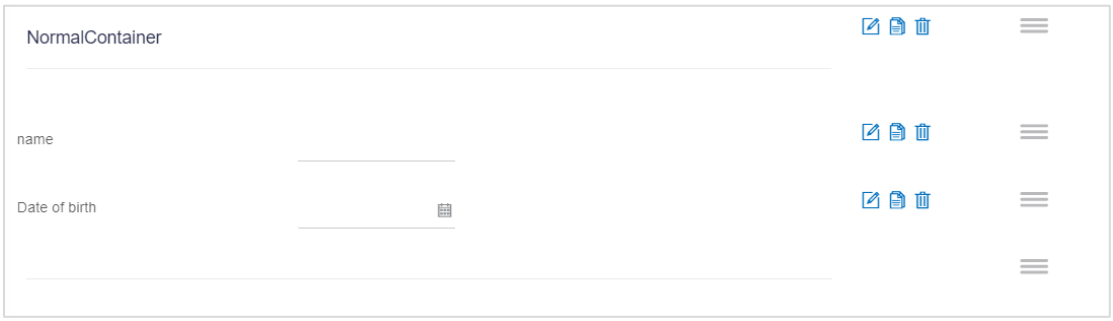
1. **Normal Container:** This is the most basic type of container which is used to group an element to:
   - apply a common style
   - repeat a form section on page
   - hide and display form section
   - apply grid size to a form section

Example:

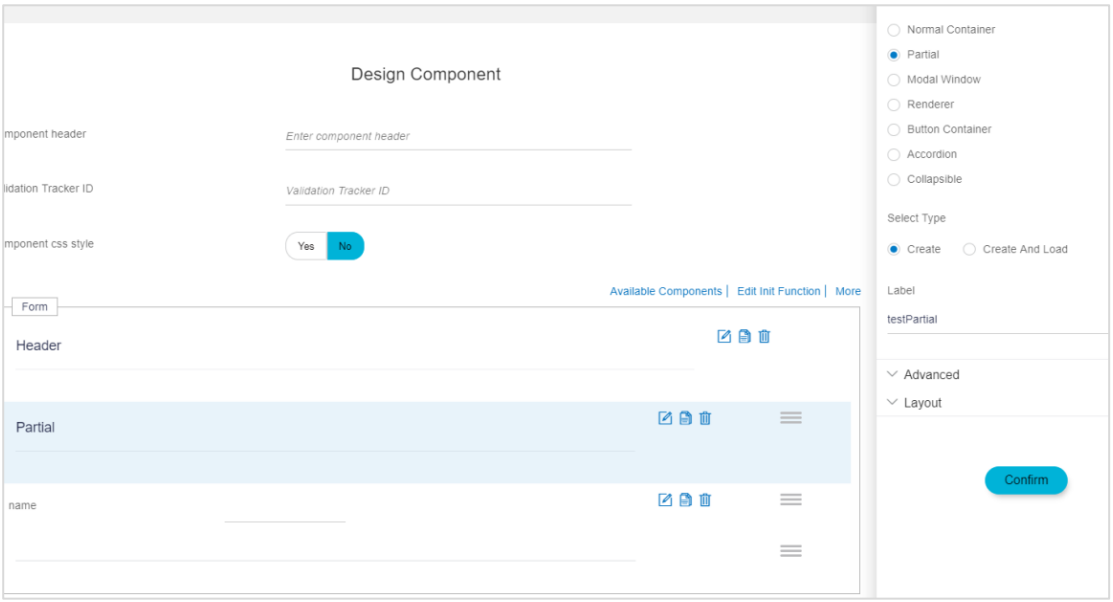
In the below example, normal container option has been selected. Two elements will be added with a gap in between them. User can drop any form element inside it.



In the below image an input box and date picker are added inside a normal container.



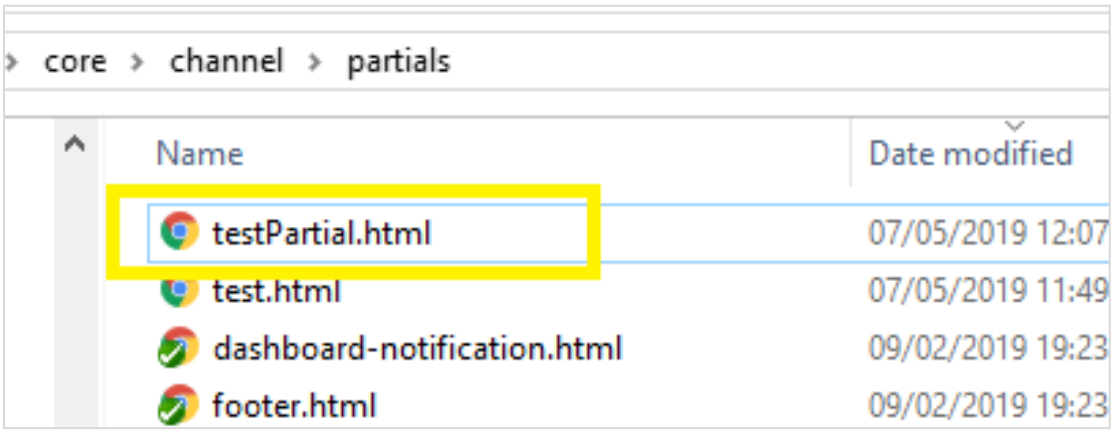
2. **Partial:** Partials are small unit of the page which can be reused.



Example:

When user chooses a container type as partial and create a component, an extra file will be created in partials folder along with other artefacts (refer creating component) at location. ChannelPath/partials/you_module/label.html.

Generated partials:



To use it, refer component loader or select type ‘create and load’ to use is at same location where container is dropped.

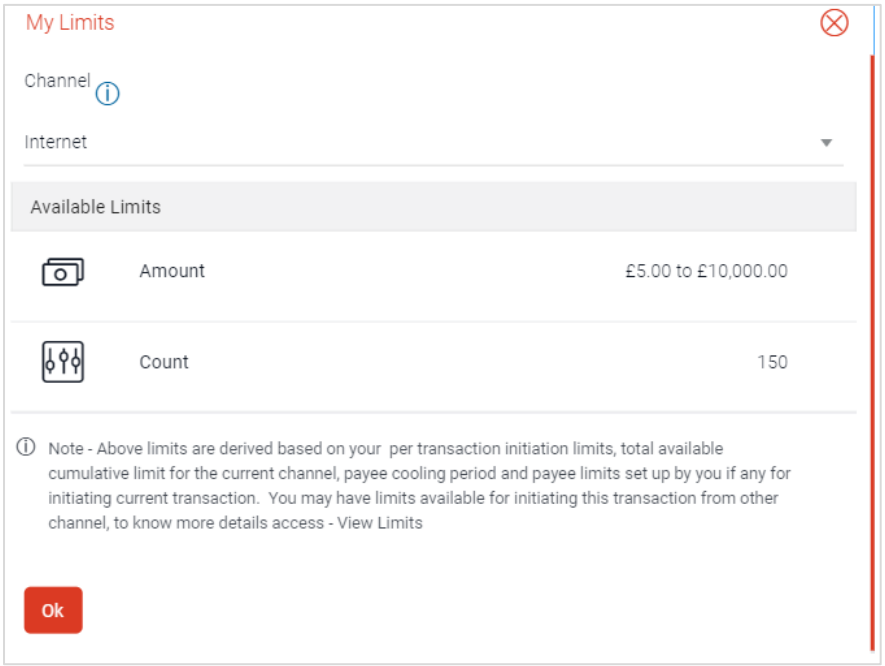
Select Type:

- Create: to create the partial.
- Create And Load: to create and load the partial.

3. Modal Window:

Usage: This container type is used to display the layout in overlay window or dialog box.

Example:



4. Renderer

Usage: Renderer are used to design row of Table or ListView.

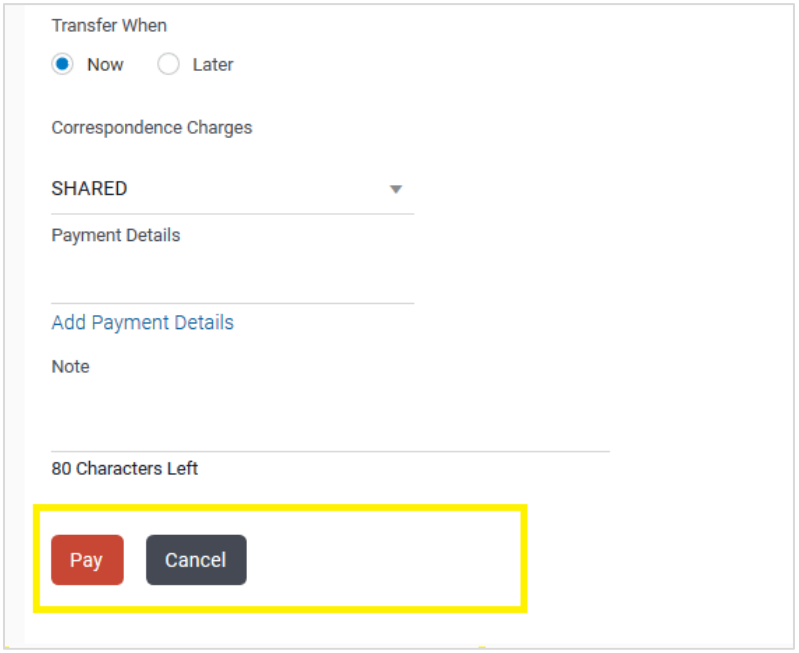
For reference go to http://jet.us.oracle.com/.

- Container ID: This Id will be mapped to Row Template in case of Table or Renderer Id in case of ListView (refer Table and ListView Section).

5. Button Container:

Usage: This container is used to display one or more button as the call to action button. Only buttons can be placed inside this container

Example:

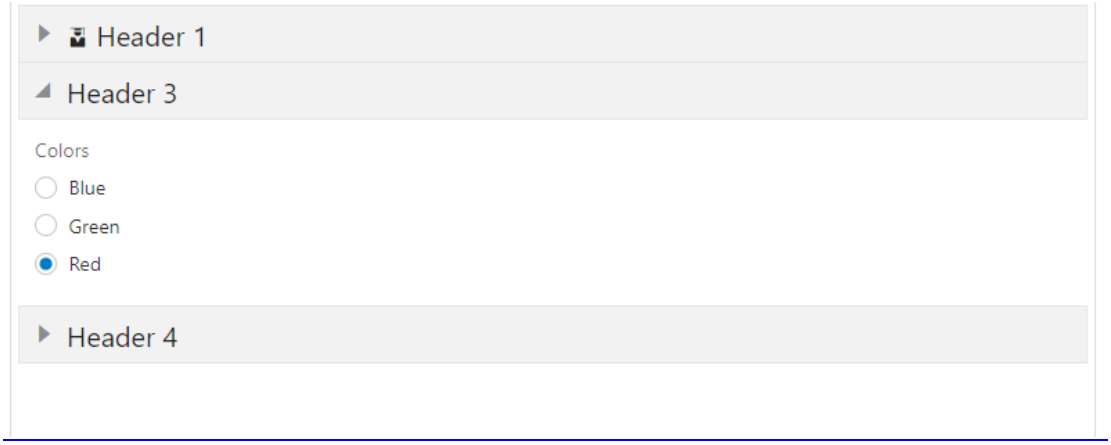


6. Accordion:

Usage: A vertically stacked element which allows the user to toggle between sections of content is an accordion. Accordion container is used to create a group of collapsible items. A collapsible container is dropped inside it. To know about accordion and its use follow the link:

http://jet.us.oracle.com/jetCookbook.html?component=accordion&demo=basicAccordion.

Example:



7. **Collapsible:**

   Usage: Collapsible container can contain any element inside it, which the user uses to toggle between sections of content. To know more about collapsible visit http://jet.us.oracle.com/jetCookbook.html?component=collapsible&demo=basicCollapsible.

Note: UX Extensibility Toolkit and UI Workbench are used interchangeably.