Form Builder as a Service

Administrator’s Guide to the Form Builder

April 2022

Legal Notice

Contents

About this Book 5

1 Introduction 7

About Form Builder. . . . 7
How does the Form Builder Work? . . . 8
Launching the Form Builder . . . 8

2 Exploring the Form Builder User Interface 9

Toolbar . . . 9
Form Components . . . 10
- Basic Components . . . 10
- Advanced . . . 11
- Custom . . . 11
- Layout . . . 11
- Data. . . . 12

Workspace . . . 12
General Settings for Selected Component . . . 12
- Display Tab . . . 12
- Data Tab . . . 13
- Validation Tab. . . . 13
- API Tab . . . 14
- Conditional Tab . . . 14
- Logic Tab . . . 14
- Layout Tab . . . 14

3 Creating and Editing Forms 15

Creating Custom Request and Approval Forms . . . 15
- Creating a Custom Request Form . . . 15
- Creating a Custom Approval Form . . . 16

Customizing the Default Request and Approval Forms . . . 17
Editing a Form . . . 18
- Editing a Form Using the Form Builder User Interface . . . 18
- Editing a Form in the JSON Editor. . . . 18
- Editing a Form in the JS Editor . . . 19

4 Rendering the Forms 23

5 Localization of a Form 25

Providing Translations of Form Field Labels . . . 25

6 Troubleshooting 27

Using setFieldValue function in Logic is Not Supported on Radio Component . . . 27

JavaScript Code Defined in Custom Default Value Not Executed When the Component is Redrawn 27
Unable to Overwrite the Calculated Value on Date/Time Component 28

A Appendix **29**

Example of Creating a Form 29

About this Book

The *Administrator's Guide* provides conceptual information about the Form Builder product. This book describes how to work with the Form Builder product. It also provides step-by-step guidance for creating forms using simple examples.

Intended Audience

This book provides information for individuals responsible for understanding the form building concepts and using the Form Builder to create forms for their application.

Contact Information

Our goal is to provide documentation that meets your needs. If you have suggestions for improvements, please email Documentation-Feedback@netiq.com. We value your input and look forward to hearing from you.

For detailed contact information, see the Support Contact Information website.

For general corporate and product information, see the NetIQ Corporate website.

For interactive conversations with your peers and NetIQ experts, become an active member of our community. The Net IQ online community provides product information, useful links to helpful resources, blogs, and social media channels.

1 Introduction

Forms are an integral part of any application that involves workflow process and data management. Forms can range from simple surveys to complex resource management forms. Typically, forms are embedded in workflows and connect to an API platform at the back end of an application, which can make the overall form creation process complex and difficult. Moreover, integrating forms with third-party servers and legacy systems increases the complexity. Form Builder simplifies the process by making it a hassle-free experience for the application administrators.

The Form Builder allows you to design responsive forms that can be accessed by any application. It provides all the basic and modern form building features, including the drag-and-drop interface, and supports a rich set of user interface elements. Using the Form Builder, you can build, customize, test, and deploy forms to an application. The application uses the Form Renderer component to render these forms and generate corresponding APIs.

The application use forms at different stages of workflow process to collect the required information and to execute relevant action for the fulfillment of the process. For example, roles and permissions workflows often need a formal approval before they are provisioned to the users. The user requests the role or permission by providing the required information through a request form. Once the request is submitted, the approvers can approve or deny the request by providing the required information through approval forms.

About Form Builder

The Form Builder is a web-based service used for designing forms. It provides a platform for application developers and administrators to build their own complex forms. It combines JavaScript forms with REST API Data Management platforms to set up form-based progressive web applications.

Form Builder provides the following features and benefits:

- Drag and drop feature that enables you to quickly create modern and responsive forms.
- Multiple components (widgets) with modern look and feel.
- Simplify connections between the forms and the REST APIs.
- Integrated with the JavaScript Editor to provide a consolidated view of all the JavaScript methods in the form.
- Integrated with the JSON Editor to edit JSON forms directly.
- Support for inbuilt localization for forms.
- Allows you to search and read the REST API description and invoke the REST API in JS methods with a single click. Automatically populates the code in JavaScript Editor to call the API.
- Uses One SSO Provider service (OSP) to provide single sign-on (SSO) across the application and the Form Builder.

How does the Form Builder Work?

Form Builder works as a service that provides form building capabilities to your application. For example, the Form Builder service integrates with the Identity Governance to provide an ability to customize the default forms or to create custom forms. You must be an authorized administrator to build and design forms in Form Builder.

When the form is saved in Form Builder, a JSON schema is generated and stored on the Identity Governance database server. The Form Renderer uses this JSON schema to render the form dynamically within the application. The schema automatically generates the corresponding APIs to pass the user and approver submissions to Identity Governance request and fulfillment workflows.

Launching the Form Builder

Perform the following steps to launch Form Builder:

1. Log in to Identity Governance as a Customer, Global, or Request Administrator, or as Application Owner.
2. (Conditional) Perform the following actions to launch the default request or approval form on Form Builder:
   - **2a** Select **Policy** > **Access Request Polices**.
   - **2b** Select **Application Default Forms** or **Permission Default Forms**.
   - **2c** Choose an application or permission as required.
   - **2d** Click the default request or default approval form to launch the Form Builder in a new browser tab.
3. (Conditional) Perform the following actions to launch the custom request or approval form on Form Builder:
   - **3a** Select **Catalog** > **Applications** > **Application Name**, or select **Catalog** > **Permissions** > **Permission Name**.
   - **3b** Select **Actions** > **Add form set**.
   - **3c** Click **Create Form Set**.
   - **3d** Click the request or approval form to launch the Form Builder in a new browser tab.

2 Exploring the Form Builder User Interface

The Form Builder interface is simple, intuitive, and easy to use. The interface displays the required form components, form component organization, and form component control type. You can simply drag and drop the required form components into the workspace to design new forms. Every component includes validation checks that are executed on both the front end and back end to create a seamless user experience.

Form Builder user interface consists of the following elements:

- Toolbar
- Form Components
- Workspace

Toolbar

You can locate the toolbar on the left pane of the Form Builder user interface. For a better view, click the > button to expand the toolbar and display the available options along with their labels.

The following options are available in the toolbar:

Table 2-1 *Toolbar: Tools and its description*

Tool	Description
Form Builder	Click to return to the Home screen. It is the default view when the Form Builder is launched.
JS Editor	Click to edit the form in the JS editor. The editor enables you to write your logic to change the custom default value and to calculate the value. For more information, see “Editing a Form in the JS Editor” on page 19.
Form JSON	Click to edit the form in the JSON editor. The editor provides a JSON representation describing a fully-featured form. You can also use the editor to duplicate and edit an existing form. For more information, see “Editing a Form in the JSON Editor” on page 18.
Preview	Click to preview the designed form.
Localization	Select the language in which you want the fields in the form to appear in the form renderer. For more information, see Chapter 5, “Localization of a Form,” on page 25.

Tool	Description
Save	Click to save the form.
Settings	Select from the following options: • **Preview Settings:** By default, **Preview Settings** is disabled. Set the toggle button to **ON** to enable the buttons in **Preview** mode. This will allow you to test the buttons such as **Add To Request** and **Submit** in the form. It is recommended to perform this action (Add To Request or Submit) only through the application. • **About:** Displays the version details of the Form Builder.

Form Components

Form components are used as building blocks to create forms. A form component collects user data and serves as the display or user interface within the system. Form components help you to define the type of widget that is required to enter data and automatically adds a property to the resource endpoint to interact with the form component. Each component includes functionality that enables you to design form fields according to your requirements.

Form Builder provides a wide range of components for creating forms. You can use these components to create a variety of forms, ranging from simple survey forms with basic form fields to complex resource management forms with dynamic form fields, where scripts can be executed for validation checks, and actions can be triggered when a form is submitted.

Form components are grouped into the following five broad categories:

- **Basic:** Includes a set of common components that can be used to design simple widgets in a form.
- **Advanced:** Includes a set of components that can be used to design widgets with advanced fields in a form.
- **Custom:** Includes a set of components that allows you to design labels and titles in a form.
- **Layout:** Includes a set of components that can be used to change the general layout of a form.
- **Data:** Includes a set of components that allows you to customize the data layout in a form.

Basic Components

When you select **Basic**, the Form Builder displays the following components:

- Text Field
- Text Area
- Number
- Password
- Checkbox

- Select Boxes
- Select
- Radio
- Button

Advanced

When you select **Advanced**, the Form Builder displays the following components:

- Email
- URL
- Phone Number
- Tags
- Date/Time
- Day
- Time
- Currency
- Survey
- Signature

Custom

When you select **Custom**, the Form Builder displays the following components:

- Label Element
- Title Element

Layout

When you select **Layout**, the Form Builder displays the following components:

- HTML Element
- Content
- Columns
- Field Set
- Panel
- Table
- Tabs
- Well

Data

When you select **Data**, the Form Builder displays the following components:

- Hidden
- Container
- Data Map
- Data Grid
- Edit Grid

Workspace

Workspace is the part of the Form Builder user interface where you drag and drop the components to add new fields into the form. When Form Builder is launched, the default view shows a set of predefined fields in the request or approval form in the workspace area.

General Settings for Selected Component

When you drag and drop a component on the Form Builder interface, a window appears in which you can configure the settings of that component. You can provide a description, select the input format, and enter the required values (which can be predefined as well). These settings, barring certain exceptions, are common for most of the components. For more information about settings, click  in the Form Builder.

The following section describes the settings that are common for most of the components.

Display Tab

The settings available on the **Display** tab define how a given component appears on the form upon rendering.

The following elements are listed on the **Display** tab for a **Text Field** component:

- Label
- Label Position
- Label Width
- Label Margin
- Placeholder
- Description
- Tooltip
- Prefix
- Suffix
- Widget
- Input Mask

- Allow Multiple Masks
- Custom CSS Class
- Tab Index
- Hidden
- Hide Label
- Show Word Counter
- Show Character Counter
- Hide Input
- Initial Focus
- Allow Spellcheck
- Disabled
- Table View
- Modal Edit

Data Tab

The settings available on the **Data** tab allow you to define the default value and how the component appears on the form.

The following elements are listed on the **Data** tab for the **Number** component:

- Multiple Values
- Default Value
- Use Thousands Separator
- Decimal Places
- Require Decimal
- Input Format
- Redraw On
- Clear Value when Hidden
- Custom Default Value
- Calculated Value
- Allow the calculated value to be overridden manually

Validation Tab

The settings available on the **Validation** tab allow you to add validation checks on the component. You can set the component as mandatory to ensure that the user fills it before submitting the form.

The following elements are listed on the **Validation** tab for the **Select** component:

- Validate On
- Required
- Error Label

- Custom Error Message
- Custom Validation
- JSONLogic Validation

API Tab

The settings available on the **API** tab allow you to define the property name and configure any custom properties for the selected component.

The following elements are listed on the **API** tab for the **Phone Number** component:

- Property Name
- Field Tags
- Custom Properties

NOTE: Each component must have a unique Property Name.

Conditional Tab

The settings available on the **Conditional** tab allow you to determine the conditions for displaying or hiding the selected component in a form. You can define a simple conditional logic based on the following rules to determine when to hide or display the component:

- Select `True` to display the component or `False` to hide the component.
- The visibility depends on another component within the same form.
- The logic is activated when the dependent component contains the defined plain text value.

In addition to simple conditional logic, you can also use advanced conditional logic, which allows you to enter custom JavaScript code or custom JSON logic for any combination of conditions.

Advanced conditional logic will override the results of the simple conditional logic.

Logic Tab

The settings available on the **Logic** tab allow you to define and configure multiple logic and actions for the selected component, which helps you design a form that can perform certain defined actions for the defined logic.

Layout Tab

The settings available on the **Layout** tab allow you to define the HTML attributes for the component and map those attributes with the component's input element. You cannot edit the component type from the values you specify in this tab. For example, if you select a **Text Field** component, you cannot change the component type to a different value such as a **Checkbox**.

3 Creating and Editing Forms

Using Form Builder, you can either create custom request and approval forms, or customize the default forms that comes with the application. Let us understand how to perform both these operations in the Form Builder through the Identity Governance application.

Creating Custom Request and Approval Forms

By default, the Identity Governance provides a default form set. This form set consists of request and approval forms with a set of predefined fields. You can use the default forms for requesting access to applications and permissions. However, when more complex forms are required, you can create a new form set and add custom fields in the request and approval forms using the Form Builder. For more information on how to create a new form set, see the Creating Custom Forms for One or More Permissions and Applications in the *Identity Governance User and Administration Guide*.

Creating a Custom Request Form

1. From Identity Governance, launch the request form associated with a permission or application.
2. In Form Builder, drag and drop the required component into the workspace to add new fields on the form. For more information, see "Form Components" on page 10.
3. Define the properties of the component, then click **Save**.
4. Click on the Home screen.

The following example provides step-by-step instructions for creating a request form that allows the user to request for a laptop. The options on the form include the laptop type, color, and the reason for placing the request.

1. From **Basic Components**, drag and drop the **Select** component into the workspace.
2. On the **Display** tab, type `Laptop Type` in the **Label** field.
3. On the **Data** tab, verify that the selected **Data Source Type** is `Values`.
4. In the **Data Source Values**, enter the following values in the **Label** field:
   - Dell
   - Lenovo
   - HP
   - Mac

   TIP: Click **Add Another** button to add new label and value fields.
5. (Optional) Verify the **Laptop Type** drop-down list functions correctly in the **Preview** area.
6. Click **Save**.
7. To add choices for laptop color, drag and drop the **Select Boxes** component into the workspace.

8. On the **Display** tab, type `Choice of Color` in the **Label** field.
9. On the **Data** tab, enter the following labels in the **Values** field:
   - Grey
   - White
   - Black
10. Click **Save**.
11. (Optional) To modify the **Reason** component (present in request form by default), click  and configure the settings as required.
12. Click  to preview the form.
13. Click  on the Home screen.

Creating a Custom Approval Form

When you customize a request form, you may also need to add the corresponding controls to the approval form to facilitate data flow. For example, if you have added **Laptop Type** and **Choice of Color** fields to the "Request for Laptop" request form, it requires that you add these fields in the approval form and configure the flowdata. Adding flowdata allows you to pass the user input from the request form to the approval form.

To add flowdata to the approval form:

1 Launch the approval form associated with the "Request for Laptop" permission in Form Builder.

2 To pass the user input for laptop type from the request form to the approval form, perform the following actions:

2a From **Basic Components**, drag and drop the **Select** component into the workspace.

2b On the **Display** tab, type `Laptop Type` in the **Label** field and select the **Disabled** check box to disable the user input in this field.

2c Click **Save**.

2d Open the JS Editor and look for the following function:

```
function laptopType_CustomDefaultValue () {}
function laptopType_CalculateValue () {}
```

2e Configure the custom default value as:

```
function laptopType_CustomDefaultValue () {value =
context.flowdata.laptopType;}

function laptopType_CalculateValue () {}
```

TIP: You can use the **IG Request** option to add the `context.flowdata.laptopType;` flowdata. Click **IG Request** > **Flowdata**, and select **Laptop Type**.

2f Click  .

3 To pass the user input for choice of color from the request form to the approval form, perform the following actions:

3a From **Basic Components**, drag and drop the **Select Boxes** component into the workspace.

3b On the **Display** tab, type `Choice of Color` in the **Label** field and select the **Disabled** check box to disable the user input in this field.

3c On the **Data** tab, add the following labels in the **Values** field:

- Grey
- White
- Black

3d Click **Save**.

3e Open the JS Editor and look for the following function:

```
function choiceOfColor_CustomDefaultValue () {}
function choiceOfColor_CalculateValue () {}
```

3f Configure the custom default value as:

```
function choiceOfColor_CustomDefaultValue () {value =
context.flowdata.choiceOfColor;}

function choiceOfColor_CalculateValue () {}
```

TIP: You can use the **IG Request** option to add the `context.flowdata.choiceOfColor;` flowdata. Click **IG Request** > **Flowdata** and select **Choice of Color**.

3g Click ⬇.

4 Click 📄 to preview the form.

5 Click ⬇ on the Home screen.

NOTE: ◆You can use the **Simulate Request Workflow** option to review the form fields by simulating the requester and approver action.

- In the Form Builder, you can create inline scripts that can be used as helper functions. Be aware, however, that since these inline scripts are published to the global javascript context, unexpected results may occur. One example of this is in the compare to draft to published area, where one has two forms up at the same time. In this case, both forms will end up sharing the same inline function, even if the definition of the function was different between the draft and published form.

Customizing the Default Request and Approval Forms

Identity Governance uses default request and approval forms for applications and permissions access. These forms are provided under the Application Default Forms and Permission Default Forms tabs in the Access Request Policies page. You can choose to customize the default forms using the

Form Builder. For more information on how to launch the default application or permission form in Form Builder, see Customizing Default Application or Permission Forms in the *Identity Governance User and Administration Guide*.

After the default request or approval form is launched in the Form Builder, you can either modify the existing fields, or you can add new fields to the form. Drag and drop the required component into the workspace, define the properties of the component, then click **Save**. Refer to the "Form Components" on page 10 for more information about available components. After saving the form in Form Builder, you need to publish the form on Identity Governance to set it as the default request or approval form for applications or permissions.

Editing a Form

You can edit a form as follows:

- "Editing a Form Using the Form Builder User Interface" on page 18
- "Editing a Form in the JSON Editor" on page 18
- "Editing a Form in the JS Editor" on page 19

Editing a Form Using the Form Builder User Interface

Each component in Form Builder includes edit, copy, move, paste, and remove functionality. The supporting icons appear when the cursor is positioned over the selected component.

Perform the following actions to edit a component in the form:

1. Click next to the component that you want to edit.
2. Make the necessary edits in the modal window. You can preview the changes in the **Preview** area on the right side of the window.
3. Click **Save**.
4. Click on the Home screen.

NOTE: You can move the components across the workspace in a form. Select next to the component you want to move, drag and drop it to the desired location.

Editing a Form in the JSON Editor

All forms rendered within Form Builder use a JSON schema. When you add new components to a form, you are defining a JSON schema in the background. Form Builder uses this schema to invoke the REST APIs needed to support the form. This section provides an explanation of the structure of the JSON schema and the components that can be rendered within a form.

TIP: Do not directly edit the form in the JSON editor unless you are very comfortable using the editor. You must make a backup of the form before editing it.

The example form described in the "Example of Creating a Form" on page 29 can be designed using the JSON editor as well.

Figure 3-1 *JSON Editor - Sample form*

You can use the same JSON schema to duplicate the form. You can make edits to the form using the JSON editor directly.

Editing a Form in the JS Editor

Form Builder provides a global JS Editor which enables you to add or modify the JavaScript methods for all the form fields in one place. From this page, you can add or modify (invoke the API) the required HTTP method (GET or POST) and apply the value to the required field.

The JS Editor automatically populates the method to set **Custom Default Value** and **Calculated Values** for all the configured fields in the form. You must use this editor to write your own JavaScript logic for these methods.

For example, while setting the **value** for a field, the `value` variable is used by default. However, if you use asynchronous JavaScript, you must use the `instance.setFieldValue(data)` value. The `instance.setFieldValue(value)`function and its parameters are described as follows:

`instance.setFieldValue (response,valueProperty,labelProperty,defaultValueToSet)`, where:

- `response` {Array}: is an array of objects or strings to be set.
- `valueProperty` {String}: is the value for each option to be set. It is expected that each object in the response array has this property.
- `labelProperty` {String}: displays label for the options. It is expected that each object in response array has this property.
- `defaultValueToSet` {String}: when set as string, it selects the corresponding item in the response array that has same `valueProperty` as the `<string>`. This parameter is optional.

The following example shows how to set options or values dynamically for **Select** component:

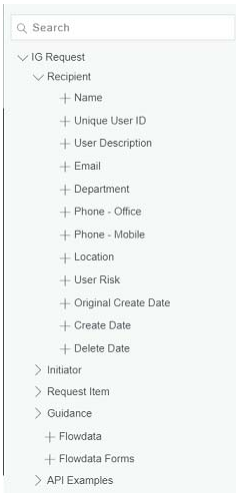
```
function selectForApi_CustomDefaultValue () {
    utils.get('','/api/dcs/schema/GROUP', '', '',
function(response) {
    console.log(response);

instance.setFieldValue(response.attributes,'attributeKey','displayName');
}, function(err) {
console.log(err);
});
}
```

Using IG Request Attributes

The JS Editor provides an option to include Identity Governance attributes in the form. These attributes are passed as `RequestItems` between Form Builder and Identity Governance. Click the **IG Request** option to display the list of supported Identity Governance attributes. You can select the required attribute to design the form accordingly.

A snippet of the supported attributes is shown in the following figure:

Figure 3-2 *IG Request Attributes*



The following section discusses different ways to use the **IG Request** attributes:

- "Using Flowdata Attribute to Pass Data from Request to Approval Form" on page 20
- "Requesting Resource from Identity Governance REST API" on page 21
- "Requesting a Resource From an External API" on page 22

Using Flowdata Attribute to Pass Data from Request to Approval Form

The `flowdata` attribute is used to map a field data in the request form to the corresponding field in the approval form. It facilitates the flow of information from the request to the approval form.

When you add a new field to the request form, you may also need to add the corresponding controls to the approval form to facilitate the data flow. For more information, see "Creating a Custom Approval Form" on page 16.

Requesting Resource from Identity Governance REST API

The **API Examples** option in the JS Editor includes a sample **Calling IGA API** method that shows how to retrieve a resource from Identity Governance REST API. The sample method shows how to fetch information about the logged in user by requesting the `/api/whoami` endpoint.

```
var url = '/api/whoami';
   // get(serviceId, APIUri, body, options, successCallback, errorCallback)
   // serviceId should be left as ''
utils.get('',url, '', {},

  function(response) {

   instance.setFieldValue(response.principal);

  },

  function(err) {

   console.log(err);

});
```

You can edit the endpoint and response in this sample method to retrieve the desired resource from the Identity Governance REST API. For example, if you want to create a drop-down list using the **Select** component in the Form Builder, the options in this list should be fetched dynamically from the Identity Governance REST API. To accomplish this requirement, perform the following actions:

1. In Form Builder, go to **Basic Components**, then drag and drop the **Select** component into the workspace.
2. On the **Display** tab, type `Group Attributes` in the **Label** field.
3. On the **Data** tab, select **Asynchronous API** as **Data Source Type**.
4. Click **Save**.
5. Open the JS Editor and look for the following function:

   ```
   function laptop_CustomDefaultValue () {}
   function laptop_CalculateValue () {}
   ```

6. Configure the custom default value as:

   ```
   function laptop_CustomDefaultValue () {
   utils.get('','/api/dcs/schema/GROUP', '', '',
   function(response) {
   console.log(response);
   instance.setFieldValue(response.attributes,'attributeKey','displayName
   ');
   }, function(err) {
   console.log(err);
   });
   }
   ```

NOTE: The request made to the Identity Governance REST API server will add the OAuth header automatically.

---

7. Click .
8. Click and verify that the options are listed under the **Group Attributes** drop-down list.

Requesting a Resource From an External API

Form components automatically add OAuth headers while making API calls to the REST servers or while invoking the URL method to return a JSON array of data to a form field from an external source. Based on your requirement, you may want to create a drop-down list where the options are fetched from an external resource without adding the OAuth header. You can accomplish this requirement in the JS Editor. The **Populating Select list via External API** method available under the **API Examples** provides a sample on how to retrieve a resource from an external API.

You want to create a drop-down list using the **Select** component, where the options in the list are fetched from a file `<filename>.json` which is hosted on AWS. While requesting the file, you want to ensure that the component does not add an OAuth header in the URL. To accomplish this requirement, perform the following actions:

1. In Form Builder, go to **Basic Components**, then drag and drop the **Select** component into the workspace.
2. On the **Display** tab, type `Laptop` in the **Label** field.
3. On the **Data** tab, select `Asynchronous API` as **Data Source Type**.
4. Click **Save**.
5. Open the JS Editor and look for the following function:

```
function laptop_CustomDefaultValue () {}
function laptop_CalculateValue () {}
```

6. Configure the custom default value as:

```
function laptop_CustomDefaultValue () {

var url = 'https://iga-demo.s3.us-east-2.amazonaws.com/api/
laptops.json';

   $.get(url,
    function(response) {
        instance.setFieldValue(response.laptops, 'value', 'label');

        // let us assume this is on an approval form, and
context.flowdata.laptop contains the value from the request screen
        if (context.flowdata && context.flowdata.laptop) {
           instance.setValue(context.flowdata.laptop);
        }
    });
}

function laptop_CalculateValue () {}
```

7. Click .
8. Click and verify that the options are listed correctly under the **Laptop** drop-down list.

4 Rendering the Forms

Forms created in the Form Builder are rendered on the application through a Form Renderer. The Form Renderer uses the form JSON schema to render the forms dynamically on the front-end of the application. The schema automatically generates the corresponding APIs to receive the data when the form is submitted. Form Renderer uses custom CSS styles specific to the application to render the default look and feel of the form.

5 Localization of a Form

The purpose and usage of forms created using the Form Builder varies with the audience and the application where it is embedded. Form Builder offers the capability of translating forms into the language of your choice. The built-in **Localization** option provides multiple language support for translating forms. You need to provide the translated values for the label key and value in the preferred language, and the resulting form will be rendered on your application in the language set on the browser.

The following languages are supported:

- Chinese (Taiwan)
- Chinese (China)
- Swedish
- Russian
- Portuguese
- Polish
- Dutch
- Norwegian
- Japanese
- Italian
- French
- Spanish
- German
- Danish
- English

Providing Translations of Form Field Labels

To set up the labels and values for different languages, perform the following actions:

1. After saving the final form, click in Form Builder.
2. Select the language of your choice.
3. Specify the values for the label keys in the selected language.
4. Click . The form fields appear with the relevant changes in the option.

NOTE: When creating or editing a form in Form Builder, some field and button labels in the Editing pane appear in English, rather than language selected for localization. The completed form will be correctly localized, however. You can click  on the left navigation to verify that the form is correctly localized.

6 Troubleshooting

The following section contains information about troubleshooting the issues when the Form Builder or its component may not function as intended.

Using setFieldValue function in Logic is Not Supported on Radio Component

Issue: If you set logic on the **Radio** component using the `setFieldValue` function to fetch values dynamically when triggered by an event, the radio field might not work as expected.

Workaround: You can use the `setFieldValue` function in either `CustomDefaultValue` or `CalculateValue` method in the JS Editor to fetch values dynamically on a radio field. For example, you can configure the following function on the **Radio** component and set the default value as 1:

```
function radio_CustomDefaultValue () {
    setTimeout(()=>{
        instance.setFieldValue(fetchValuesForRadio(),null,null,1);

    },1000)
}

function radio_CalculateValue () {}

function fetchValuesForRadio(){
    return ['1','2''3','4'];
}
```

JavaScript Code Defined in Custom Default Value Not Executed When the Component is Redrawn

Issue: In Form Builder, if you configure two components, for example, a **Number** and a **Text Field**, where you define a JavaScript code to set the values on the **Number** component dynamically and select to **Redraw On** whenever there is change in the **Text Field**, then the **Number** component will not reset when you enter a value in the **Text Field** component. This issue occurs because the JavaScript code defined in the **Custom Default Value** is not executed when the **Number** component is redrawn.

Workaround: There is no workaround at this time.

Unable to Overwrite the Calculated Value on Date/Time Component

Issue: On **Date/Time** component, when you define a JavaScript code using `moment()` function to calculate the date and time based on user input in the **Number** component, the value is calculated as expected. However, if you select the **Allow the calculated value to be overridden manually** check box, you will not be able to change the calculated value in form preview mode. This issue occurs because the time is updated continuously on the **Date/Time** component, which does not allow the calculated value to be overwritten manually.

Workaround: You can resolve this issue by using the `startOf('day')` or `endOf('day')` function in the code to ensure that the set time does not change when the value is calculated on the **Date/Time** component. Perform the following actions:

1. From the **Basic Components**, drag and drop the **Number** component into the workspace, then provide a label in the **Label** field and click **Save**.
2. From the **Advanced** components, drag and drop the **Date/Time** component into the workspace.
3. (Conditional) To set the time to 12:00 AM (start of day), click the **Data** tab and provide the following JavaScript code in the **Calculated Value** field:

   ```
   value = moment().startOf('day').add(data.number, 'days')
   ```

4. (Conditional) To set the time to 11:59 PM (end of day), click the **Data** tab and provide the following JavaScript code in the **Calculated Value** field:

   ```
   value = moment().endOf('day').add(data.number, 'days')
   ```

5. Select the **Allow the calculated value to be overridden manually** check box.
6. Click **Save**.

A Appendix

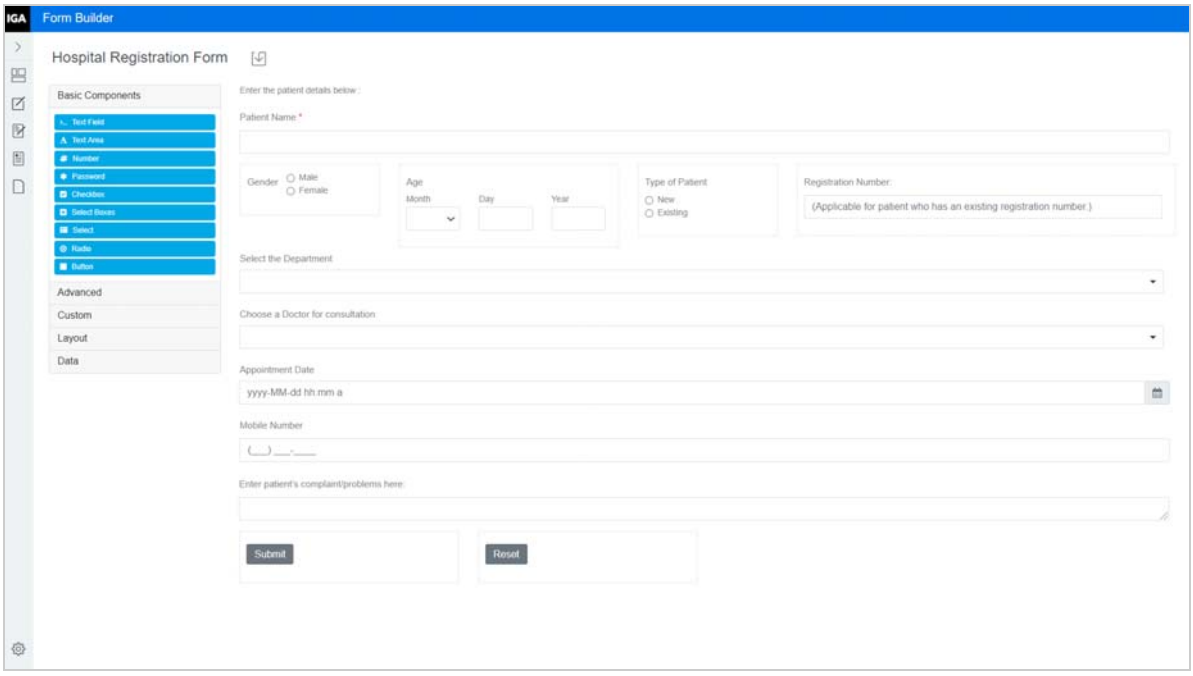
The following section provide additional reference information and advanced topics for the Form Builder.

Example of Creating a Form

This section provides details about creating a form using an example. The following example explains the process of creating a hospital registration form that patients use for requesting an appointment with a Doctor.

The approval form can be created using the instructions mentioned in “Creating a Custom Approval Form” on page 16.

Let us discuss the widgets required to design the following hospital registration form.



Select the required widgets, drag and drop them into the workspace, and configure the settings. The following table lists the components and settings used to create the registration form:

Field Name	Component/Widget Used	Settings	Reference
Enter the patient details below	**Label Element** (Custom) Used to add a non-editable, information-only field in the form.	◆ On the **Display** tab, type `Enter the patient details below:` in the **Label** field. ◆ Select the **Hide Colon** check box to hide the colon symbol next to the label.	For more information, see Label Element.
Patient Name	**Text Field** (Basic) Used to provide a text input field in the form.	◆ On the **Display** tab, type `Patient Name` in the **Label** field. ◆ To mark this component as mandatory in the form, go to the **Validation** tab and select the **Required** check box.	For more information, see Text Field.
-	**Columns** (Layout) Used to split an area in the form into columns. Define the properties for each column, as required. Drag and drop components into each column to add fields in the form.	By default, the **Columns** component splits an area into two columns. On the **Display** tab, click **Add Column** twice to add a third and fourth column. Change the **Width** of the columns depending on the fields to be added. Each of these four columns will be used to add the following fields in the form: ◆ Gender ◆ Age ◆ Type of patient ◆ Registration Number	For more information, see Columns.
Gender	**Radio** (Basic) Used to provide a list of two or more options that are mutually exclusive. The user will be allowed to select only one option from the list.	◆ On the **Display** tab, type `Gender` in the **Label** field. ◆ On the **Data** tab, add the first **Label** as `Male` and the second as `Female`.	For more information, see Radio.
Age	**Day** (Advanced) Used to allow the user to input their date of birth in month, day, and year format.	On the **Display** tab, type `Age` in the **Label** field.	For more information, see Day.

Field Name	Component/Widget Used	Settings	Reference
Type of patient	**Radio** (Basic) Used to provide a list of two or more options that are mutually exclusive. The user will be allowed to select only one option from the list.	◆ On the **Display** tab, type `Type of patient` in the **Label** field. ◆ On the **Data** tab, add the first **Label** as `New` and the second as `Existing`.	For more information, see Radio.
Registration Number	**Number** (Basic) Used to restrict the user to input a numerical value.	◆ On the **Display** tab, type `Registration Number` in the **Label** field. ◆ To add an instruction which will appear when the field is empty, type `(Applicable for patient who has an existing registration number.)` in the **Placeholder** field.	For more information, see Number.
Select the Department	**Select** (Basic) Used to provide values that will display as options in a drop-down list for selection.	◆ On the **Display** tab, type `Select the Department` in the **Label** field. ◆ On the **Data** tab, select the **Data Source Type** as `Values`. Add the following labels in the **Data Source Values** field: 1. `Cardiology` 2. `Neurology` 3. `Oncology` 4. `Obstetrics` 5. `Gynecology`	For more information, see Select.

Field Name	Component/Widget Used	Settings	Reference

Choose a Doctor for Consultation	**Select** (Basic)
	
	Used to provide values that will display as options in a drop-down list for selection.
		• On the **Display** tab, type `Choose a Doctor for Consultation` in the **Label** field.
		• On the **Data** tab, select the **Data Source Type** as `Custom` and provide the following logic in the **Custom Values** field:
		
		`let dept = data.selectTheDepartment;`
		`let choices = getDoctors();`
		`let optionsToSet = [];`
		`for(let i=0; i < choices.length; i++){`
		
		`if(choices[i].department == dept) {`
		`optionsToSet = choices[i].doctors;`
		`break;`
		`}`
		`}`
		`values = optionsToSet;`
			For more information, see Select.
			
			Values populated in **Choose a Doctor for Consultation** field depend on the user selection in the **Select the Department** field. Use the JS Editor to define the logic to calculate the value in this field. For more information, see "Editing the Hospital Registration Form Using JS Editor" on page 33.

Appointment Date	**Date/Time** (Advanced) Used to allow the user to input either a date or a time or both, as desired.	On the **Display** tab, type `Appointment Date` in the **Label** field.	For more information, see Date/Time.
Mobile Number	**Phone Number** (Advanced) Used to allow the user to enter a phone number in the form.	On the **Display** tab, type `Mobile Number` in the **Label** field.	For more information, see Phone Number.
Enter patient's complaint/problems here	**Text Area** (Basic) Used to provide a multi-line input field in the form that enables the user to enter longer text.	• On the **Display** tab, type `Enter patient's complaint/problems here` in the **Label** field. • On the **Validation** tab, add the Maximum Word Length as `200`.	For more information, see Text Area.
Submit	**Button** (Basic) Used to perform actions within the form.	On the **Display** tab, type `Submit` in the **Label** field and select the **Action** as `Submit`.	For more information, see Button.

Field Name	Component/Widget Used	Settings	Reference
Reset	**Button** (Basic) Used to perform actions within the form.	On the **Display** tab, type `Reset` in the **Label** field and select the **Action** as `Reset`.	For more information, see Button.

TIP: Each component has a **Preview** area on the right hand side that displays how the form would render when you make the changes or edits to any field.

You can create or make edits to an existing form using the JSON or JS editor. For more information about using the JSON Editor, see "Editing a Form in the JSON Editor" on page 18.

Editing the Hospital Registration Form Using JS Editor

In the hospital registration form, the **Select the Department** and **Choose a Doctor for Consultation** fields have to be designed in such a way that the options listed in the **Choose a Doctor for Consultation** field depends on what the user selects in the **Select the Department** field. This operation can be achieved using the JS Editor, which enables you to write logic to change the custom default value and to calculate the value of the component.

To define the logic for the Choose a Doctor for Consultation field:

1. Click  in the Form Builder.
2. Define the values to return for the `getDoctors` object as follows:

```
function getDoctors() {
    return [
        {department: 'cardiology', doctors : ['Dr. Adam Hasting','Dr.
Raymond Chase']},
        {department: 'neurology', doctors : ['Dr. Robert Calling','Dr.
Homes']},
        {department: 'oncology', doctors : ['Dr. Steve Lambart','Dr.
Shyama Mukerjee']},
        {department: 'obstetrics', doctors : ['Dr. Daniel Cruise','Dr.
Angelika Sun']},
        {department: 'gynecology', doctors : ['Dr. Katie Homes','Dr.
Ludwig Cunnings']}
    ]
}
```

3. Click  to verify the behavior of the two form fields and preview how the final form would appear after rendering.
4. Click  on the Home screen.