Developer Documentation

- Form and Data Dictionary Development processes
  - Form Design
  - Form Builder Layout Components
  - Cliqon requirements
- Site Design Blocks
  - Design Blocks
- Publishing data
  - Datatables
- Module creation
  - Introduction
  - Preparation
  - Menu entries
  - Programming the Module

Form and Data Dictionary Development processes

Form design and data processes

Form Design

Learn about the Form Components and the settings available within the Form Components.

A Form component collects user data and serves as the display or user interface within the system. Form components define the type of widget that users will enter their data and will automatically add a property to the resource endpoint to interact with the Form component.

Adding a Form Component

To add a form component to a form, drag and drop the component from the left column into the desired location within the form.

Each new form starts with a submit button automatically added to it. This can be removed or edited as necessary.



Drag and drop a component from the left column to any place in the form

Editing a Form Component

To edit a form component on a form, hover over the component and click the **gear** icon. You will then be presented with a settings form for the component.



Click the Edit (gear) button to edit a form component

The settings for a form component are different for each component type.

General Settings

Below is a list of general settings that are offered for the majority of our components.

- **Label - **The name or title for this component

- **Hide Label - **Hide the label of this component. This setting will display the label in the form builder, but hide the label when the form is rendered.
- **Label Position - **Position for the label for this field.
- **Label Width - **The width of label on line in percentages.
- **Label Margin - **The width of label margin on line in percentages.
- **Placeholder - **The placeholder text that will appear when this field is empty.
- **Description - **The Description is text that will appear below the input field.
- **Tooltip - **Adds a tooltip icon to the side of this field.
- **Error Label - **The label that will display for the field when a validation error message is shown.
- **Input Mask -**An input mask helps the user with input by ensuring a predefined format. For a phone number field, the input mask defaults to (999) 999-9999.
- **Allow Multiple masks -**This setting will allow you to set multiple input masks for the field. The mask is selected by the user via a dropdown list and will dynamically switch the mask for the field when selected. This feature is only availabe on our formio.js renderer.
- **Prefix - **The text to show before a field. An example is '$' for money
- **Suffix - **The text to show after a field. An example would be 'lbs' for weight.
- **Custom CSS Class - **A custom CSS class to add to this component. You may add multiple class names separated by a space.
- **Tab Index - **Sets the `tabindex` attribute of this component to override the tab order of the form. See the MDN documentation on `tabindex` for more information on how it works.
- **Multiple Values - **If checked, multiple values can be added in this field. The values will appear as an array in the API and an "Add Another" button will be visible on the field allowing the creation of additional fields for this component.
- **Enable Spell Check - **This setting will enable spell check on the field.
- **Protected - **If checked, this field is for input only. When being queried by the API it will not appear in the properties. You can still see the value on form.io by viewing the form submissions.
- **Persistent - **If checked, the field will be stored in the database. If you want a field to not save, uncheck this box. This is useful for fields like password validation that shouldn't save.
- **Encrypted - **Encrypt this field on the server. This is two way encryption which is not be suitable for passwords. This setting is only available on the 'Enterprise' project plan.
- **Hidden - **A hidden field is still a part of the form JSON, but is hidden when viewing the form is rendererd.
- **Initial Focus**
- Make this field the initially focused element on this form when renderered. Only one component on this form or wizard page can carry the Initial Focus setting.
- **Hide Input - **Hide the input when viewing the form from the front end browser. This does not encrypt on the server. Do not use for passwords.
- **Disabled - **Disable this field on the form.
- **Show Label in DataGrid- **Show the label of this component when in a datagrid.
- **Table View - **If checked, this value will show up in the table view of the submissions list.

Basic Components

Text Field

A Text Field can be used for short and general text input. There are options to define input masks and validations, allowing users to mold information into desired formats.



Widget: You can change widget. The widget is the display UI used to input the value of the field.



Text Area

A Text Area has the same options as the Text Field form component. The difference is that it will be a multi-line input field that allows for longer text. The Text Area can also be utilized as ACE, CKEditor or Quill WYSIWYG editor for the end user which is configured within the component settings.



Rows: This allows control over how many rows are visible in the Text Area.

Editor: This option will turn the text area into the WYSIWYG Editor user interface of your choice.



Editor Settings: Here you can modify the code base of the WYSIWYG Editor to customize the UI specific to your needs.



Number

Number fields can be used whenever a field should be limited to a type of number value. There are options to set Thousands Separator, set Decimal Places and Require Decimals.



- **Use Thousands Separator:** Check this setting if you would like the value in this component to separate thousands by local delimiter.
- **Decimal Places:** The maximum number of decimal places for the values in this field.
- **Require Decimals:** Always show decimals for this field, even if trailing zeros.

Password

The password field has the same options as a text field component. It differs from a text field in that its html `<input>` type will be password instead of text. This will cause the field to display asterisks instead of the value entered.



Check Box

A check box can be used for boolean value input field. It can either be on or off. There are options to set Shortcut and Input Type.



- **Shortcut:** Shortcut can be added to check/uncheck checkbox.
- **Input Type:** Input type can be changed to checkbox or radio.

Select Boxes

The Select Boxes allows the user to check multiple values from list of options.



Values: These are the values that will be selected on this field. The Value column is what will be stored in the database and the Label is what is shown to the users.



- **Inline Layout:** If checked, this field will layout the checkboxes horizontally instead of vertically.



Select

The Select displays a list of values in a dropdown list to users. Users can select one of the values.



- **Widget Type:** You can select the type of widget you would like to use.
- **Data Source Type:** Select the type of data the options will be pulled from.



Values: These are the values that will be selected on this field. You should fill Data Source Values. The Value column is what will be stored in the database and the Label is what is shown to the users.



Raw JSON: Enter a JSON Array into Data Source Raw JSON to use. It should be formatted as an array of objects with named properties.



URL: Enter a url with a data source in JSON Array format. This can be used to populate a Select list with external JSON values. For example, suppose you wish to populate your Select dropdown with a list of all States of the U.S. You can use an external JSON array like the following.

https://cdn.rawgit.com/mshafrir/2646763/raw/states_titlecase.json

Place that in **Data Source URL**. Then you will need to provide the **Value Property** as well as change the **Item Template** so that it will pull the right value as well as display correctly within the dropdown. The following image shows how the configuration would look like for this particular setup.



This will now turn your select dropdown into a list of available States of the U.S.

Value Property: If Raw JSON, URL or Custom is selected as Data Source Type, enter the name of the property for the objects that will contain the value that will be stored in the database. If it is not specified, the item itself will be used.

Search Query Name: If URL or Resourse is selected as Data Source Type, enter the name of the search query parameter to filter requests with. Example, if your url is `http://api.dogs.com/dogs,` and Search Query Name is set to `type`, and then user types `nice` in the Select field, then this component will send a request to `http://api.dogs.com/dogs?type=nice` and update the Select items with the results. If this option is omitted, no new requests will be made when user enters text in the select field.

Item Template: Use the template field to determine how the values will be displayed in the Select dropdown. You can use the **item** variable to access the current object in the array. For example, you can embed the value by using `{{ item.value }}` in a template.

Dynamic Select Filtering

A very common use case that many people have in forms is to dynamically filter a Select dropdown based on the selection of another Select dropdown. The most typical use case is a form that provides the Make, Model and Year of vehicles where when you select the Make dropdown, it filters the Model dropdown for those that are inside that Make. This functionality is covered in detail in our user guide resources section.

Radio

The Radio allows the user to check only one value from list of options.



- **Values:** These are the values that will be selected on this field. The Value column is what will be stored in the database and the Label is what is shown to the users.
- **Inline Layout:** If checked, this field will layout the radio buttons horizontally instead of vertically.



Button

Buttons can be added to perform various actions within the form.



Action: This is the action that will be performed. Currently there are the next actions that can be performed - Submit, Save in State, Event, Custom, Reset, OAuth and POST to URL.



- *Submit:* A submit action submits the form to either the form.io server or a custom callback url that has been priorly set up.
- *Reset:* Reset the form fields back to their original state.
- *OAuth:* Opens an OAuth authentication popup. This will only work after it has been assigned to an OAuth Action. See the OAuth guide for more information on how to set up OAuth in your project.
- **Theme:** Set a theme (color) for the button. These options currently map to Bootstrap classes that will be added to the button.
- **Size:** Set the size of the button. These options currently map to Bootstrap classes that will be added to the button.
- **Left Icon:** If you have an icon library and would like to include an icon to the left of the button label, you can do so by adding the icon class here.
- **Right Icon:** If you have an icon library and would like to include an icon to the right of the button label, you can do so by adding the icon class here.
- **Block:** If checked, the display of the button will be set to “block” which will cause it to span the full width of the container.
- **Disable on Form Invalid:** If checked, this button will be disabled if any of the client side validation fails. This is useful for preventing the submission of a form that has invalid data entered.
- Advanced Components Email
- The email component is nearly identical to the text field component. The Email component has a custom validation setting that, if set up correctly, can ensure the value entered is a valid email address. The email component can also more easily be integrate into a form’s email action. Use this component when you want an email address field for your form.
- **Realtime Email Validation using Kickbox.io**

- In addition to the normal email format validation, we are excited to bring real-time Email validation through our integration with Kickbox.io. For more information on how this works, please checkout the Kickbox Integration section.

URL

The URL component is nearly identical to the text field component. The URL component has a custom validation setting that, if set up correctly, can ensure the value entered is a valid url.



Phone Number

The phone number form component can be used to enter phone numbers in a form.



28 phone number

- A textfield can be used for general text input that is shorter than a sentence. There are options to define input masks and validations so information can be molded into desired formats.
- **Label:** The label for this field that will appear next to it.
- **Placeholder:** The placeholder text that will appear when this field is empty.
- **Input Mask:** An input mask helps the user with input by ensuring a predefined format. For a phone number field, the input mask defaults to (999) 999-9999.
- 9: numeric
- a: alphabetical
- Example telephone mask: (999) 999-9999
- **Prefix - **The text to show before a field. An example is '$' for money
- **Suffix - **The text to show after a field. An example would be 'lbs' for weight.
- **Custom CSS Class - **A custom CSS class to add to this component. You may add multiple class names separated by a space.
- **Tab Index - **Sets the `tabindex` attribute of this component to override the tab order of the form. See the MDN documentation on `tabindex` for more information on how it works.
- **Multiple Values - **If checked, multiple values can be added in this field. The values will appear as an array in the API and an "Add Another" button will be visible on the field.
- **Unique- **If checked, this field will be enforced as unique for this form. Submissions will be checked to see if an existing value matches. This validation is currently server side only.
- **Protected**

- If checked, this field is for input only. When being queried by the API it will not appear in the properties and also should not appear in exported data. You won't be able to see the value on form.io, but it will be stored in database under the hood.
- **Persistent - **If checked, the field will be stored in the database. If you want a field to not save, uncheck this box. This is useful for fields like password validation that shouldn't save.
- **Table View - **If checked, this value will show up in the table view of the submissions list.



29 phone vali

- **Required - **If checked, the field will be required to have a value.

Tags

Multiple or single tags can be used to categorize items. Use the tag component when it's useful for the user.



Address

Important Notice: With recent changes to the Google GeoCode API usage policy, you are now required to provide the Google GeoCode API key along with the configuration for the Address component. Please see the section below on how to setup and configure this within your Google account.

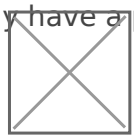
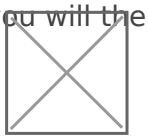
The address form component is a special component that does lookups for addresses entered. It can be entered in free form and will save the address as well as geolocation and other information.



17 address

Google Maps API Key

Due to recent changes to the Google Geocode API usage policies, this is now required for all Address components. To setup an API key, you must follow the following directions.

- First, go to https://cloud.google.com/maps-platform/ and then click on the button that says **Get Started**
- Next, click on the button that says **Places*** and then press **Continue** 
- On the next page, click the dropdown and select **\+ Create new project** (note, if you already have a project, then just select it) and then give your project a name, then click **Next**. 
- Now that you have a project created, you can then setup Billing for that project by clicking on **Set Account** on the billing modal. If you selected an existing project, you will not see this page. 
- If you created a new project, you will then need to provide a payment mechanism so that Google can charge for your usage of their GeoCode API.
- On the next modal, you will then click **Next** to enable the Mapping API's needed for the Address compoennt. 
- The next modal will then show you the API Key that you need to use for this field in the Address component. 

You will then paste this API key into the place on the Address component modal that says **Google Maps API Key**.

- **Label:** The label for this field that will appear next to it.
- **Placeholder:** The placeholder text that will appear when this field is empty.
- **Custom CSS Class:** A custom CSS class to add to this component. You may add multiple class names separated by a space.
- **Tab Index:** Sets the `tabindex` attribute of this component to override the tab order of the form. See the MDN documentation on `tabindex` for more information on how it works.
- **Allow Multiple Addresses:** Allow multiple addresses to be entered into the field.
- **Unique:** If checked, this field will be enforced as unique for this form. Submissions will be checked to see if an existing value matches. This validation is currently server side only.
- **Protected:** If checked, this field is for input only. When being queried by the API it will not appear in the properties and also should not appear in exported data. You won't be able to see the value on form.io, but it will be stored in database under the hood.
- **Persistent:** If checked, the field will be stored in the database. If you want a field to not save, uncheck this box. This is useful for fields like password validation that shouldn't save.
- **Table View:** If checked, this value will show up in the table view of the submissions list.

Required: If checked, the field will be required to have a value.

Date/Time

Date/Time form components can be used to input dates, times or both dates and times.

22 date time

- **Label:** The label for this field that will appear next to it.
- **Default value:** The default value for the date component. You can put `new Date();` for the current date or use a few of Moment.js functions to set the date to a specific date. For example: `moment().add(50, 'days').calendar();` You can use the add or subtract function to go forward or backwards in dates.
- **Placeholder:** The placeholder text that will appear when this field is empty.
- **Date Format:** The format for displaying this field's date. The format must be specified like the AngularJS date filter.
- **Custom CSS Class:** A custom CSS class to add to this component. You may add multiple class names separated by a space.
- **Tab Index:** Sets the `tabindex` attribute of this component to override the tab order of the form. See the MDN documentation on `tabindex` for more information on how it works.
- **Protected:** If checked, this field is for input only. When being queried by the API it will not appear in the properties and also should not appear in exported data. You won't be able to see the value on form.io, but it will be stored in database under the hood.
- **Persistent:** If checked, the field will be stored in the database. If you want a field to not save, uncheck this box. This is useful for fields like password validation that shouldn't save.
- **Table View:** If checked, this value will show up in the table view of the submissions list.



23 date

- **Enable Date Input:** If this is checked, dates can be entered for this field.
- **Initial Mode:** When the date picker appears, this sets the initial view of the picker.
- **Minimum Date:** A value of the date that this field's value must be after.
- **Maximum Date:** A value of the date that this field's value must be before.
- **Starting Day:** On the calendar, select the day of the week that should start the week. In Europe this is typically Monday and in the US this is typically Sunday.
- **Minimum Mode:** Limit the datepicker modes to a minimum of this value.
- **Maximum Mode:** Limit the datepicker modes to a maximum of this value.
- **Maximum Years Displayed:** The maximum number of years to display in the datepicker.

- **Show Week Numbers:** If this is checked, when in Day mode, the datepicker will display the week number on the left hand side of the datepicker.



24 time

- **Enable Time Input:** Allow entering a time as part of this field.
- **Hour Step Size:** The number of hours to increment/decrement in the time picker.
- **Minute Step Size:** The number of minutes to increment/decrement in the time picker.
- **12 Hour Time (AM/PM):** If checked, time will be displayed in 12 hour (am/pm) time. If not checked, it will appear in 24 hour time.
- **Read Only Input:** If checked, users cannot directly enter a time. They can only use the increment/decrement controls to change the time. If unchecked, users can directly enter time values.



26 date time vali

- **Required:** If checked, the field will be required to have a value.
- **Custom Validation:** You can use javascript to perform validation on a field. The way to respond is by setting the `valid` variable. If it is set to `true` then the validation passes. If you set it to a string, the validation fails and the validation message is set to whatever the `valid` variable is set to.
- In addition, `input` variable is set to the value that has been entered in the field. The `component` variable is set to the definition of the field.
- You can also reference other resources and properties for validation. For example, if there is a user resource with a password field, you can use its value with `user.password`

Day

The Day component can be used to enter values for the Day, Month, and Year using a number or select type of field.



- **Type of input:** Type of input for Month, Day and Year can be selected between Number or Select type on Day, Month and Year tabs.
- **Hide Input Labels:** The labels of component Month, Day, Year inputs can be hidden on the form.



- **Day first:** The order of the fields can be changed. The Day field can be displayed before the Month field, if Day first is checked on the Day tab.



- **Minimum Year:** A value of the year that this field’s value must be or after.
- **Maximum Year:** A value of the year that this field’s value must be or before.

Time

The Time component can be used to input time using different time widgets you would like to use.



- **Input Type:** The type of Time widget can be selected between HTML5 Time Input and Text Input with Mask.



Currency

The Currency component should be used when a field should display currency amounts on a form. This component holds a numeric input mask that allows two decimal values and automatically adds commas as a user inputs a currency amount.



currency

- **Label - **The name or title for this component.
- **Placeholder - **The placeholder text that will appear when this field is empty.
- **Prefix - **The text to show before a field. An example is ‘$’ for money
- **Suffix - **The text to show after a field. An example would be ‘lbs’ for weight.
- **Custom CSS Class - **A custom CSS class to add to this component. You may add multiple class names separated by a space.

- **Tab Index - **Sets the tabindex attribute of this component to override the tab order of the form. See the MDN documentation on tabindex for more information on how it works.
- **Multiple Values - **If checked, multiple values can be added in this field. The values will appear as an array in the API and an “Add Another” button will be visible on the field allowing the creation of additional fields for this component.
- **Table View - **If checked, this value will show up in the table view of the submissions list.

Resource

A resource field allows users to reference other resources in your project. For example, if you have a Director resource and a Movie resource, you can add a resource field on the Movie to reference the Director.



30 resource

- **Label - **The label for this field that will appear next to it.
- **Placeholder - **The placeholder text that will appear when this field is empty.
- **Resource - **Select the type of resource to reference. This must be an existing resource within your project.
- **Search Expression - **A regular expression to filter the results with. If you don’t want to show all possible resources within the Resource type selected in **Resource** you can limit which ones will appear in the options by enter a Regular Expression.
- **Select Fields - **Select which properties on the resource to return as part of the options. Select whichever fields you want to display in the template.
- **Search Fields - **A list of search filters based on the fields of the resource. See the Resource.js documentation for the format of these filters.
- **Item Template - **How an item should appear in the list. Use `{% raw %}{{}}{% endraw %}` brackets to reference variables to display. Be sure to use “Select Fields” above to select the fields to display.
- **Custom CSS Class - **A custom CSS class to add to this component. You may add multiple class names separated by a space.
- **Tab Index - **Sets the `tabindex` attribute of this component to override the tab order of the form. See the MDN documentation on `tabindex` for more information on how it works.
- **Allow Multiple Resources - **If checked, more than one value will be allowed to be entered.
- **Table View - **If checked, this value will show up in the table view of the submissions list.

Survey

The Survey component works similar to the radio. Instead of one question, users are able to select a value for multiple questions which are configured within the component settings. Survey is a great component to utilize when asking multiple questions with the same context of answers or values.

- **Label - **The label for this component that will appear next to it.
- **Questions - **The questions you would like to ask within the survey.
- **Values - **The values that can be selected per question. Example: ‘Satisfied’, ‘Very Satisfied’, etc.
- **Custom CSS Class - **A custom CSS class to add to this component. You may add multiple class names separated by a space.
- **Tab Index - **Sets the tabindex attribute of this component to override the tab order of the form. See the MDN documentation on tabindex for more information on how it works.
- **Protected - **If checked, this field is for input only. When being queried by the API it will not appear in the properties and also should not appear in exported data. You won’t be able to see the value on form.io, but it will be stored in database under the hood.
- **Persistent - **If checked, the field will be stored in the database. If you want a field to not save, uncheck this box. This is useful for fields like password validation that shouldn’t save.
- **Table View - **If checked, this value will show up in the table view of the submissions list.
- **Required - **If checked, the field will be required to have every question answered. If it is required, you may not need to persist the value as it can be assumed to be checked when a form was submitted or it will not submit.
- **Custom Validation - **You can use javascript to perform validation on a field. The way to respond is by setting the valid variable. If it is set to true then the validation passes. If you set it to a string, the validation fails and the validation message is set to whatever the valid variable is set to. In addition, input variable is set to the value that has been entered in the field. The component variable is set to the definition of the field.

Signature

A signature field is a special field that allows someone to sign the field with either their finger on a touch enabled device or with the mouse pointer. This signature will be converted into an image and stored with the form submission.



31 signature

- **Footer Label - **You can enter a short instruction here.
- **Width - **How wide the signature area should be. This can be in pixels or percents.
- **Height - **How high the signature area should be. This can be in pixels or percents.

- **Background Color - **The Background color of the signature area. This can be an RGB value in `RGB(255,255,255)` format, a hex value like `#000000` or the name of a color.
- **Pen Color - **The pen color in the signature area. This can be an RGB value in `RGB(255,255,255)` format, a hex value like `#000000` or the name of a color.
- **Custom CSS Class - **A custom CSS class to add to this component. You may add multiple class names separated by a space.
- **Table View -** If checked, this value will show up in the table view of the submissions list.

Nested Form

Nested Form component allows you to insert one (child) Resource/Form with all its fields into another (parent) Resource/Form.

For example, if you have Song resource (parent) and Artist resource (child) and you want to create both Song and Artist submissions by submitting one form, you may use Nested Form component for this case.

- **Name - **The label for this field that will appear next to it.
- **Hide Label - **Check if you want to hide the label when your parent form is rendered
- **Label Position - **Choose where to render the label relatively to the Nested Form component itself
- **Form - **Select a child Form/Resource that you'd like to nest into parent form
- **Custom CSS Class - **CSS class / classes (space separated) that will be added to component's element
- **Save as reference - **Check if you'd like to save only _id of child form submission instead of storing all child submission object inside of parent submission. This will also help all changes you've made in child submission be reflected in parent submission. If not checked, whole child submission object will be saved, and no any child submission changes would be reflected in parent submission.
- **Nested Forms for remotely deployed project - **If your project is remotely deployed with subdirectories, then you need in your app call Formio.setProjectUrl(<project_URL>). This will help Formio understand where to fetch nested resources from and will set up base URLs properly.
- **Component Data- **The Component 'Data' settings will allow you to configure the way your data is saved, rendered, or calculated on your field.
- **Default Value- **Default value will be the value for this field, before user interaction. Having a default value will override the placeholder text.
- **Input Format - **Input Format setting protects from cross-site scripting attacks. Input types default to 'Plain', but 'HTML' as well as 'Raw' can be selected.
- **Database Index - **Set this field as an index within the database which can increase performance for submission queries. *Note - this setting is only available on the Enterprise plan and must have a Submission Collection set in order to activate.*
- **Custom Default Value - **Write JavaScript or JSON to customize the value the field will be defaulted to.
- **Calculated Values - **Caclulated values allow calculating values based on the values in other fields of the form. Calculated values uses plain javascript and can return any value

to the field. There are two variables available to calculate off of, data and row. data is the full data in the form. You can access values within it by using the field keys. For example data.myfieldkey. If you field is within a datagrid, there is an additional variable available of row which contains the values for that row of the datagrid. You can access the values the same as with data as row.myfieldkey. The values are also in data.mydatagrid\[0\].myfieldkey and data.mydatagrid\[1\].myfieldkey plus each additional row as the index.

- Also you have access to special util variable - library of useful functions. More information about util library could be find here.
- Return the calculated value in the value variable and it will be set. Each time the form values change it will be recalculated. You do NOT need to watch form fields as you do for other custom logic in your form. It will automatically update.
- It is very common to disable the field using calculated values.
- **Calculate on Server - **Perform the custom calculations on the server as well as the frontend.

Custom

Custom components allow creating a form field with a custom JSON schema that can be rendered as anything within a frontend application. This is usefully for special or complex form fields that don't already exist in form.io. Using this type, any kind of field can be created.

To use a custom component, create a JSON definition of the field with the information needed to render it. Create a custom field and paste the JSON object into it. This must be a valid JSON object.

{ "type": "custom", "isNew": true, "key": "custom", "protected": false, "persistent": true }

There are several properties that are required but **you may add any additional properties that you would like**.

Type

The type property will be used to select which component to render on the frontend. It cannot be an existing field type.

Key

The key field is where the data will be saved to. This must be unique per field. For example, if `key = 'customers'` then the value of the field will be saved in `data.customers`.

Persistent

This will determine whether or not the value is saved to the main database. This is useful for using Remote Middleware, verify password fields or sending the data in an action but not saving it.

Protected

This will determine whether or not the field will be visible from the API. If it is a protected field then it will only be writeable but not readable.

Rendering

In order to render the custom component, the frontend application must register the component template. This is done in the config step with the formioComponentsProvider.

app.config(\['formioComponentsProvider', function (formioComponentsProvider) { formioComponentsProvider.register('checkmatrix', { title: 'Check Matrix', template: 'formio/components/check-matrix.html', settings: {} }); } \]);

The template will then be used to render the component. In addition, a controller may be added to the template to create more interactive form elements.

This is a working example of a custom component. It is a Matrix checkbox that changes the number of columns and rows based on two other form fields.

reCAPTCHA

reCAPTCHA component implements reCAPTCHA v3.

Type of event

Event to which this reCAPTCHA component would react

Button Key

API Key of the button for reCAPTCHA component to react to (only for Button Click type of event)

Project Settings

If you want to use reCAPTCHA component on your forms, you need to set reCAPTCHA's Site Key and Secret Key in your Project Settings on portal:

4. Go to your project on portal -> Settings -> Integrations -> reCAPTCHA 
5. Specify Site Key and Secret Key with values from your reCAPTCHA Admin panel
6. Save Settings
7. As a result your project will have following in its JSON:

"settings": { "recaptcha": { "secretKey": "your_secret_key", "siteKey": "your_site_key" } }

Render from URL

The easiest way to use reCAPTCHA component is to add it to your form using our form builder and render using its URL. In this case you don’t need to do any additional work, just do following:

Formio.createForm(document.getElementById('formio'), 'your_form_url').then(function(form) { // Provide a default submission. form.submission = { data: { } }; });

Render from JSON

If you want to render form from JSON, you’d also need to do following:

1. Your form should have following settings added to its JSON:

{ "settings": { "recaptcha": { "isEnabled": "true", "siteKey": "your_site_key" } } }

2. Before rendering form you need to set Project URL to the URL of project where you have your reCAPTCHA Secret Key set in settings:

Formio.setProjectUrl('<your_project_URL'); //for ex. https://examples.form.io/

After above is done you can render the form from JSON:

Formio.createForm(document.getElementById('formio'), 'your_form_json').then(function(form) { // Provide a default submission. form.submission = { data: { } }; });

How it works

When you add reCAPTCHA component to the form, you can specify if it’s going to react on Form Load or on Button Click (for Button Click you’ll also need to specify to which button it should react).

Form Load reCAPTCHA

When your form with Form Load reCAPTCHA emits ‘formLoad’ event, we send verification request to Google for action called ‘<your_form_name>Load’. Google’s responce becomes the submission value of your Form Load reCAPTCHA component:

{ "data": { "reCaptcha": { "success": true, "challenge_ts": "2019-01-11T10:29:39Z", "hostname": "your_domain", "score": 0.9, "action": "<your_form_name>Load" } } }

Button Click reCAPTCHA

When any button is clicked on your form, our renderer searches for Button Click reCAPTCHA component tied to a button with same API Key as clicked one. If renderer finds this reCAPTCHA component, we send verification request to Google for action called '<your_button_key>Click'. Google's responce becomes the submission value of your Button Click reCAPTCHA component:

{ "data": { "reCaptcha": { "success": true, "challenge_ts": "2019-01-11T10:29:39Z", "hostname": "your_domain", "score": 0.9, "action": "<your_button_key>Click" } } }

Example

Please see this example of rendering form with reCAPTCHA component.

Component API

Property Name

Form components directly translate to a resource property on the API that is generated for the form. By default a property name is generated by camel casing the field title. You can change the property name by going to the API tab in the form component settings.

Field Tags

Tag the field for use in custom logic or within your application.

Custom Properties

This allows you to configure any custom properties for this component by setting a ‘Key’ and ‘Value’ the property.

Component Layout Settings

In addition to Layout Components, which are detailed in the next section of this guide, you can marginally change the arrangement of the components on your Form within the Layout settings. Each component allows for marginal layout changes from top, bottom, left, and right.

To change the layout, simply input a margin amount in the Top, Right, Bottom, or Left field. Components will be arranged accordingly depending on what margin field you input to. Set margins must be a valid CSS measurement input like “10px” in order to render properly.

layout

Conditional Components

The conditional component requires an API key to be configured to function correctly.

Any form component can use conditional logic to determine when to hide or display itself. The settings for a conditional field, are configured on the component itself, and can be found by viewing the Conditional tab within the components settings.

The conditional logic is based on the following rules:

- Each field can hide or display.
- The visibility is dependent on another component defined within the form.
- The logic is activated when the configured field contains the plaintext value defined in the settings.

In addition to Simple Conditional logic, you can also use Advanced Conditional logic, which uses actual JavaScript for any combination of conditions.

JavaScript conditional logic requires you to set the value of show to either true or false. You have access to the current value of any form component via the data object, and the components API key. Advanced Conditional logic will only work, if Simple Conditional logic isn’t already defined.

When using Advanced Conditional logic with the select boxes form component, you must use the following syntax to get the value of the select box, which will always be true or false, depending on if it is checked or not: data.selectbox_component.selectbox_value

Calculated Value

Caclulated values allow calculating values based on the values in other fields of the form. Calculated values uses plain javascript and can return any value to the field. There are two variables available to calculate off of, *data* and *row*. *data* is the full data in the form. You can access values within it by using the field keys. For example *data.myfieldkey*. If you field is within a datagrid, there is an additional variable available of *row* which contains the values for that row of the datagrid. You can access the values the same as with data as *row.myfieldkey*. The values are also in *data.mydatagrid\[0\].myfieldkey* and *data.mydatagrid\[1\].myfieldkey* plus each additional row as the index.

Also you have access to special util variable - library of useful functions. More information about *util* library could be find here.

Return the calculated value in the *value* variable and it will be set. Each time the form values change it will be recalculated. You do NOT need to watch form fields as you do for other custom logic in your form. It will automatically update.

It is very common to disable the field using calculated values.

Form Builder Layout Components

Layout components are used to change the general layout of forms

HTML Element

An HTML Element component may be added to a form to display a single HTML Element. This is useful if you wish to quickly insert and configure some HTML in your form. All unsafe HTML is stripped before rendering to prevent cross-site scripting exploits. This includes tags like `<script>`, `<embed>`, and `<style>`, and attributes like `onmouseover` or `onload`.

If you wish to insert more complicated HTML in your form, see the Content component



HTML Element settings

HTML Tag: The name of the HTML tag to display.

CSS Class: The CSS class to add to the HTML Element. You may specify multiple classes by separating them with single spaces.

Attributes: Attributes and their values to add to the HTML Element. This is commonly used to add `href` attributes to `<a>` tags, or `src` attributes to `<img>` tags.

Content: The text content of the HTML Element. While adding more child HTML tags here will properly display them, it is recommended you use the Content component to easily write and preview more complex HTML.

Refresh on Change: Makes the HTML Element re-renders whenever any value in the form changes. It might be useful when you want the HTML Element to display dynamic data of the other components after they are filled in with values during the form filling. Simply enter `{{ data.{componentApiKey} }}` into the HTML Element and enable this setting.

Content

A Content component may be added to a form to provide non-field information. For example, if you need instructions at the top of a form that are for display only, use the Content component. The Content component value is **not** submitted back to the server.

A WYSIWYG editor has been provided to help with formatting the content. If you use the HTML view of the editor, note that all unsafe HTML is stripped before rendering to prevent cross-site scripting exploits. This includes tags like `<script>`, `<embed>`, and `<style>`, and attributes like `onmouseover` or `onload`.



Refresh on Change: Makes the Content component re-renders whenever any value in the form changes. It might be useful when you want the Content component to display dynamic data of the other components after they are filled in with values during the form filling. Simply enter `{{ data.{componentApiKey} }}` into the Content component and enable this setting.

Columns

This component can be used for grouping other components, like Text Field, Text Area, Checkbox etc., into configurable columns. It might be useful if you want to display more than one component in one line.



Column Properties: Here you can configure a number of columns that will be displayed in a form and specify the size, width, offset, push and pull settings for each column. When configured, you can easily rearrange the columns using the drag & drop feature without a need to make everything from scratch.

Auto Adjust Columns: If all the nested components inside one of the columns are hidden, all the other columns position will be adjusted.

Field Set

A Field Set can be used to create a title of an area of the form. This is useful to put inside Layout components or in between lots of related components. The Field Set is for display only and will not be saved to the API.



Legend: A Field Set Legend that will appear for the component in the form.

Panel

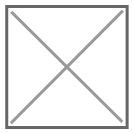
A Panel is used to wrap groups of fields with a title and styling. This currently maps to Bootstrap Card - Header and Footer.



Title: A Panel Title that will be displayed at the top of the Panel.Theme: The theming of the Panel. Select one of the options to have the class added to the wrapper div.Collapsible: Turn the Panel into a collapsible Panel. It might be useful if you want to improve usability of the form that has a big number of components, allowing users to hide the fields they don't want to see while they are filling one or another section of the form.Initially Collapsed: The Panel will be collapsed on form load. Applied only when the Collapsible setting is enabled.

Table

A Table component allows creating a Table with columns and rows that allow adding additional components within it.



Number of Rows: Number of rows that will display in the Table.

Number of Columns: Number of columns that will display in the Table.

Clone Row Components: Clones the components that are in a cell of one of the columns to all the other cells of that column. Might be useful if you want to add a lot of Table rows that will have the same content.

Cell Alignment: Horizontal alignment for cells of the Table. Can be Left, Center and Right.

Striped: Adds striped shading to the Table rows.

Bordered: Adds visible borders for the Table.

Hover: Highlights a row on a mouse hover.

Condensed: Condenses the size of the Table, making it takes less space.

Tabs

This component allows creating tabs for grouping different sets of components. To switch between tabs, there is a navigation bar with tab buttons, each of which opens an appropriate tab with a set group of components. Only one tab at a time displays in a rendered form. This currently maps to Bootstrap Navs



Tabs: A data grid that allows adding, configuring, reordering and removing tabs.

Vertical Layout: Makes the navigation bar display in vertical orientation instead of horizontal, configured by default.

Well

Wells are wrapped in a div with a class. These currently map to Bootstrap Cards.



Cliqon requirements

Use the blank_form.json to start a new form.

Enter the table name (if this is a form to create a new table) as the name or key of the **Panel**

Make sure the field names are added to each field as the name or key for the field

Site Design Blocks

Reference guide for site and page design blocks and strings

Design Blocks

Workspace Link

```
// Use this inside an li tag
    function workspaceLink($lcd, $action, $table, $token) {
        $attrs = "
        class='pointer'
        hx-post='/cmsapi/$lcd/$action/'
        hx-target='#workspace'
        hx-vals='{\"table\": \"$table\", \"token\": \"$token\"}'
        ";
        return $attrs;
    }
```

> ```
> <ul class="links">
>
>         <li @raw( workspaceLink($lcd, 'getdatatable', 'dblinks', $token) )>Useful
>     Links</li>
>
>         <li @raw( workspaceLink($lcd, 'getdatatable', 'dbdocuments', $token)
>     )>Documents</li>
>
>         <li @raw( workspaceLink($lcd, 'getcalendar', 'dbevents', $token)
>     )>Calendar</li>
>
>         <li @raw( workspaceLink($lcd, 'getgallery', 'dbimages', $token)
>     )>Gallery</li>
>
>     </ul>
> ```

Notes

Article Link

```
    // Use this inside an li tag
    function articleLink($lcd, $ref, $token) {
        $attrs = "
        class='pointer'
        hx-post='/cmsapi/$lcd/getdocument/'
```

```
    hx-target='#workspace'
    hx-vals='{\"table\": \"dbdocuments\", \"reference\": \"$ref\", \"token\": \"$token\"}'
    ";
    return $attrs;
}
```

> ```
> 
>             <h6>Services</h6>
>             <ul class="links">
>                   <li @raw( articleLink($lcd, 'ref', $token) )>Web Design</li>
>                   <li @raw( articleLink($lcd, 'ref', $token) )>Web Development</li>
>                   <li @raw( articleLink($lcd, 'ref', $token) )>Content Management</li>
>                   <li @raw( articleLink($lcd, 'ref', $token) )>App Development</li>
>              </ul>
>
>    
> ```

Publishing data

Publish data on the front end website, especially, datatables, datacards, gallery and calendar

Datatables

Initial data

You can either supply initial data through the options object or by starting with a table that already has data filled in.

If you start out with a table that already contains header cells, you can add these attributes to individual header cells to influence how the corresponding column is treated:

- `data-type`: Can be used to set the type of the column. It works the same as using `type` in the columns option.
- `data-hidden`: If set to `"true"` will hide the column. It works the same as using `hidden` in the columns option.
- `data-searchable`: If set to `"false"` will prevent searching of the column. It works the same as using `searchable` in the columns option.
- `data-sortable`: If set to `"false"` will prevent sorting of the column. It works the same as using `sortable` in the columns option.

If you start out with a table that already contains data, you can add these attributes to individual cells to influence how the cell is being processed:

- `data-content`: If this attribute is present on a cell, the value of this attribute rather than the contents of the cell will be processed.
- `data-order`: If this attribute is present on a cell, the value of this attribute will be used for ordering the row in case of sorting. Any other sort order will be overridden. IF the field only contains numeric characters, the field will first be converted to a number.

Module creation

Allows for the creation and management of new modules for the Application

Introduction

General introduction

Creating a new module

Preparation

The things to consider and to have to hand before starting

Menu entries

Administrative menu entries

Start with the menu entries. Example is Accounting module

```
# Ledgers
 [[admenu.topleft.content.submenu]]

    label = '9999:Ledgers'
    title = '9999:Ledgers to which all transactions belong'
    icon = 'landmark'
    id = 'ledgers'
    event = 'actionbutton'
    action = 'fetchadminsubpage'
    table = 'dbledger'
    displaytype = 'datatable'
    url = 'getdatatable'
    acl = 70
    submenu = false
    parent = 'content'

 # Transactions
[[admenu.topleft.content.submenu]]

    label = '9999:Transactions'
    title = '9999:Generates a cash book for a bank account with analysis'
    icon = 'cash-register'
    id = 'transactions'
    event = 'actionbutton'
    action = 'fetchadminsubpage'
    table = 'dbtransaction'
    displaytype = 'cashbook'
    url = 'getcashbook'
    acl = 70
    submenu = false
    parent = 'content'

 # Journals
[[admenu.topleft.content.submenu]]
```

```
label = '9999:Journals'
title = '9999:Generates two sided transactions that must balance'
icon = 'coins'
id = 'journals'
event = 'actionbutton'
action = 'fetchadminsubpage'
table = 'dbtransaction'
displaytype = 'datajournal'
url = 'getdatajournal'
acl = 70
submenu = false
parent = 'content'
```

Things to note:

We will convert 9999's to real entries in Strings and Translations in due course.

Unique ID for each menu entry is advisable. Icons from Fontawesome. Just significant part of free icon name (no need for fa-). Can the data display and entry be satisfactorily handled by existing admin facility or do wee need to write special routine. If the latter, should this be in ApiExtended or as a Plugin?

Programming the module

Programming the Module