# Creating and editing a template

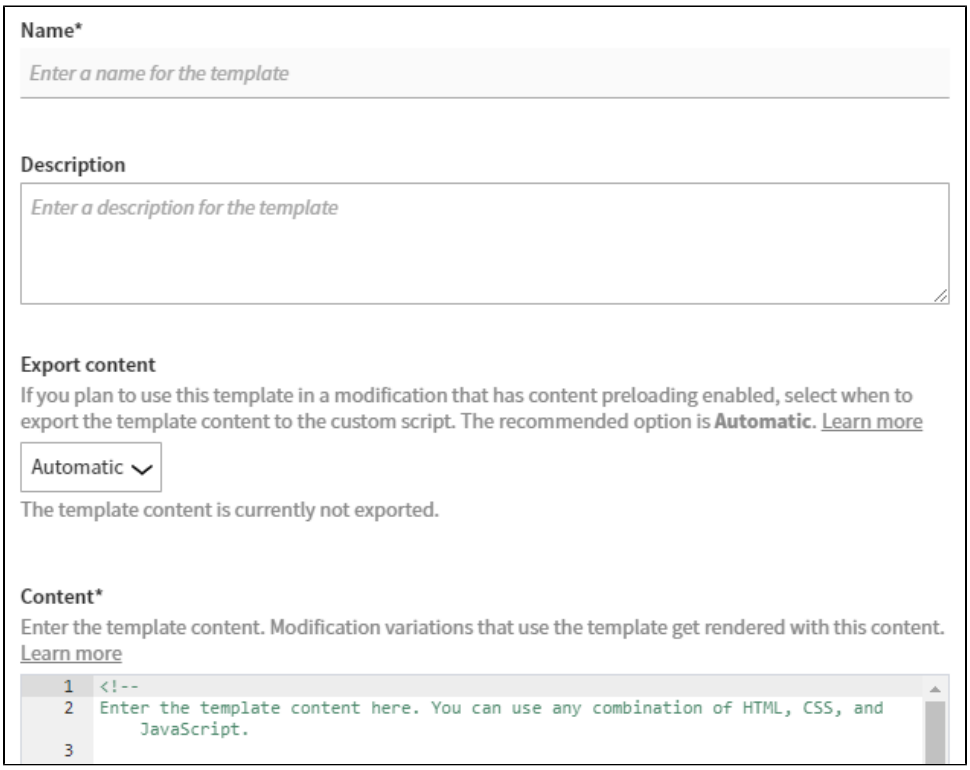
Learn how to create and edit a template, define the settings for the template, and create a content options schema for adding dynamic content to the template.



## Creating a template

To create a template:

1. In the Frosmo Control Panel, select **Modifications > Templates**.
2. Click **Create template**.
3. Define the template settings.




4. When you're done, click **Save**.
5. To return to the templates list, click **Close**.

## Editing a template

> Be careful when editing a template that is in use, since changes to a template affect all modifications that use that template. In particular, if you edit the content of a template that is currently used in an active variation of an active modification, the changes will affect visitors who see the content.

To edit a template:

1. In the Frosmo Control Panel, select **Modifications > Templates**.
2. In the templates list, find the template you want to edit, and click its name.
3. Edit the template settings.
4. When you're done, click **Save**.
5. To return to the templates list, click **Close**.

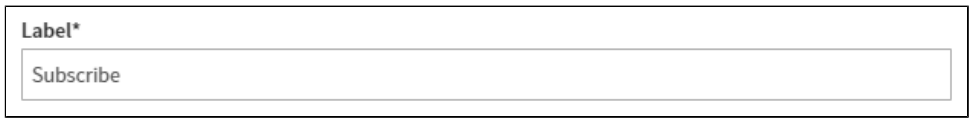
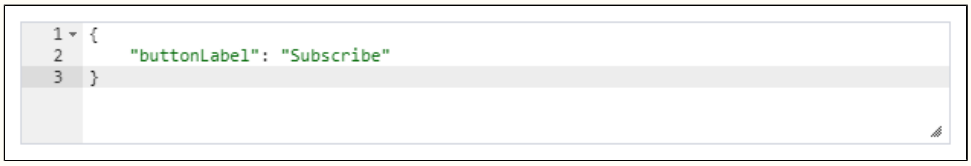
# Template settings

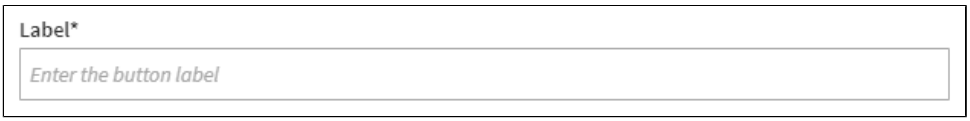
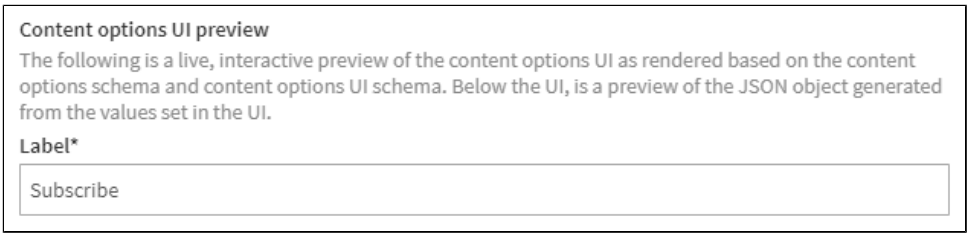
The following table describes the settings you can define for a template in the Control Panel.

**Table: Template settings**

| Setting | Description | Role |
|---|---|---|
| **Name** | Enter a name for the template. | Required |
| **Description** | Enter a description for the template. You can use the description to, for example, explain what sort of content the template renders, and when and how to use the template. | Optional |
| **Export content** | Select when to export the template content to the custom script of the site. Exporting the content means adding it directly to the custom script.<br><br>Modifications that have content preloading enabled require the template content to be available in the custom script. Modifications that do not use content preloading get their content from the Frosmo back end, not from the custom script, so this setting does not affect them.<br><br>By default, this is set to **Automatic**, which means the platform automatically determines whether the template content needs to be added to the custom script. The main reason to exclude the content from the custom script is if the template is currently not actively used in modifications that have content preloading enabled: either the template is not used in any modification variation, or the template is only used in variations that are inactive or whose modifications are inactive.<br><br>Do not change this setting unless you're sure you know what you're doing.<br><br>You can select from the following options:<br><br>• **Automatic**: The platform determines whether the template content needs to be added to the custom script. If the template is used in an active variation of an active modification, and if the modification has content preloading enabled, the platform adds the template content to the custom script. Otherwise, the platform does not add the content. **This is the recommended option**, as it makes the content available only when needed and keeps the custom script size to a minimum.<br>• **Always**: The platform always adds the template content to the custom script, even if the template is not actively used in modifications that have content preloading enabled. This can bloat the custom script and decrease performance.<br>• **Never**: The platform never adds the template content to the custom script, even if the template is actively used in modifications that have content preloading enabled. This option is mainly reserved for templates used in legacy recommendation modifications. **If you want the template to work with regular modifications, do not select this option.**<br><br>The default option is **Automatic**. | Required |
| **Content** | Enter the template content. You can enter any combination of HTML, CSS, and JavaScript.<br><br>When you create a new template, this field contains example code to get you started. You can remove the code if you don't need it.<br><br>You can also use Mustache tags anywhere in the content to create placeholder elements for values. The tags get replaced with specific values in the rendered modification based on the content options you define in the template and map to the tags. You can allow Frosmo Control Panel users to set the content option values from a modification variation, or you can set the values directly in the **Content prerenderer** field. You can also mix these two approaches.<br><br>If you want to separate the structure, presentation, and static parts of the content (HTML, CSS, JavaScript) from the logic that defines the values for the Mustache tags (JavaScript), use the **Content** field for the former and the **Content prerenderer** field for the latter.<br><br>For an example template that uses the **Content** field for all its content code, see Example: Creating a hero banner from a template.<br><br>The following example content creates a button whose label is a Mustache variable with the key `buttonLabel`. To render the variable with a specific value, the template must either define a content options schema that describes the `buttonLabel` content option, which is then set in a modification variation, or define and set the `buttonLabel` content option in the **Content prerenderer** field.<br><br>`<button class="btn btn-subscribe">{{buttonLabel}}</button>` | Required |

| | | |
|---|---|---|
| **Content prerenderer** | Enter the JavaScript code for defining the values for the Mustache tags in the template content and for rendering the content with those values.<br><br>Use this field for implementing the logic that needs to be executed before the content can be displayed on the page. While you can put all your JavaScript in the **Content** field, separating your prerendering logic from your presentation logic (content) makes it easier to manage each.<br><br>If all the Mustache tag values can be defined directly in a modification variation, meaning you do not need to separately retrieve or otherwise build the values, leave this field empty, and instead define a content options schema for the template.<br><br>You can use the following object variables in the code:<br><br>• `easy`: Access Frosmo Platform features and data, such as the site configuration (`easy.config`) and visitor context data (`easy.context`). This is the same object you get when you use `frosmo.easy` in the browser console.<br><br>For more information about debugging modifications and other platform features in the console, see Console debugging with Frosmo Core.<br>• `site`: Access site functions and data defined in the site's custom code. This is the same object you get when you use `frosmo.site` in the browser console.<br><br>`site` is only available on sites that include custom code in their custom script.<br><br>• `templateInstance`: Access the data and functionality for the current template instance:<br>  • `name`: String containing the name of this template.<br>  • `options`: Object containing the content options set in the modification variation.<br>  • `render`: Function for rendering the template content defined in the **Content** field. The function takes an optional object parameter containing the values for the Mustache tags. If the template content does not use any Mustache tags, omit the parameter when calling the function.<br><br>To render the template content for the modification, call `templateInstance.render()`.<br><br>If you use the **Content prerenderer** field, but do not call `templateInstance.render()` in it, the template content is not rendered on the page.<br><br>The following example code takes the `buttonLabel` content option value set in a modification variation, appends the value with an exclamation mark, and renders the template content with the modified value.<br><br>`var newButtonLabel = templateInstance.options.buttonLabel + '!';`<br>`var newOptions = { buttonLabel: newButtonLabel };`<br><br>`templateInstance.render(newOptions);`<br><br>The following figures show the final modification as rendered on the page.<br><br>  <br> | Optional |

| | | |
|---|---|---|
| **Content options schema** | If the template content uses Mustache tags whose values must be set in the modification variation, or if defining the values requires some other input from the variation, enter a content options schema for the template. The Frosmo Control Panel uses the schema to generate and display a content options UI in the variation settings. The UI allows users to set the Mustache tags to specific values for the variation.<br><br>For more information about creating the schema, see Content options schema.<br><br>When you create a new template, this field contains example code to get you started. You can remove the code if you don't need it.<br><br>The following example schema describes a single content option: `buttonLabel`, which takes a string value. In the content options UI of a modification variation, the schema translates into a single, mandatory string field with the title "Label". Entering a value in the field sets the `buttonLabel` content option to that value for the variation.<br><br>`{ "type": "object", "title": "", "properties": { "buttonLabel": { "type": "string", "title": "Label" } }, "required": [ "buttonLabel" ] }`<br><br>The following figures show the content options UI rendered from the preceding schema, with an example value, and the final modification as rendered on the page, respectively. (In this example, the template content uses a `{{buttonLabel}}` variable that gets its value as is from the modification variation.)<br><br>  <br><br>  <br><br>If you do not enter a content options schema for the template, the content options UI of a variation that uses the template is a single text field, which expects a JSON object that defines the values for the Mustache tags. The following figure shows the content options UI and the JSON object for the preceding example when the template does not include a schema. In this case, whomever creates the modification variation must also provide the JSON object.<br><br>  | Optional |

| | | |
|---|---|---|
| **Content options UI schema** | If you want to further customize the content options UI, define a UI schema for it. The UI schema is a JSON object that describes how specifically to render the form input components that make up the content options UI. The UI schema is specific to the `react-jsonschema-form` React component, which the Control Panel uses to build the content options UI.<br><br>When you create a new template, this field contains example code to get you started. You can remove the code if you don't need it.<br><br>For more information about the `react-jsonschema-form` component, see:<br><br>• react-jsonschema-form.readthedocs.io/en/latest for the full component documentation<br>• rjsf-team.github.io/react-jsonschema-form for a live playground environment, where you can generate different types of forms from editable JSON and UI schemas<br><br>The following example UI schema customizes the `buttonLabel` text input component with a placeholder text that is displayed in an empty input field.<br><br>`{ "buttonLabel": { "ui:placeholder": "Enter the button label" } }`<br><br>  | Optional |
| **Content options UI preview** | You can view a live, interactive preview of the content options UI as rendered based on the **Content options schema** and **Content options UI schema**.<br><br>  <br><br>You can also preview the JSON object that gets generated from the values set in the content options UI. The Control Panel saves this object to the modification variation, and Frosmo Core uses the object to replace the Mustache tags in the template content with the actual values to render on the page.<br><br>  | |

Name*

Subscription Button

Description

*Enter a description for the template*

Export content

If you plan to use this template in a modification that has content preloading enabled, select when to export the template content to the custom script. The recommended option is **Automatic**. Learn more

Automatic

The template content is currently not exported.

Content*

Enter the template content. Modification variations that use the template get rendered with this content. Learn more

```
1  <button class="btn btn-subscribe">{{buttonLabel}}</button>
```

Content prerenderer

If the template content requires preprocessing before it can be rendered on a page, enter the JavaScript that implements the preprocessing. Learn more

```
1
```

Content options schema

If the template content must be customized or otherwise configured in a modification variation, define the appropriate content options as a JSON schema. Learn more

```
{
    "type": "object",
    "title": "",
    "properties": {
        "buttonLabel": {
            "type": "string",
            "title": "Label"
        }
    },
    "required": [
        "buttonLabel"
    ]
}
```

Content options UI schema

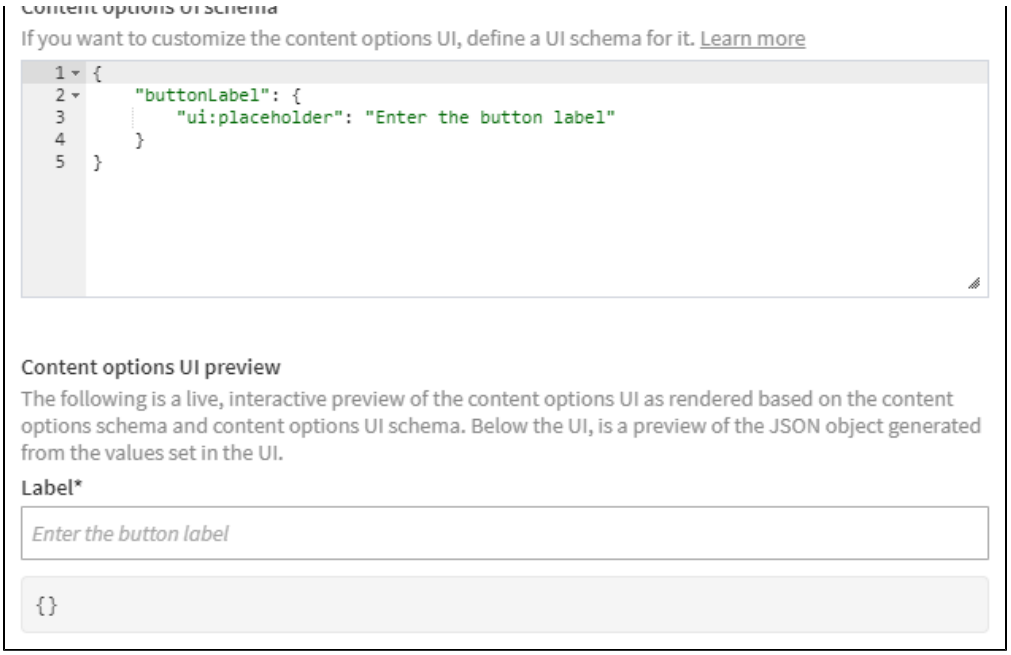



**Figure: Defining the template settings (click to enlarge)**

# Content options schema

Content options allow you to customize template content for individual modifications. While all modifications that use the same template share the same basic content, the template can provide options for customizing that content per modification. The content options are based on the Mustache tags in the template content, and either directly provide the values for those tags or otherwise provide input that is required to define the final values. You define the content options in the template, but you set the options separately in each modification variation that uses the template. The template content together with the content options as set in a variation define what finally gets rendered on the page.

**Example 1:** Template content uses a Mustache variable for a customizable button label. You define a content options schema that describes the label, so that users can enter the desired label text in the modification variation.

**Example 2:** Template content uses a Mustache section for rendering a list of trending products. The product data for the list is fetched in the content prerenderer. However, fetching the data requires as input the number of products to list. You define a content options schema for the number, so that users can select the desired number of products in the modification variation, which then gets passed as input to the prerenderer.

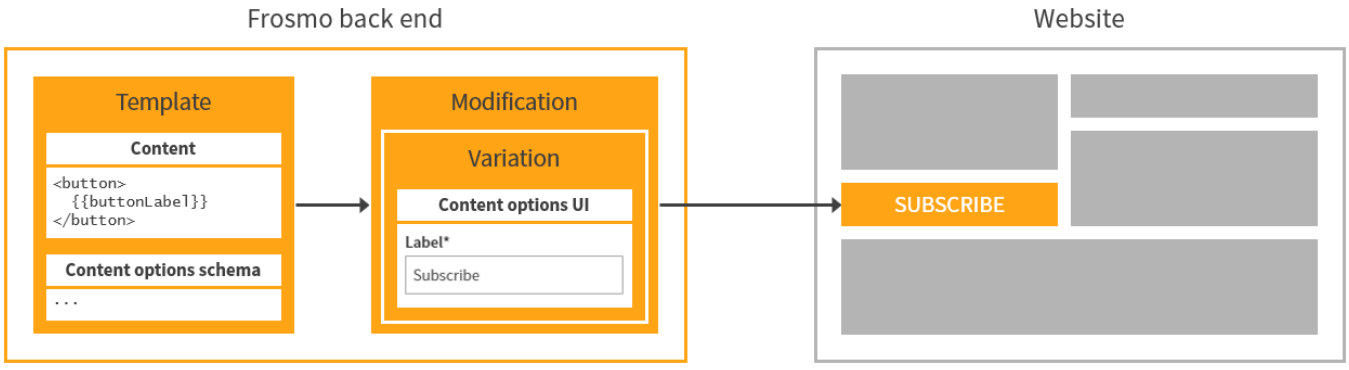



**Figure: From template content through content options to rendered content (click to enlarge)**

The content options schema is a JSON schema that describes:

- The data type of the value expected for each unique Mustache tag.
- The details for rendering a form input component for each unique Mustache tag. The components together make up the content options UI for customizing the template content in a modification variation. The Control Panel uses the `react-jsonschema-form` React component to build the UI. You set the Mustache tags to specific values in the UI when creating a modification variation.

For more information about defining content options schemas, see:

- Basic content options schema
- Supported data types for content options

For more information about how content options work in the modification flow, see Feature: Template.

## Basic content options schema

The following is the minimum JSON schema for a content options schema. Use it as a blueprint when creating your own schemas.

```
{
    "type": "object",
    "title": "Content options",
    "properties": {
        "mustacheTagKey": {
            "type": "fieldDataType",
            "title": "fieldLabel"
        }
    }
}
```

In the basic schema, the parts that you need to customize are:

- `title`: This is the title of the schema. The content options UI generated from the schema displays the title as its main heading. You can also leave the value empty or omit the `title` property entirely, in which case the content options UI displays no main heading.
- `properties`: This object defines the content options. You need to define a child object with at least the `type` and `title` properties for each unique Mustache tag in the template content. The child object key is the Mustache tag key. The `type` and `title` properties define the data type and label of the option, respectively, and determine the form input component rendered for the option in the content options UI. For more information about the different data types, see Supported data types for content options.

Here's a schema that showcases a more complex set of content options:

```
{
    "type": "object",
    "title": "Content options",
    "properties": {
        "stringField": {
            "type": "string",
            "title": "String field",
            "description": "Enter a string"
        },
        "arrayField": {
            "type": "array",
            "title": "Array field",
            "description": "Add items to the array",
            "items": {
                "$ref": "#/definitions/arrayItem"
            }
        }
    },
    "required": [
        "stringField"
    ],
    "definitions": {
        "arrayItem": {
            "type": "object",
            "title": "Array item",
            "properties": {
                "stringField": {
                    "type": "string",
                    "title": "String field in an array item",
                    "description": "Enter a string"
                },
                "numberField": {
                    "type": "integer",
                    "title": "Number field in an array item",
                    "description": "Enter a number"
                }
            }
        }
    }
}
```

For more information about writing JSON schemas and what properties you can use in a JSON schema, see the official documentation:

- Getting Started Step-By-Step
- Understanding JSON Schema
- JSON Schema specification

To generate JSON schemas from JSON data, and to test your schemas, check out the JSON Schema Tool.

## Supported data types for content options

The Frosmo Platform supports the following data types for content options:

- Array
- Boolean
- Number
- Object
- String

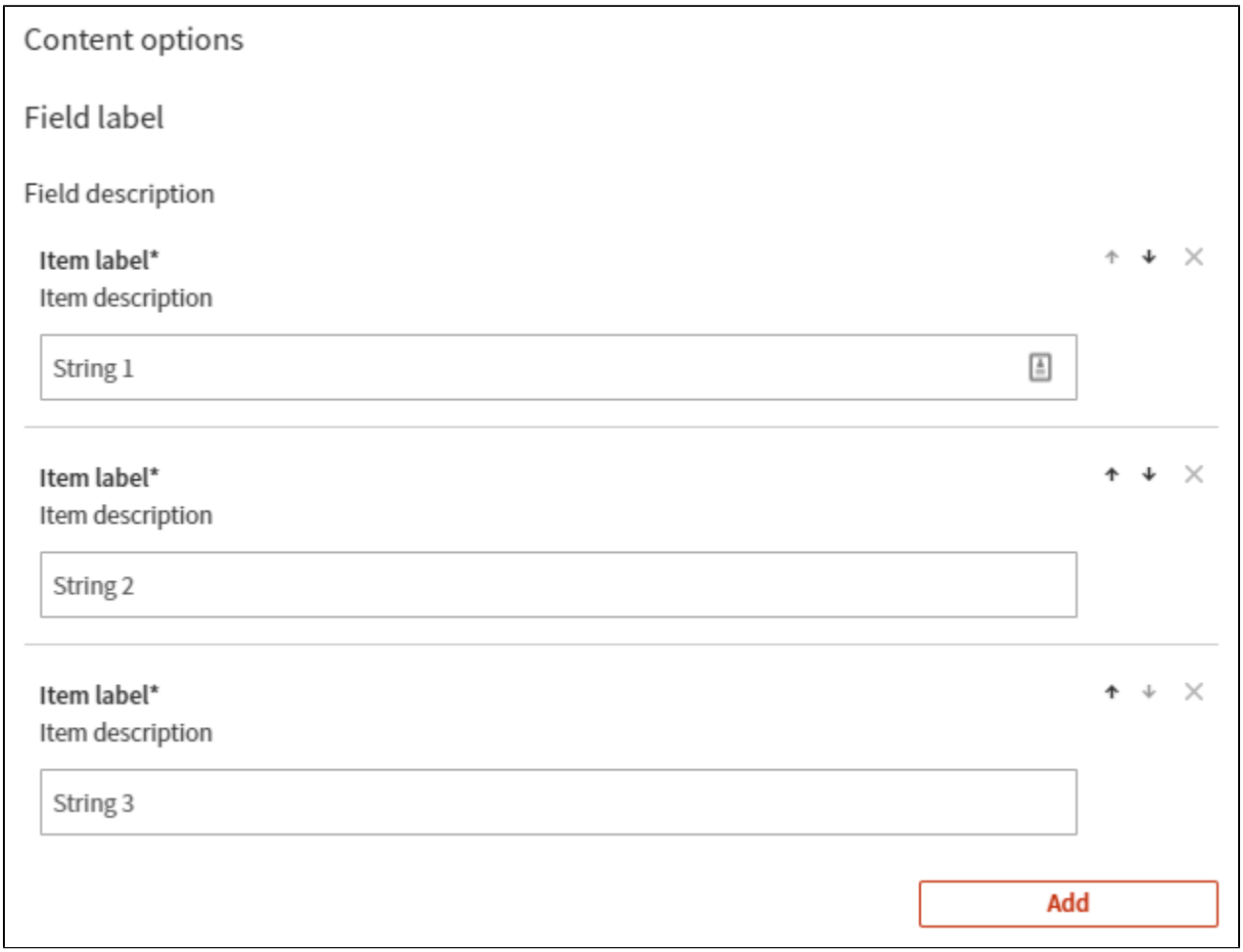
### Array

An array contains a single `items` property, which defines the data type or types of the array items. Depending on the specific item settings, the Control Panel generates the content options UI with zero or more form input components, each representing an item, and with controls for adding and removing items as appropriate.

**JSON schema: Array field (example)**

```
{
    "type": "object",
    "title": "Content options",
    "properties": {
        "arrayField": {
            "type": "array",
            "minItems": 1,
            "maxItems": 5,
            "title": "Field label",
            "description": "Field description",
            "items": {
                "type": "string",
                "title": "Item label",
                "description": "Item description",
                "default": ""
            }
        }
    }
}
```

The following figure shows the content options UI generated from the preceding array field example. The code below the figure is the JSON object generated from the settings shown in the figure.



**JSON object: Array field value (example)**

```
{
    "arrayField": [
        "String 1",
        "String 2",
        "String 3"
    ]
}
```

## Boolean

**JSON schema: Boolean field (example)**

```
{
    "type": "object",
    "title": "Content options",
    "properties": {
        "booleanField": {
            "type": "boolean",
            "title": "Field label",
            "description": "Field description",
            "default": false
        }
    }
}
```

The following figure shows the content options UI generated from the preceding Boolean field example. The code below the figure is the JSON object generated from the setting shown in the figure.


**JSON object: Boolean field value (example)**

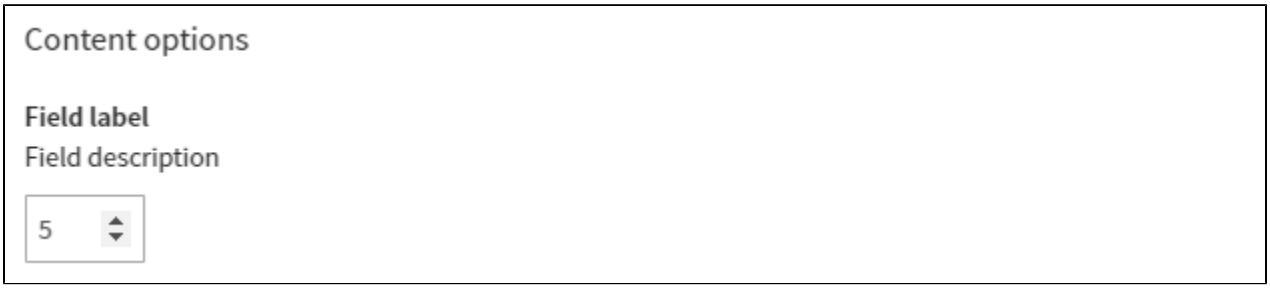
```
{
    "booleanField": true
}
```

## Number

**JSON schema: Number field (example)**

```
{
    "type": "object",
    "title": "Content options",
    "properties": {
        "numberField": {
            "type": "integer",
            "title": "Field label",
            "description": "Field description",
            "default": 0
        }
    }
}
```

The following figure shows the content options UI generated from the preceding number field example. The code below the figure is the JSON object generated from the setting shown in the figure.




**JSON object: Number field value (example)**

```
{
    "numberField": 5
}
```

## Object

An object can contain any number of fields of any data type, including arrays and other objects. The Control Panel generates the form input components for these fields normally based on their data type.

**JSON schema: Object field (example)**

```
{
    "type": "object",
    "title": "Content options",
    "properties": {
        "objectField": {
            "type": "object",
            "title": "Field label",
            "description": "Field description",
            "properties": {
                "key1": {
                    "type": "string",
                    "title": "Key 1 label",
                    "description": "Key 1 description",
                    "default": ""
                },
                "key2": {
                    "type": "integer",
                    "title": "Key 2 label",
                    "description": "Key 2 description",
                    "default": 0
                },
                "key3": {
                    "type": "boolean",
                    "title": "Key 3 label",
                    "description": "Key 3 description",
                    "default": false
                }
            }
        }
    }
}
```

The following figure shows the content options UI generated from the preceding object field example. The code below the figure is the JSON object generated from the settings shown in the figure.

**JSON object: Object field value (example)**

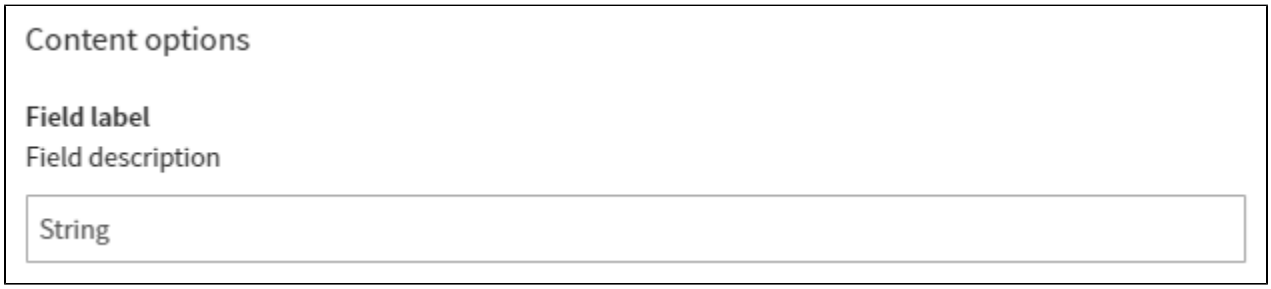
```
{
    "objectField": {
        "key1": "String",
        "key2": 5,
        "key3": true
    }
}
```

## String

**JSON schema: String field (example)**

```
{
    "type": "object",
    "title": "Content options",
    "properties": {
        "stringField": {
            "type": "string",
            "title": "Field label",
            "description": "Field description",
            "default": ""
        }
    }
}
```

The following figure shows the content options UI generated from the preceding string field example. The code below the figure is the JSON object generated from the setting shown in the figure.




**JSON object: String field value (example)**

```
{
    "stringField": "String"
}
```