1 Enter setup 3
1.1 Create configuration list 3
1.2 Enter setup or add / change the license code 4
1.2.1 Generate challenge code 5
1.2.2 Production or development mode 5
1.2.3 Select the form you want to configure 6
1.2.4 Install DFFS in the form 6
2 Configuration of a form 6
2.1 Banner buttons 7
2.1.1 Save 7
2.1.2 Import /Export 7
2.1.3 Create a restore point 8
2.1.4 Switch form 8
2.1.5 Delete 8
2.2 Formbuilder 8
2.2.1 Row 8
2.2.1.1 Accordions 8
2.2.2 Column 8
2.2.3 Tab set 8
2.2.3.1 Set selected tab in a tab set 9
2.2.4 Rich text 9
2.2.5 HTML 9
2.2.5.1 Adding event handlers on custom HTML elements 9
2.2.5.2 Hiding custom HTML elements from the printout 10
2.2.6 Grid 10
2.2.7 Fields 10
2.2.7.1 Attachments 10
2.2.7.2 Tooltips 10
2.2.7.3 Render a lookup field as a tree view 10
2.2.7.4 Create cascading lookups 11
2.2.7.5 vLookup 16
2.2.7.6 Render a single line of text field as a Dropdown 18

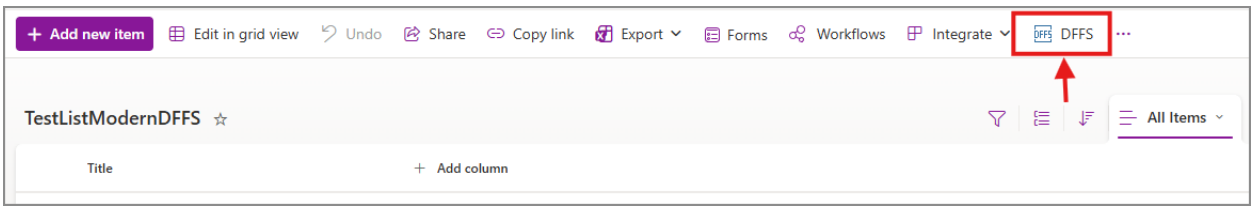
2.2.7.7 Render a single line or multi line of text field as an Autocomplete 20
2.2.7.8 Cascading dropdowns / autocompletes 22
2.2.7.9 Render a single line of text field using a custom render function - advanced 24
2.2.7.10 Translate a choice field 25
2.3 Preview form 25
2.4 Rules 25
2.4.1 Run based on the result of a Custom JS function (on form load) 27
2.4.2 Run from a Custom JS function 27
2.5 Custom JS 27
2.5.1 Some examples 27
2.5.1.1 Check the previous value of a field 27
2.5.1.2 Get the list context of the current form 28
2.5.1.3 Select a tab in a tabset from custom js 28
2.5.1.4 See the current users' group membership and user profile 28
2.5.1.5 Print the form 28
2.5.1.6 Check the status of a rule from Custom JS 28
2.5.1.7 Run code when the form is loaded 29
2.5.1.8 Do something before the item is saved 29
2.5.1.9 Save the item from Custom JS 29
2.5.1.10 Do something after the item is saved 29
2.5.1.11 Do something if the item is cancelled / closed 30
2.5.1.12 Cancel the form from Custom JS 30
2.5.1.13 Do something if the item is deleted 30
2.5.1.14 Show a modal dialog 30
2.5.1.15 Trigger custom code when a field is changed 30
2.5.1.16 Add a callback when a vLookup has loaded 31
2.5.1.17 Reload the vLookup table from Custom js 31
2.5.1.18 Open a vLookup list item from Custom JS 31
2.5.1.19 Set a vLookup are required from custom js 31
2.5.1.20 List the attachments on a list item 31
2.6 Custom CSS 31
2.7 Miscellaneous 32
2.7.1 Password 32

2.7.2 Override strings used in DFFS 32
2.7.3 Form panel type (only for vLookup child forms) 32
2.7.4 Show panel type menu in the form (only for vLookup child forms) 32
2.7.5 Currency symbol position for currency fields 32
2.7.6 Comments 32
2.7.7 Use Jodit Editor 32
2.7.8 Show an icon to the left of the field label 32
2.7.9 Show option to quick-save the form without closing it 32
2.7.10 Show button to turn on automatic saving in the footer of the form 32
2.7.11 Tab colors 32
2.7.12 Load the configuration from another site or from a file 33
2.8 Note to self 33
2.9 Install DFFS 33
3 Linking to a DFFS form 33
4 Sending emails using FLOW 33
4.1 FLOW configuration for Send now 34
4.2 FLOW configuration for Send later 39
5 DFFS Configuration WebPart 44
6 Modern DFFS Formpage Web Part 45
6.1 Setting a default redirect in the form URL 46
7 Upgrade from Classic DFFS 47
8 Troubleshooting 47
8.1 Blocked web resources 47
8.2 Other blocked resources 47

1 Enter setup

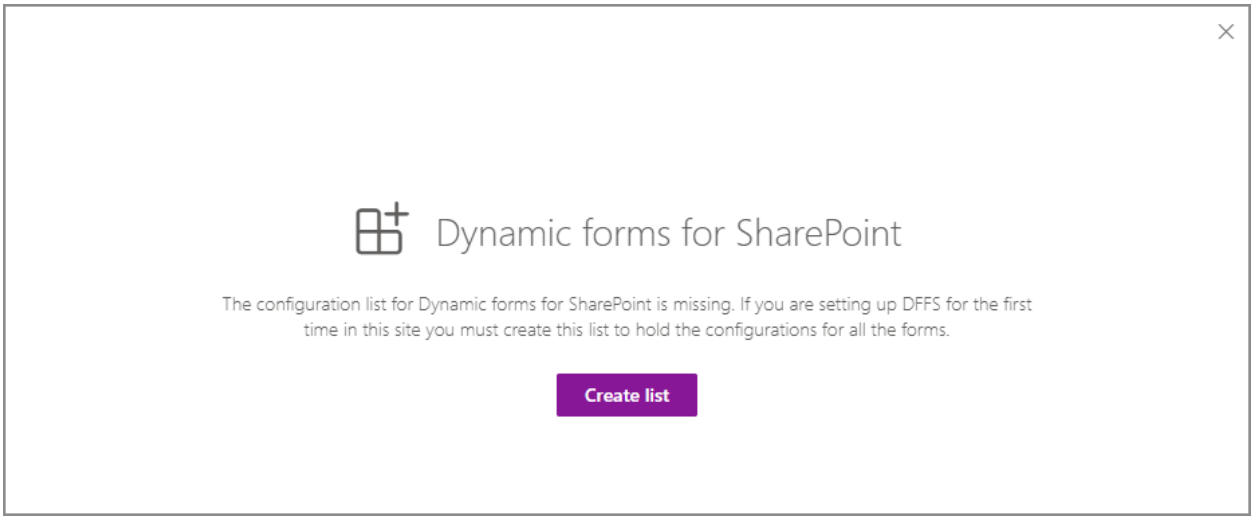
1.1 Create configuration list

When you have installed the Modern DFFS you will see a button **DFFS** in the banner of the list. This will only be shown for users with **Manage lists** permission. Click this button to enter the setup.



In v1.2.0 a new DFFS configuration web part has been added. It lets you configure the forms from a site page on each site. Read more about this web part in a separate section.

The first time you enter setup you must create the configuration list:

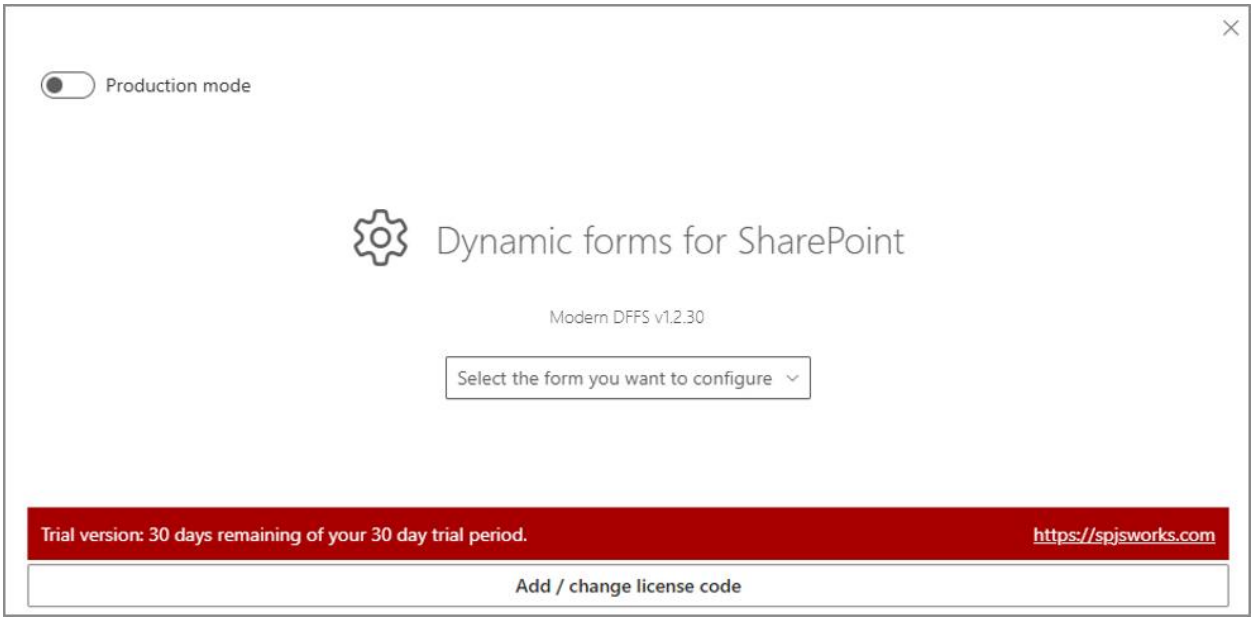


This configuration list is a standard *custom list*, but it is by default hidden from all site contents. You are not supposed to manually edit this list, but it can be accessed by typing in the list name in the URL like this: *…/Lists/DFFSConfigurationList*

Please note that all users must have permission to view items in this list to access the configuration for the list.

1.2 Enter setup or add / change the license code

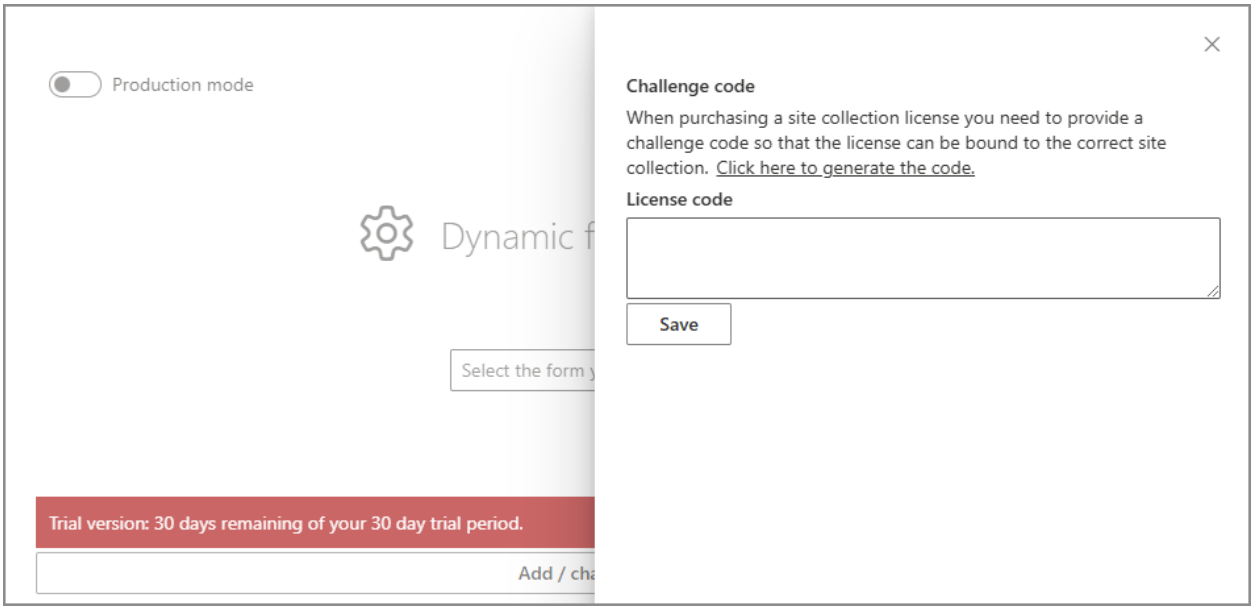
When you have created the configuration list and clicked the DFFS button again you will see this screen:



1.2.1 Generate challenge code

When you purchase a site collection license you must generate a challenge code to bind the license code to the current site collection. Click the **Add / change license code** to access the challenge code.

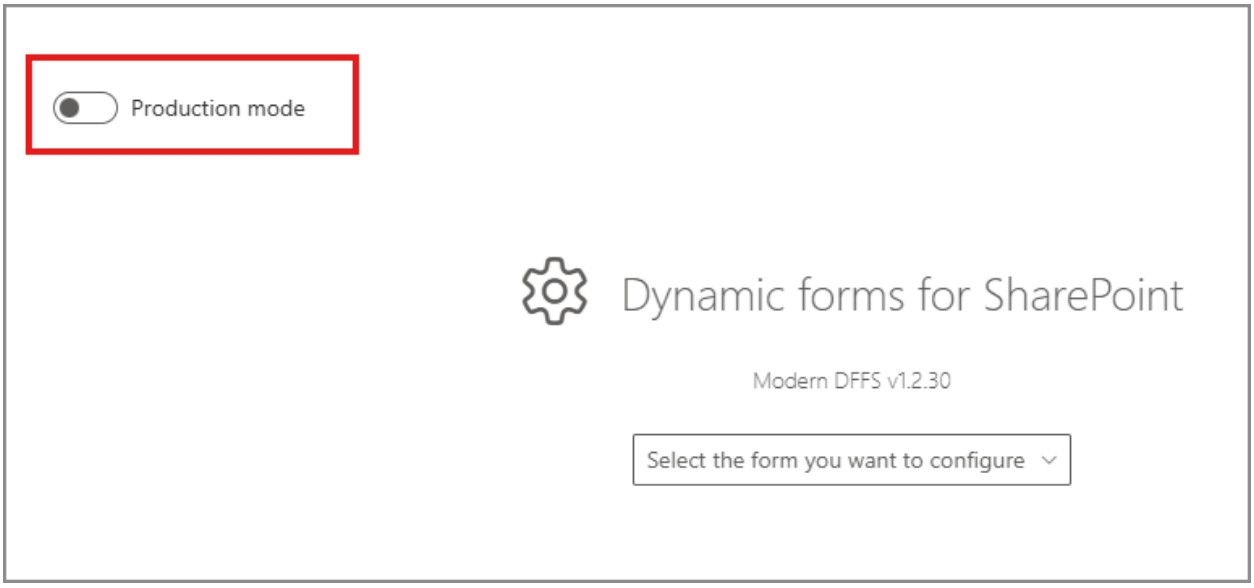
When the license code is saved it can only be viewed by the user that saves the license code.



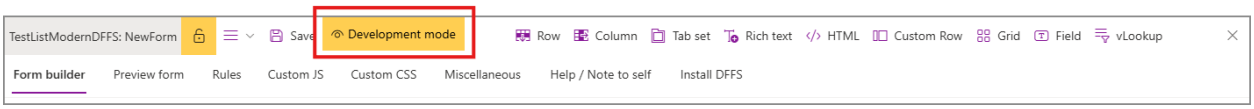
You get a 30-day trial when you first install the solution. You can purchase a license or ask for a quote from here: https://spjsworks.com/purchase

1.2.2 Production or development mode

You can toggle between production and development mode using the toggle in the top left corner:



When changing to ***Development mode*** and entering a form configuration you will see a yellow button in the banner:



If you click this button, you can assess your development mode configuration in a new window.

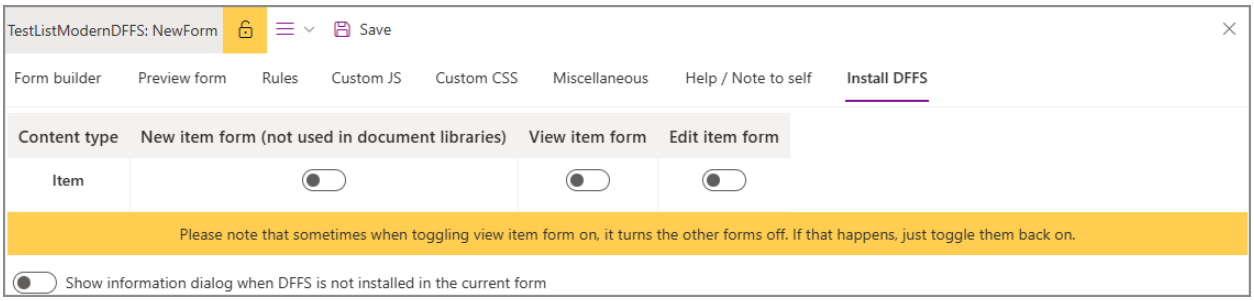
1.2.3 Select the form you want to configure

You can select the form you want to configure from the drop-down menu. You can choose from the following forms: NewForm, DispForm and EditForm.

You use NewForm to create new list items (a document library does not have an NewForm), DispForm to display an existing list item and EditForm to edit an existing list item.

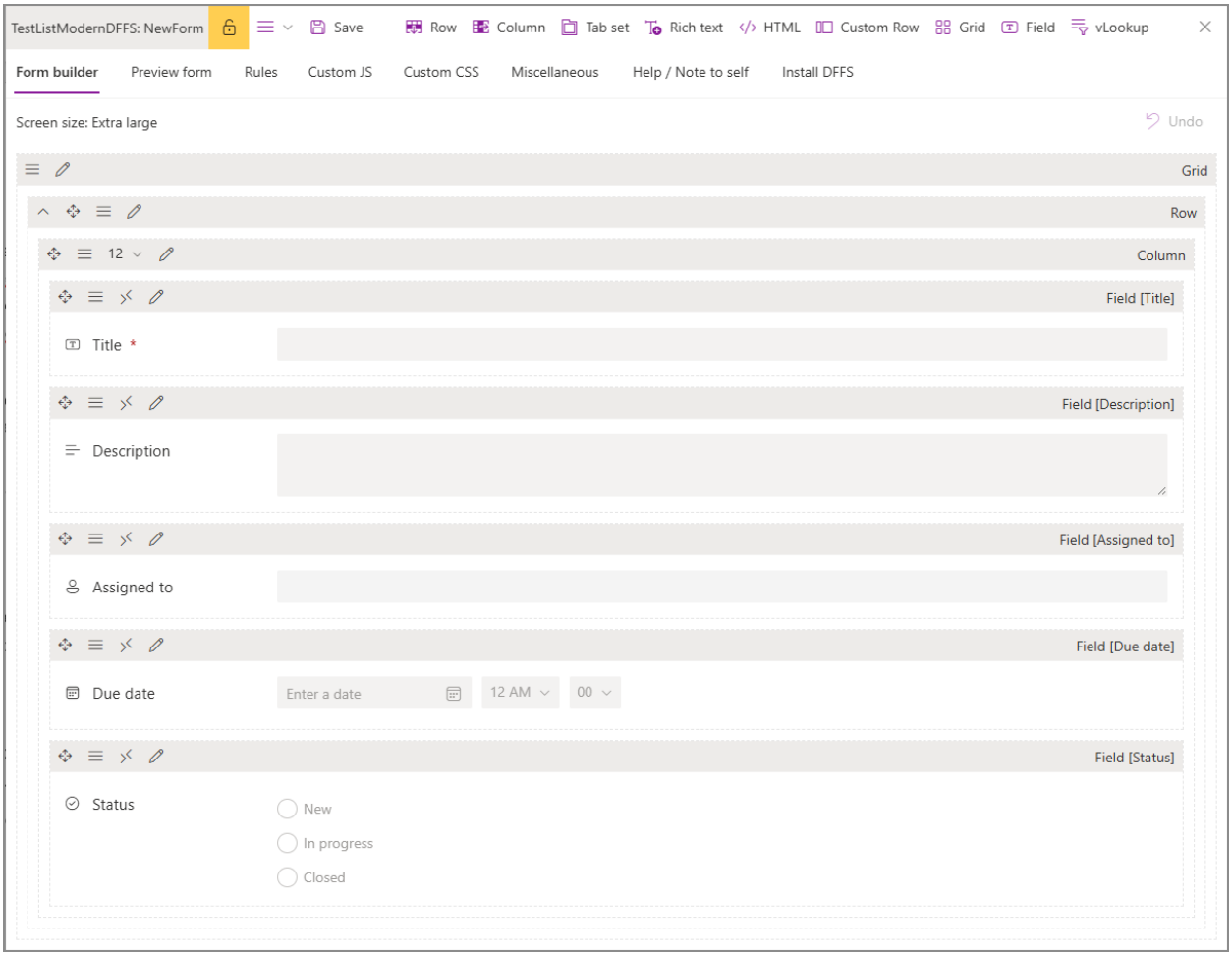
1.2.4 Install DFFS in the form

From v1.1.x of Modern DFFS you must install / register DFFS as the default form for each of the forms in the content types in your list. Use the Install DFFS tab and toggle the button to activate it.



2 Configuration of a form

When you open the configuration for the first time, all fields are automatically added to the form. You can delete the ones you do not want to have visible in the form. Deleting a field from the configuration for one form does not delete it from the list settings.



To add new fields to your list you must use the default list settings. You will find a link to open list settings from the Miscellaneous tab.

You can rearrange the fields by drag-and-drop, and you can access the properties pane for all elements by using the hamburger button or the right click menu. The different form elements have different properties – you find a bit more information in the next section.

2.1 Banner buttons

2.1.1 Save

Saves the configuration.

2.1.2 Import /Export

This opens a panel where you can browse through other configurations and restore points created for this list, import configurations from the Classic DFFS (if you have the Classic DFFS version already in use in this site) or import / export as JSON from the Classic DFFS or the Modern DFFS.

See more details in the Upgrade from Classic DFFS section below.

2.1.3 Create a restore point

Create a restore point that lets you change back to this configuration if needed. It is recommended that you create a restore point before making major changes.

2.1.4 Switch form

Switch between NewForm, DispForm and EditForm of the list (NewForm is not available in document libraries).

2.1.5 Delete

Use this to send the configuration to the recycle bin and change back to the out of the box SharePoint form.

2.2 Formbuilder

Use the banner buttons to drag-and-drop form elements onto the form.

You can choose from the following elements:

- Row
- Column
- Tabset
- Rich text
- HTML
- Custom Row
- Grid
- Field
- vLookup

You can configure all form elements by opening the properties pane using the hamburger button or the right click menu.

2.2.1 Row

Rows are used as placeholders for columns. You can add an unlimited number of rows to your form.

2.2.1.1 Accordions

You can configure rows as accordions to let the user collapse each row in a form. Select Row properties to configure a row as an accordion.

2.2.2 Column

A row has 12 available sections. You can have one column that takes up all 12 sections or add any number of columns (up to 12) and distribute the sections between them.

You can set the distribution of space depending on the screen size to make a more responsible for different screen sizes. You can find these settings under column properties.

2.2.3 Tab set

You can create tab sets and add any number of tabs. If you do not want to use the default tab color (it is based on the color theme of your site) you can set the color of the individual tab, or add a default color

for all tabs in the Miscellaneous tab. You can make a tab set “sticky” to ensure it stays on the top of the screen when scrolling a long form.

Each tab will have a placeholder where you can add rows and columns.

You can configure a tab set as a multi-step form where you hide the tabs and instead show a previous and next button at the bottom of the tab set, and you can navigate between the tabs like in a multi-step form. There are bullets below the tab set that show which tab you are currently on, and if a required field is missing data in that tab when you try to save the form.

You can detach the tabs from the tab set and for example place them vertically to the side of the tab set.

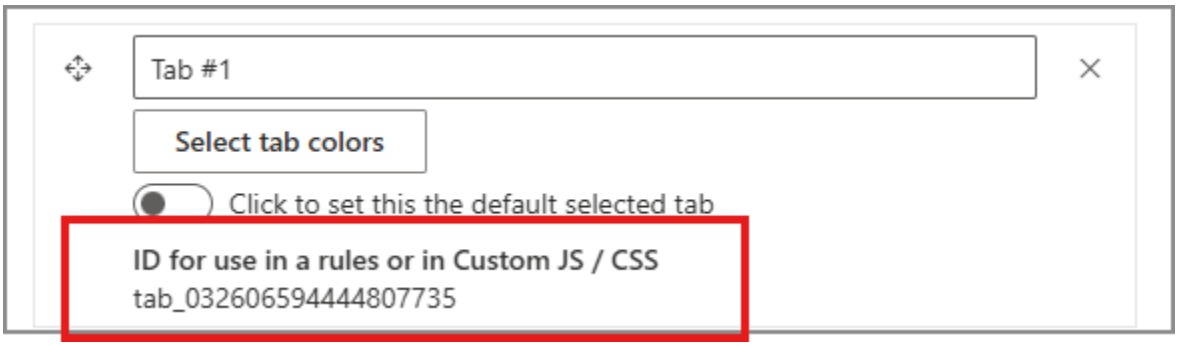
2.2.3.1 Set selected tab in a tab set

When configuring a tab set you can set the default selected tab for each tab set.

If you want to create a link to a form and want to select a specific tab - that is not the default selected tab specified in the tab set - you can create a link with an URL query string key like this:

sTab=[id_of_the_tab]

To find the ID of the specific tab you must edit the tabset and find the id here:



The number is used in the URL key like this: sTab= tab_032606594444807735

2.2.4 Rich text

You can use this control to add your own rich text to the form. This is not text that will be saved to the list, but can be used for instructions to the user that fill in the form.

2.2.5 HTML

You can add your own custom HTML. Accompany this with Custom JS and Custom CSS to create your own form elements.

2.2.5.1 Adding event handlers on custom HTML elements

Microsoft has removing support for inline event handlers (e.g. onClick="...") directly on HTML elements on March 1, 2026. Event handlers must be applied to the HTML elements using Custom JS as shown below.

Create the button in the HTML section like this

```
<input type="button" id="customBtn" value="Example button" />
```

Add this code to Custom JS to attach the click-eventhandlers

```
dffs_addClickEventListener("customBtn", () => {
  // Add your custom code here
});
```

You can also add other eventhandlers using this function

```
dffs_addEventListener('customDropdown', 'change', () => {
  // Add your custom code here
});
```

2.2.5.2 Hiding custom HTML elements from the printout

You can prevent a custom HTML element from being visible when printing the form by adding the class ***dffs_no_print*** to the HTML element.

2.2.6 Grid

Use this to make a grid layout where you can add rows and columns. You can set the width of the form or apply for example a background color by using the Grid properties panel.

2.2.7 Fields

You can select from all the built in SharePoint field types. To be able to add a field to the form, you must first create it in **list settings** for that list. You find a button to open the list settings in the Miscellaneous tab.

In the Field properties panel, you can change the field label, the styling for the label or the container or add conditional styling. See more information in the Field properties panel.

2.2.7.1 Attachments

When configuring a DFFS form you can add the Attachments field more than once to a form, and configure each control to add a suffix on the filename for files added using the attachment control.

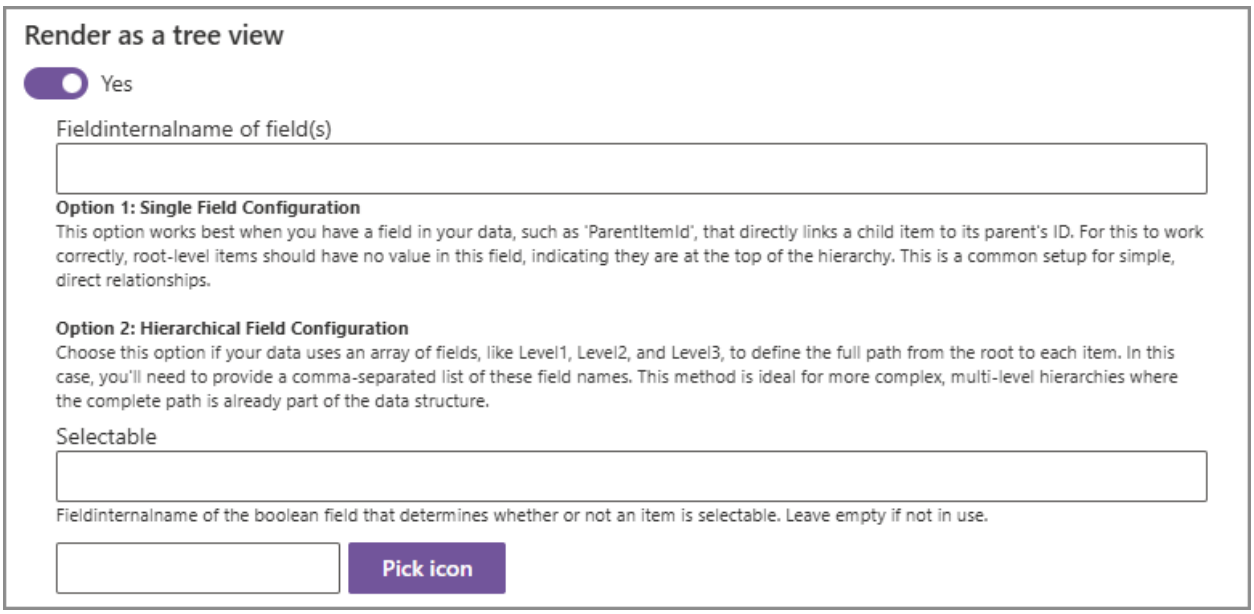
If activated, only files that have this suffix will show in this specific control. Use it when you want to show different attachments in different sections in the form. You can find the new functionality in the field properties panel on the attachment field.

2.2.7.2 Tooltips

You can configure tooltips for fields by clicking the hamburger icon (or right click) and selecting ***Tooltip***.

2.2.7.3 Render a lookup field as a tree view

A lookup column can be rendered as a tree view. Open the field properties and toggle **Render as tree view** to Yes.



There are two different options for configuring the treeview.

Option 1: Single Field Configuration

This option works best when you have a field in your data, such as 'ParentItemId', that directly links a child item to its parent's ID. For this to work correctly, root-level items should have no value in this field, indicating they are at the top of the hierarchy. This is a common setup for simple, direct relationships.

Option 2: Hierarchical Field Configuration

Choose this option if your data uses an array of fields, like Level1, Level2, and Level3, to define the full path from the root to each item. In this case, you'll need to provide a comma-separated list of these field names. This method is ideal for more complex, multi-level hierarchies where the complete path is already part of the data structure.

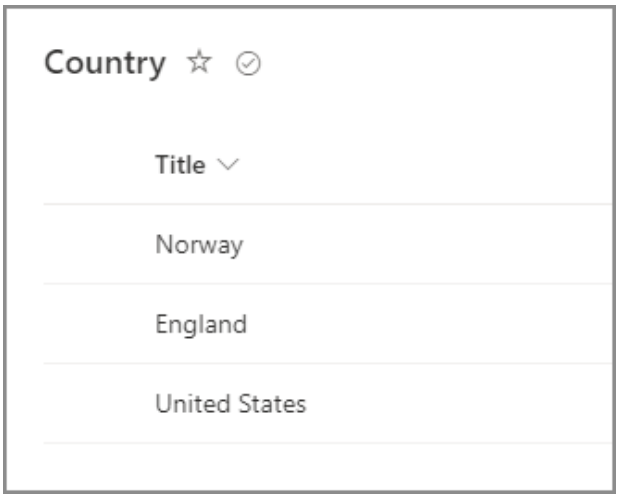
You can also have a Boolean field in your list that determines whether you can select each item or not.

2.2.7.4 Create cascading lookups

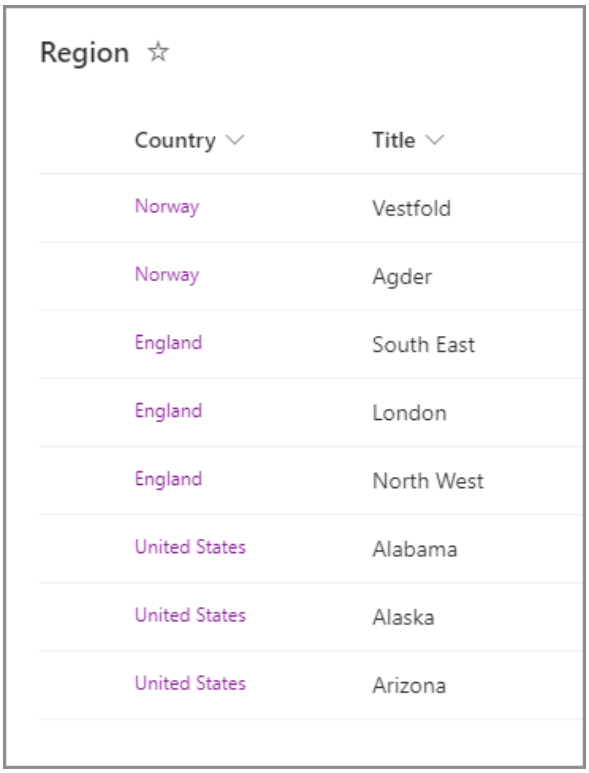
You can make your lookup fields cascading by using the Filter option. Click the filter field and use the **Add dynamic content** menu in the bottom right corner to pick the field you want to use as filter. The filter is updated when that field is changed.

This example shows how you can set up two lookup columns in your form *Country* and *Region* where the second one is filtered by the selection in the first. To do this you must create two lists that your lookup columns will get their values from – one for country and one for region.

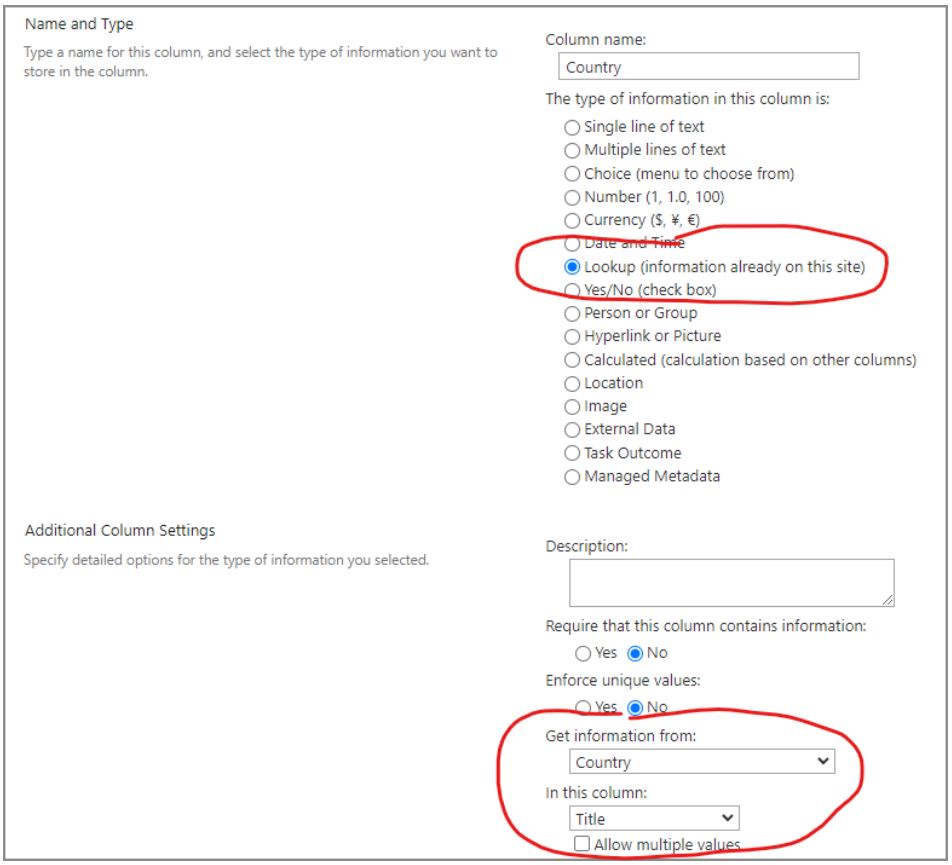
The first list *Country* is set up with only the Title field:



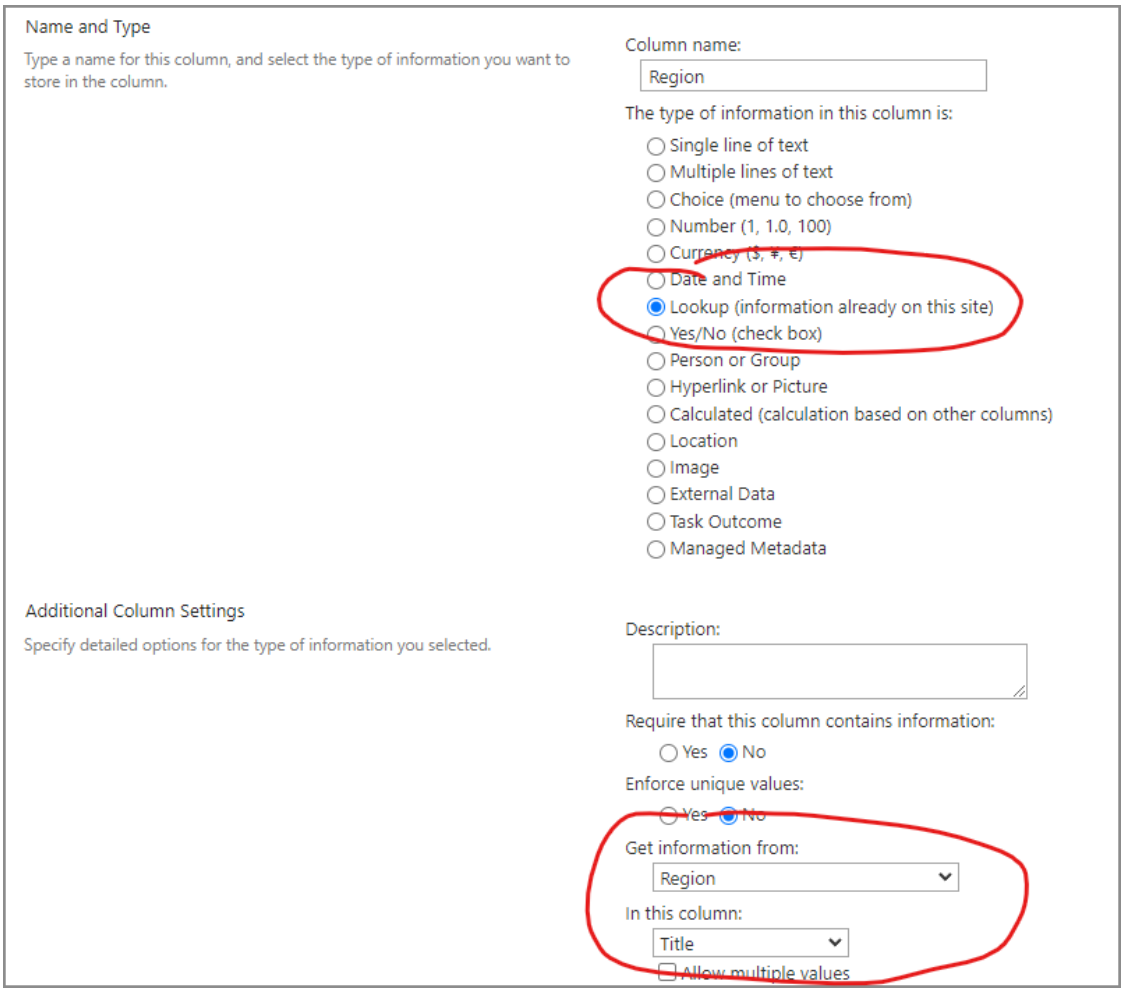
The second list *Region* is set up with a lookup column that looks up the Title field from the *Country* list. The Region name is specified in the Title field:



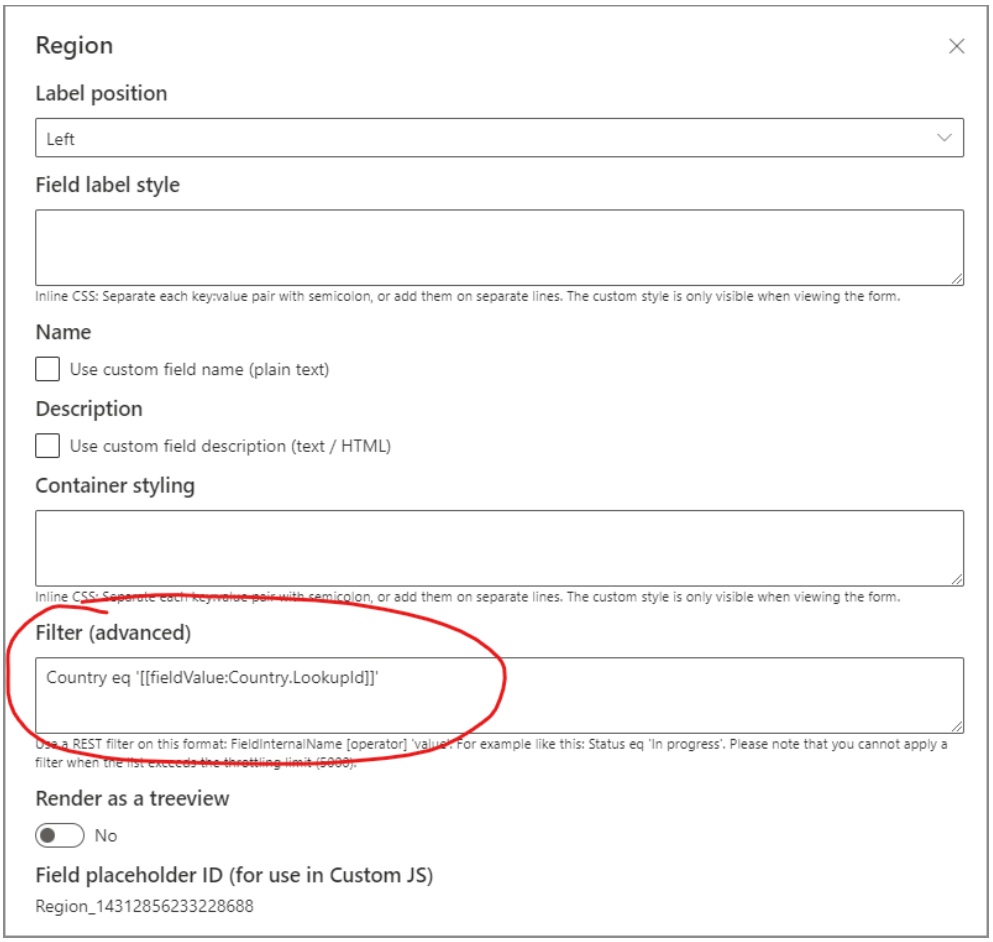
Now you can create the two lookup columns in your form – one for *country*



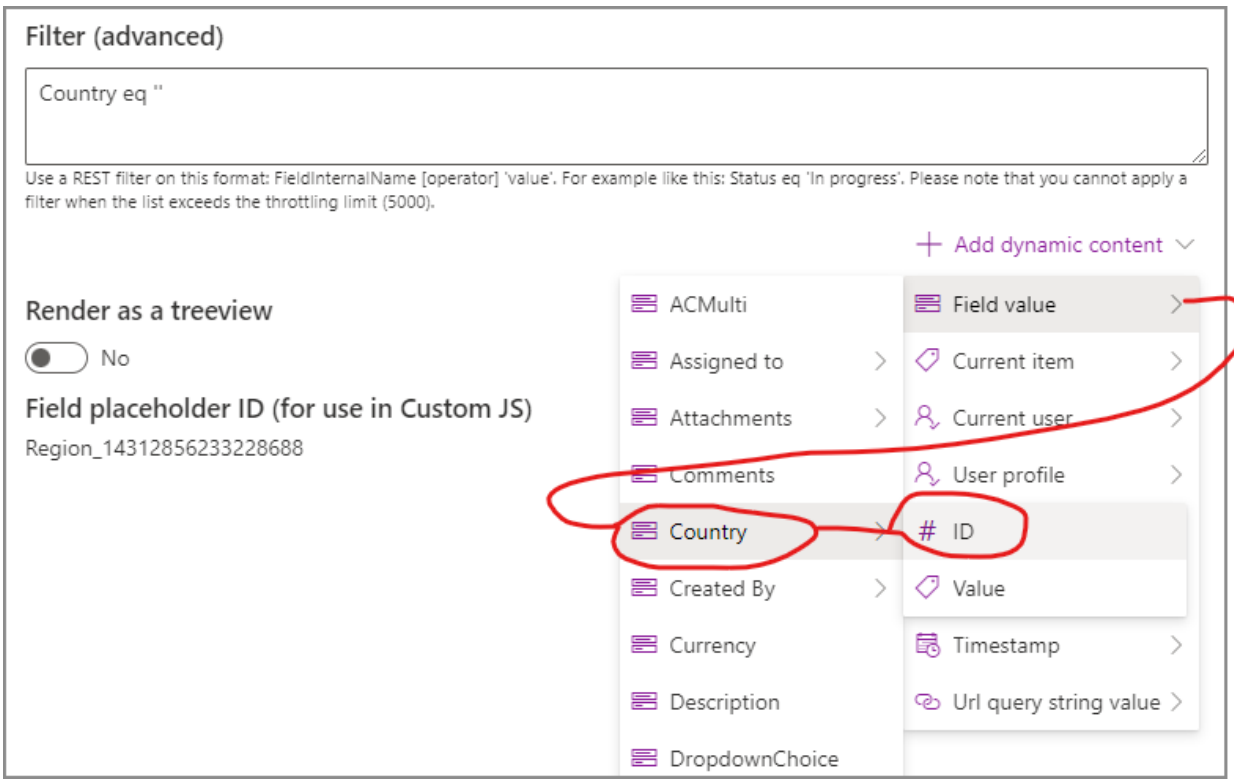
And one for *region*


Name and Type
Type a name for this column, and select the type of information you want to store in the column.
Column name:
Region
The type of information in this column is:
Single line of text
Multiple lines of text
Choice (menu to choose from)
Number (1, 1.0, 100)
Currency ($, ¥, €)
Date and Time
Lookup (information already on this site)
Yes/No (check box)
Person or Group
Hyperlink or Picture
Calculated (calculation based on other columns)
Location
Image
External Data
Task Outcome
Managed Metadata
Additional Column Settings
Specify detailed options for the type of information you selected.
Description:
Require that this column contains information:
Yes
No
Enforce unique values:
Yes
No
Get information from:
Region
In this column:
Title
Allow multiple values

Now you can enter the Modern DFFS configuration to set up the filter on the *Region* lookup column:



The field name *Country* in the filter is the internal name of the lookup field *Country* in the list *Region*. The filter value *[[fieldValue:Country.LookupId]]* can be found by clicking the Add dynamic content below the textarea (shown when you focus on the textarea) and selecting the lookup field for Country in the current list – like this:



Your Regions column will now be filtered to show only the regions that fall under the Country selected in the first lookup field.

2.2.7.5 vLookup

Use this control to show a table view of items from a child list. You configure the control by selecting the web, list, and filter to retrieve the desired items.

You can select which fields to populate the table and configure **Add new** to be able to create a new child item that is automatically linked to the current item by prefilling values from the current item (the same values as you use in the filter).

Linking parent and child forms using _DFFSID or _vLookupID

Like in the classic DFFS version you can link the “parent” and “child” lists using an automatically generated unique id in your form instead of having to use a lookup column to link the two.

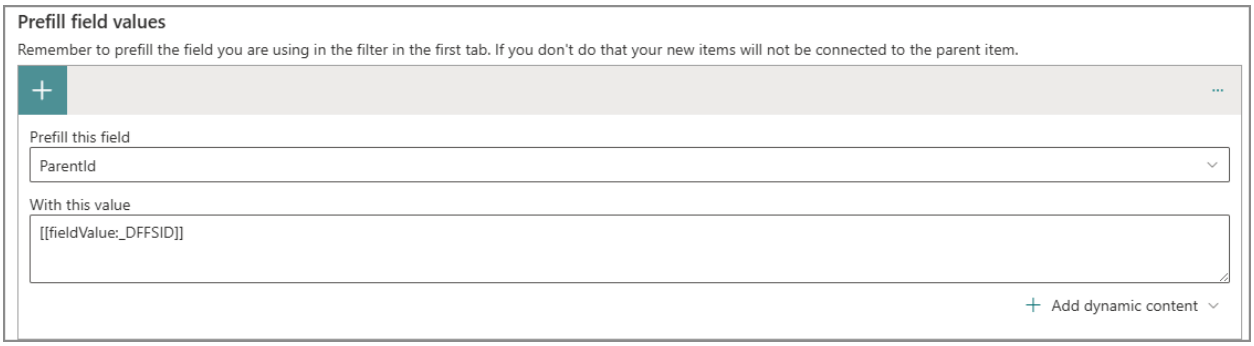
If you add a field named **_DFFSID** to your list, it will automatically be populated with an auto-generated id (a timestamp in milliseconds). This can, using the **Prefill field values** functionality, be transferred to a single line of text field that you create in the child for.

You can use this field in the filter using a format like this:

```
ParentId eq '[[fieldValue:_DFFSID]]'
```

Where ***ParentId*** is the field that you created in the child-list.

Please note that for this filter to work you must write the ***_DFFSID*** value to the child items when you create new. This is done in the Add new item tab like this:



For backwards compatibility with older lists created with the classic DFFS version you can also use the ***_vLookupID*** field – if added to the list, it will auto-generate a similar unique id as for the classic DFFS version.

Add a callback function when vLookup table is ready

You can add a custom function that triggers every time a vLookup table is loaded (on form load, or if a new item is added) by using code like this in Custom JS:

```
function dffs_vLookup_callback(id, items){
  if(id === "vLookup_7943515991751215"){
      // Loaded vLookup with id vLookup_7943515991751215 - you can find the ID when editing the vLookup configuration
     console.log(id, items);
  }
}
```

Override configuration for a vLookup in Custom JS

This is for advanced users and can for example be used to switch between different document libraries depending on a selection in the form.

To use this functionality, you must configure the vLookup using the vLookup control and then add an object like this in the Custom JS editor in the form to override the values:

```
var vLookup_config_override = {};
if (getFieldValue("DocLibSelect") === "DocLibA") {
 vLookup_config_override["vLookup_13389631338251728"] = {
  "list": {
    "override": "DocLibA",
    "guid": "f31dfde0-3299-4415-80fd-64806cf091bc",
    "title": "DocLibA",
    "rootfolder": "DocLibA",
    "BaseTemplate": 101
  }
 };
} else {
 vLookup_config_override["vLookup_13389631338251728"] = {
  "list": {
    "override": "DocLibB",
    "guid": "91c5e295-ccdd-4855-8419-06ec284c0c22",
    "title": "DocLibB",
    "rootfolder": "DocLibB",
    "BaseTemplate": 101
```

```
    }
  };
}
```

The ID used in the vLookup_config_override object can be found in the first tab in the vLookup configuration. You can override all the values, but because of how the override object is merged with the current configuration in the code you must add all properties and not for example only “guid” in “list” in the example above.

To find the complete vLookup configuration object you must export the configuration (using the Import, export and restore menu item in the top left corner) and parse the data from the FormJSON property.

Refresh a vLookup table from Custom JS

If you want to refresh a vLookup table from Custom JS you can use this function:

```
vLookup_loadItems("vLookup_06588267196671516").then(() => {
  console.log("done");
});
```

The id can be found in the vLookup configuration at the bottom of the first tab.

Open a vLookup form from Custom JS

To open a vLookup from form Custom JS you can use code like shown below. Opening NewForm uses only 2 arguments – the ID for the vLookup and the form type “new” like this:

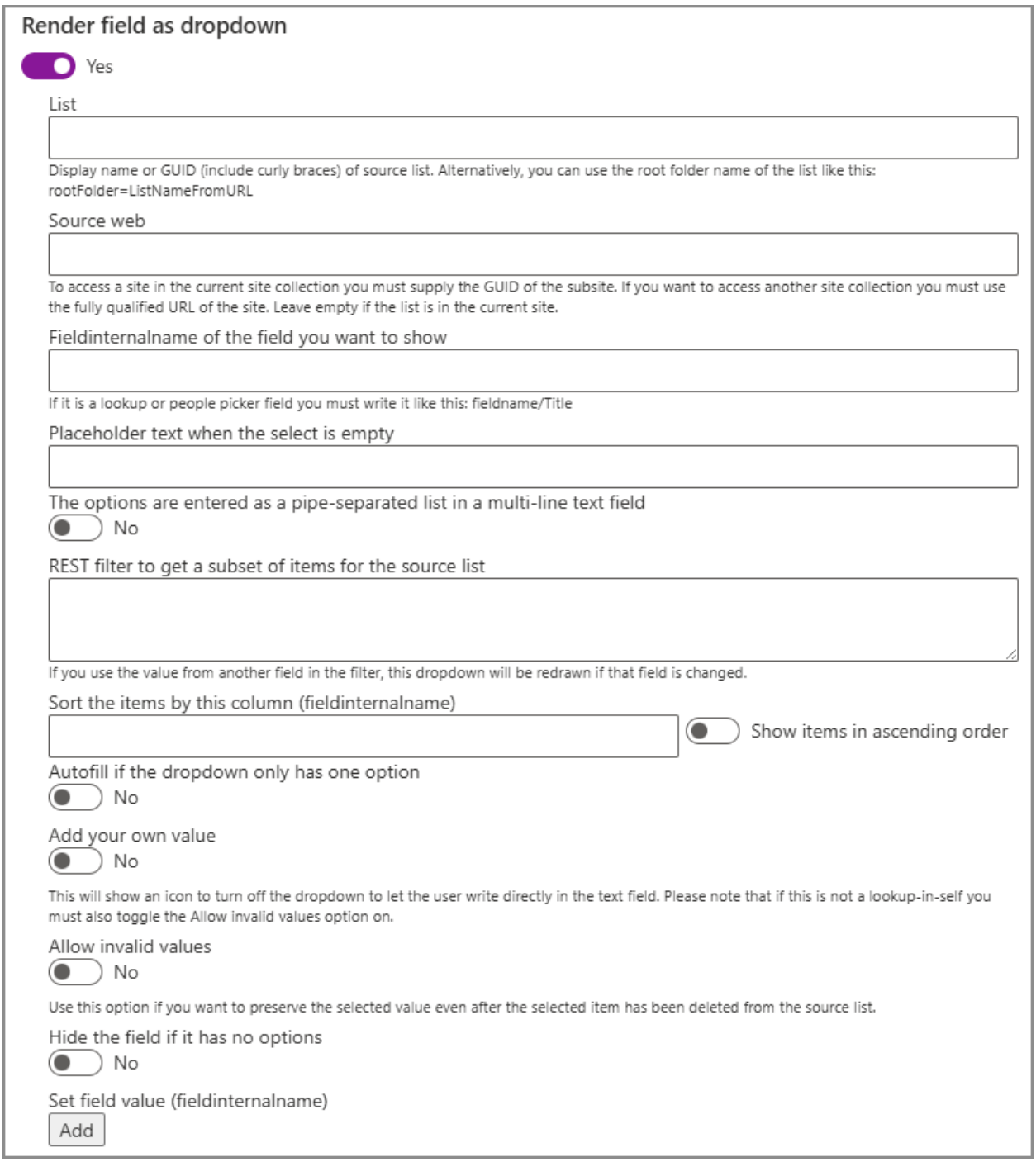
```
vlookup_open("vLookup_06588267196671516", "new");
```

Opening DispForm or EditForm uses three arguments – the id of the vLookup, the form type and the Item ID like this:

```
vlookup_open("vLookup_06588267196671516", "disp", 123);
vlookup_open("vLookup_06588267196671516", "edit", 123);
```

2.2.7.6 Render a single line of text field as a Dropdown

You can render a single line of plain text field as a dropdown that lists the options from another list in your SharePoint site collection. Open the field properties and toggle **Render field as dropdown** to Yes. Configuration is done in the field properties like this:



Now specify the display name or the GUID of the source list, the internal name of the field you want to use as the selectable options, the placeholder text, and any REST filter you want to use to filter the data source. You can also change the sort order. You can also check **Autofill if the dropdown only has one option** to have it automatically fill in the value when it is the only option.

Pipe-separated options

Check this box if you have the options for the dropdown in a multiline textfield in this format:

```
Red|Green|Yellow|Blue
```

Please note that if you check this option, the Set field value functionality will be disabled.

Autofill if the dropdown only has one option

If your dropdown only has one option, toggling this selection to ON will automatically select that option.

Add your own value

This will show an icon to turn off the dropdown to let the user write directly in the text field. Please note that if this is not a lookup-in-self you must also toggle the Allow invalid values option on.

Allow invalid values

Use this option if you want to preserve the selected value even after the selected item has been deleted from the source list.

Set field value

If you use a single line of text field you can set additional values by specifying the field to pull the data from and the field to write it to. You can write the From field like this to pull from a complex field type like a people picker or a lookup column:

```
FieldInternalName/Title
```

Remove dropdown functionality

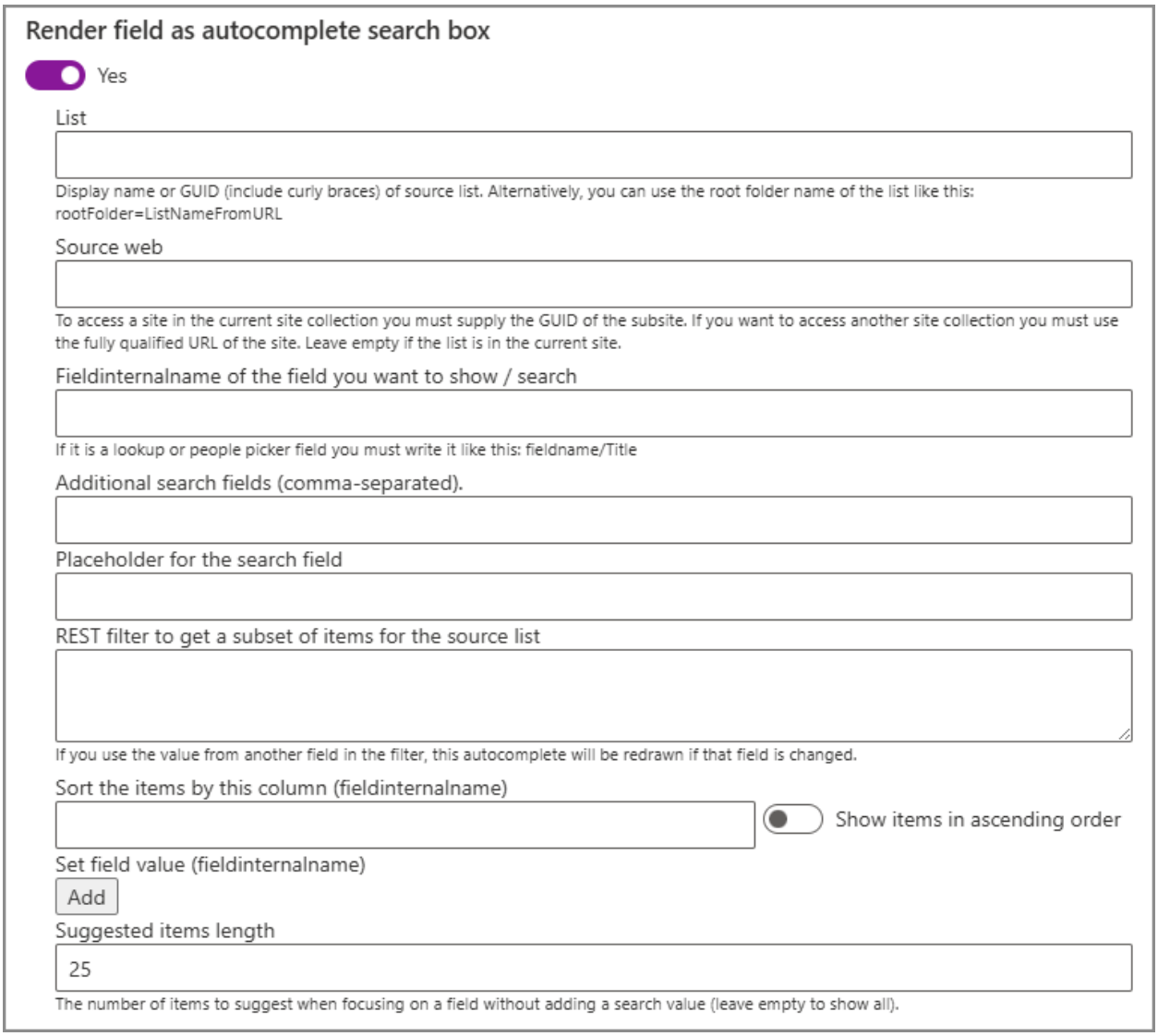
If you have a field that is rendered as a dropdown, you can remove the dropdown functionality from the field from Custom JS using the id of the field like this:

```
dffs_cancel_fieldAsDropdown("YourField_600232300999622");
```

2.2.7.7 Render a single line or multi line of text field as an Autocomplete

You can render a single line (for single choice) or multiline (for multi choice) plain text field as an autocomplete lookup field that lists the options from another list in your SharePoint site collection.

Open the field properties and toggle **Render field as autocomplete search box** to Yes. Configuration is done in the field properties like this:



Now specify the display name or the GUID of the source list, the internal name of the field you want to use as the selectable options, the placeholder text, and any REST filter you want to use to filter the data source. You can also change the sort order. Additional search fields can be used if you want to let the user search for records using multiple search fields.

You can make your autocompletes cascading by using the **REST filter to get a subset of items for the source list** option. Focus on the filter field and use the **Add dynamic content** menu in the bottom right corner to pick the field you want to use as filter. The filter is updated when that field is changed.

Set field value

If you use a single line of text field you can set additional values from the lookup source list by specifying the field to pull the data from and the field to write it to. You can write the From field like this to pull from a complex field type like a people picker or a lookup column:

```
FieldInternalName/Title
```

Remove autocomplete functionality

If you have a field that is rendered as an autocomplete, you can remove the autocomplete functionality from the field from Custom JS using the id of the field like this:

```
dffs_cancel_fieldAsAutocomplete("YourField_600232300999622");
```

2.2.7.8 Cascading dropdowns / autocompletes

You can make your custom dropdowns and autocompletes cascading by using the **REST filter to get a subset of items for the source list** option.

In the below example I will set up three levels of cascading fields (two single choice dropdowns and one multiselect autocomplete) based on data in a custom list using this format:

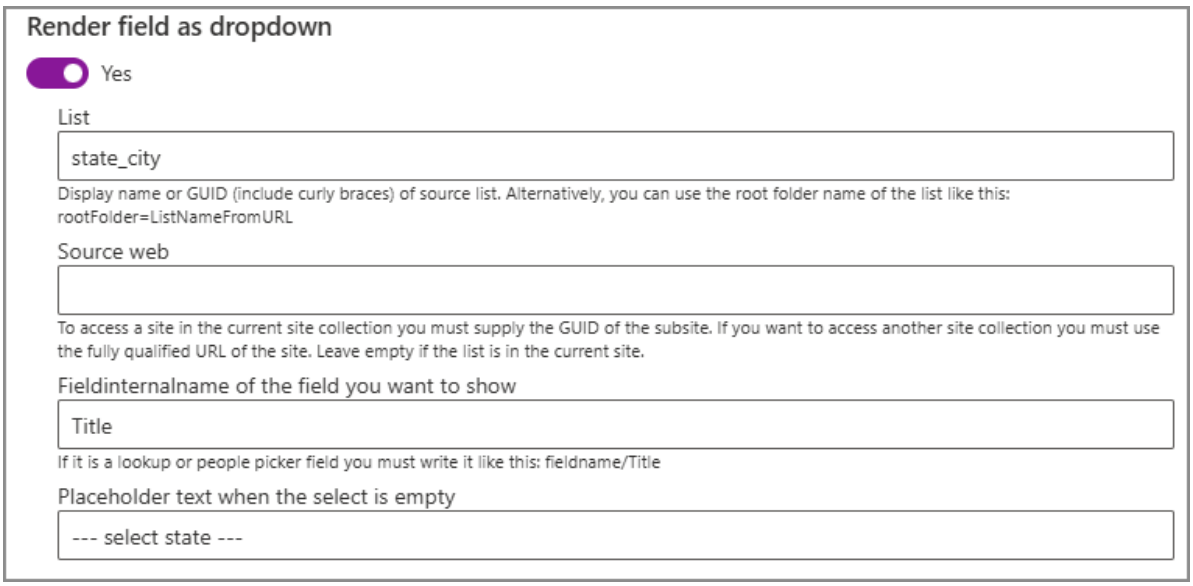
state_city

Tittel	field_1	field_2
Alabama	Tuscaloosa	Northport
Alabama	Tuscaloosa	Cottondale
Alabama	Tuscaloosa	Lake View
Arizona	Phoenix	Scottsdale
Arizona	Phoenix	Mesa
Arizona	Phoenix	Tempe
Arizona	Tucson	Oro Valley
Arizona	Tucson	Marana
Arizona	Tucson	Sahuarita
Arizona	Mesa	Gilbert

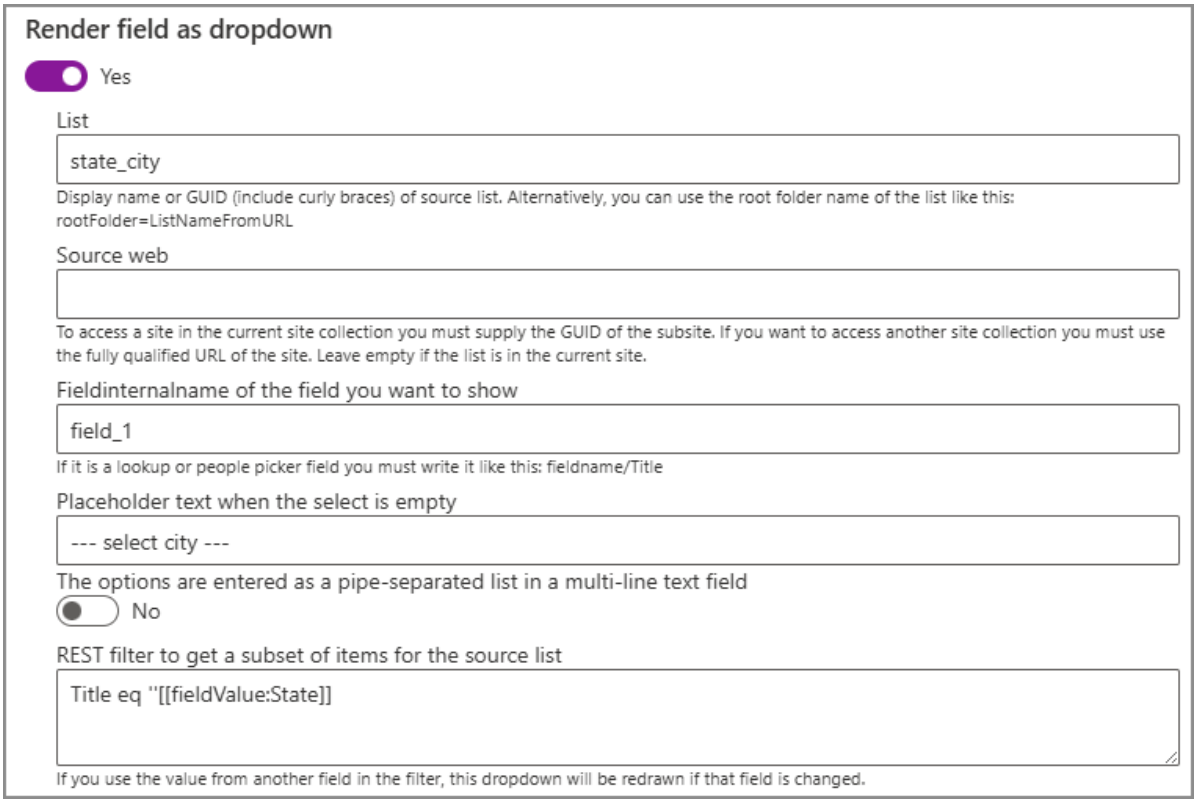
In the current list (where you want the cascading fields to appear) you must create two single-line of text fields ***State*** and ***City***, and one plain text multiline field ***Suburbs***.

Right-click the field and select ***Field properties*** and check the ***Render field as dropdown*** or ***Render field as autocomplete search box***.

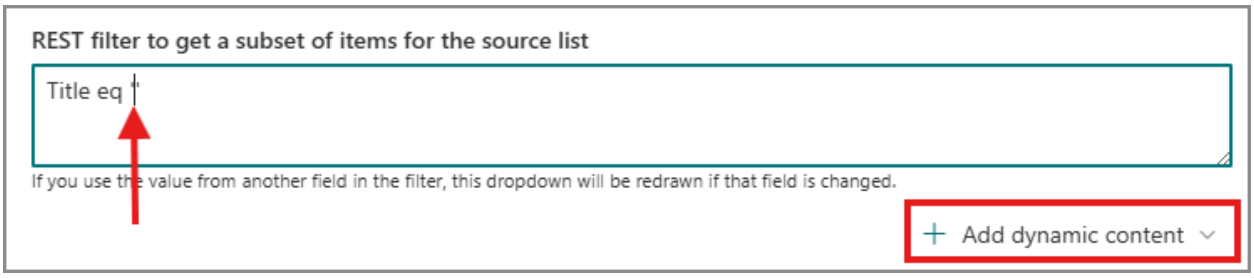
First level (State)



Second level (City)



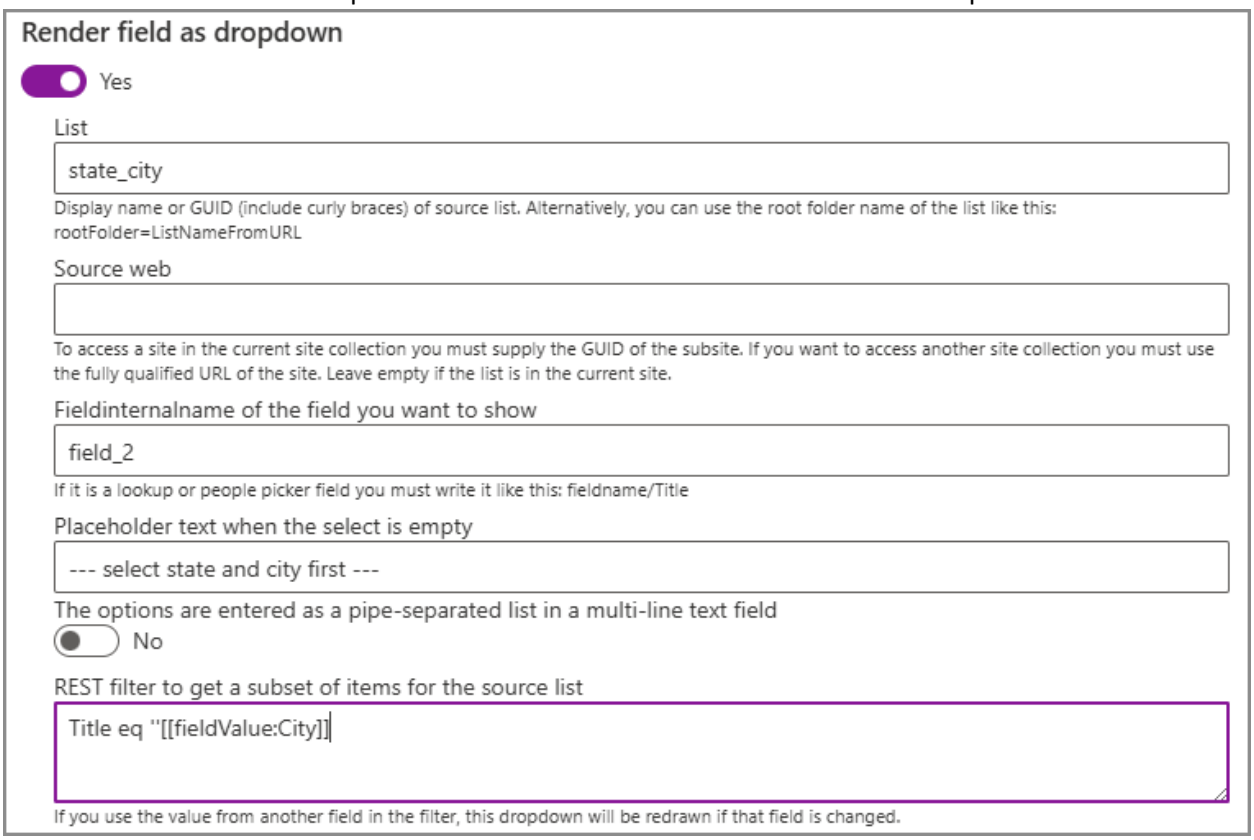
The Rest filter will ensure only items in the lookup list where the field ***Title*** equals the selection in the previous dropdown (State). The text ***[[fieldValue:State]]*** is inserted using the Add dynamic content menu that shown in the bottom right corner when focusing on the textarea.



Select ***Field value*** and then pick the State-field. Note the dynamic content menu will insert the value at the cursor (between the two single quotes in this example).

Third level (Suburbs)

The third is filtered based on the selection in the second one (City) and is rendered as an autocomplete search box. It will allow multiple selections because the Suburbs field is multiline plain text.



The Rest filter will ensure only items in the lookup list where the field field_1 equals the selection in the previous dropdown (City).

2.2.7.9 Render a single line of text field using a custom render function - advanced

A single line of text field can be rendered using a custom function if you want to add your own customized rendering.

Start by adding the field to your form and then open the field properties. Check the **Use a custom render function** and add the name of the function – for example *myTxtFieldRenderFn*.

Open the Custom JS tab and add a function with the name you specified. This example will result in a dropdown menu with two options. You can render any control you like if the value can be stored in a single line of text field.

```
function myTxtFieldRenderFn(f) {
 let options = [{ "val": "Option 1", "text": "Option one" }, { "val": "Option 2", "text": "Option two" }];
 let currFieldValue = getFieldValue(f.InternalName);
 let b = [];
 b.push("<select onchange='changeCustomDropdown(\"" + f.InternalName + "\", this)' style='padding: 6px 10px'>");
 b.push("<option value=''>---</option>");
 options.forEach(opt => {
  b.push("<option value='" + opt.val + "'" + (currFieldValue === opt.val ? " selected=true'" : "") + ">" + opt.text + "</option>");
 });
 b.push("</select>");
 return b.join("");
}

function changeCustomDropdown(fin, elm) {
 var value = elm.value;
 // Set the value to ensure it is saved with the form
 setFieldValue(fin, value);
}
```

2.2.7.10 Translate a choice field

If you work with multiple languages, you can translate choice field values in a form. The options are configured in list settings in one language, and can be translated by adding a variable like this in the Custom JS section like this:

```
var dffs_translate_field = {
 ChoiceField1: {
  Yes: 'Ja',
  No: 'Nei',
 },
 ChoiceField2: {
  'Option 1': 'Alternativ 1',
  'Option 2': 'Alternativ 2',
 },
};
```

The key is the fieldinternalname and the value is an object where the current value is mapped to the translated value. When saving the form, the default value (defined in list settings) will be saved.

2.3 Preview form

You can preview the form configuration from within the configuration. Some fields are rendered as read-only, and lookup columns rendered as treeview will have dummy content.

Please note that rules are not evaluated when previewing the form. Open a new tab and view the form outside of the configurator to get the full functionality with rules.

2.4 Rules

You can create rules to make your form dynamic. You can combine different triggers to create complex triggers by using a combination of and / or.

You can arrange rules in named groups for better overview. You can also copy a rule from one form to another by using the menu on the rule header.

You can use the form fields as trigger using these operators:

- is equal to
- is not equal to
- contains
- does not contain
- is greater than
- Is greater than or equal to
- is less than
- is less than or equal to
- starts with
- does not start with
- ends with
- does not end with
- is changed
- is changed from initial value
- is empty
- is not empty

Not all fields support all operators.

In addition to these triggers:

- Form loaded
- Before saving the form
- After saving the form
- Using a mobile device
- SharePoint group
- Selected tab
- Current form
- URL query string value
- Run based on the result of another rule
- Run based on the result of a Custom JS function (on form load)
- Run from a Custom JS function

When you have set up the trigger you must select the actions. You can set up different actions based on the result of the rule – **If yes** and **If No**.

You can select from these actions:

- Required fields
- Optional fields
- Visible fields
- Hidden fields
- Editable fields
- Readonly fields

- Show elements
- Hide elements
- Show / hide form buttons
- Call a function
- Set field value
- Prepare email
- Remove email
- Show / hide tabs
- Select tab
- vLookup readonly / editable
- vLookup required / optional
- Show a message box
- Show a field message
- Remove a field message

2.4.1 Run based on the result of a Custom JS function (on form load)

Add a function like this in Custom JS:

```
function checkFromRule(ruleId) {
  let result = false;
  // Add your custom logic here and set result to true or false to trigger the if yes or the if no section of the rule
  return result;
}
```

2.4.2 Run from a Custom JS function

When you select this trigger, you can see an example of how to trigger the rule. It takes two arguments; one is the Rule name or Rule ID and the second is a Boolean value to select the ***if yes*** or the ***if no*** part of the rule actions. The function call is like this (change the Rule name to match your rule).

```
dffs_triggerRule("your_rule_name_here", true);
```

Please note that only a rule configured with the trigger “Run from a Custom JS function” can be triggered like this.

2.5 Custom JS

In custom js you can add any JavaScript code you like. You can use SharePoint’s built in REST API to do CRUD (create, read, update or delete) operations.

You can load external *.js files or add Custom JS directly in the editor. Please note that Microsoft has enforced stricter rules regarding loading external js-files on March 1, 2026. This means that if you want to refer to a js-file from an external source (outside of your tenant), the source must be added to the Trusted script locations in the SharePoint Admin-section on the Tenant.

2.5.1 Some examples

2.5.1.1 Check the previous value of a field

To check the previous value (the value the item had in a field when you opened it) you can use the built in dffs_beforeProperties variable like this:

```
dffs_beforeProperties.NameOfField
```

You can check the change history (in the current editing session) by using this variable:

```
dffs_fieldValueHistory.NameOfField;
```

2.5.1.2 Get the list context of the current form

You can get the context of the current list by using this variable:

```
dffs_FormContext.NameOfList
```

This will give you the listId, listTitle, ListUrlName, mode, serverRelativeUrl and the folderPath. The object only holds the context of the list(s) currently being used (the main list and if you open a child form in a panel, the child list).

2.5.1.3 Select a tab in a tabset from custom js

To select a tab in a tabset using custom js you can use this function:

```
dffs_selectTab("tab_6405371368156925");
```

2.5.1.4 See the current users' group membership and user profile

You can use this variable to see the logged in users' membership in SharePoint groups:

```
dffs_userGroups
```

You can use this variable to see the logged in users' user profile:

```
dffs_userProfile
```

2.5.1.5 Print the form

Because of how the DFFS form is rendered, printing the form using the default print functionality in the browser does not work 100%. To print properly you must add a button to your form using a HTML section with code like this:

```
<div style="text-align:right" class="dffs_no_print">
<span style="font-size:2em;cursor:pointer;" id="dffs_print_form_btn">Print</span>
</div>
```

To add the click event handler, you must then add this snippet to Custom JS:

```
dffs_addClickEventListener('dffs_print_form_btn', () => {
  dffs_print();
});
```

You can replace the text Print with an image tag with your preferred print icon.

2.5.1.6 Check the status of a rule from Custom JS

You can see the state of a rule using this object:

```
dffs_ruleState["Name_of_rule"];
// by id
```

```
dffs_ruleState["id_of_rule"];
```

Please note that only rules that have run will show here. If your rule triggers on change of a field, it will not show until the rule have been triggered.

2.5.1.7 Run code when the form is loaded

If you want to run some custom js when the form is ready, add a function like this to your Custom JS:

```
function dffs_ready(){
 // Custom code
}
```

2.5.1.8 Do something before the item is saved

You can use a function like this to run your own custom code before the form is saved. The function must return true to allow save, or false to stop the save. If you use this function it will run before the rules triggering on “Before save of the form” and “After save of the form”. If your function returns false, these rules will not run.

```
function dffs_PreSaveAction(exit, autosave, goToEditForm) {
 if (exit) { // Code only runs when you use SaveAndExit
  // This example will prevent save if the title field was NOT changed from the previous value
  if (dffs_beforeProperties.Title === getFieldValue("Title")) {
   alert("Title has not changed!");
   return false;
  }
  // Allow save
  return true;
 } else {
  if (autosave) {
   console.log("AutSaving");
  } else {
   console.log("QuickSaving");
  }
 }
}
```

You can mark this function as async in case you must do some asynchronous calls before saving the item.

2.5.1.9 Save the item from Custom JS

Call this function from Custom JS to save the form:

```
dffs_triggerSave(true); // Save and exit
dffs_triggerSave(false); // Quick save
```

2.5.1.10 Do something after the item is saved

You can use a function like this to run your own custom code if a form is opened and then cancelled out of without saving.

```
function dffs_PostSaveAction() {
 // Do something after the item is saved - for example redirect to another page
 // You can read values from the current item using getFieldValue("Name_of_field")
 location.href = "https://contoso.sharepoint.com/sites/your_site/SavedMsg.aspx";
}
```

2.5.1.11 Do something if the item is cancelled / closed

You can use a function like this to run your own custom code if a form is opened and then cancelled out of without saving.

```
function dffs_PostCancelAction() {
 // Do something if the item is cancelled - for example redirect to another page
 location.href = "https://contoso.sharepoint.com/sites/your_site/CancelledMsg.aspx ";
}
```

2.5.1.12 Cancel the form from Custom JS

If you want to close / cancel out of the current form you can use this function

```
dffs_cancelForm()
```

2.5.1.13 Do something if the item is deleted

You can use a function like this to run your own custom code after an item is deleted (from within a form, not from the list view).

```
function dffs_PostDeleteAction() {
 // Do something if the item is deleted - for example redirect to another page
 location.href = " https://contoso.sharepoint.com/sites/your_site/DeletedMsg.aspx";
}
```

2.5.1.14 Show a modal dialog

To show a modal dialog with a custom message you can use this code. Please note that this will be a non-blocking dialog so it will not prevent other code from running when the dialog is shown (like the default browser “alert” function does).

```
var modalId = dffs_showModal({
 "title": "A message to you",
 "body": "Add your message body here. You can use plain text or HTML",
 "isBlocking": true,
 "showClose": true,
 "showFooter": true,
 "closeCallbackFn": () => {
  // Add your custom code that runs when the dialog has been dismissed
  alert("Closed");
 },
 "cancelBtnLabel": "Cancel button",
 "cancelBtnFn": () => {
  // Add your custom code to run when clicking the cancel button
 },
 "okBtnLabel": "OK Button",
 "okBtnFn": () => {
  // Add your custom code to run when clicking the OK button
 }
});
```

To hide a modal from Custom JS you can call this function with the id of the modal:

```
dffs_hideModal(modalId);
```

2.5.1.15 Trigger custom code when a field is changed

Use a function like this to run custom code when a field is changed.

```
function dffs_fieldChanged(fin, new_value, previous_value) {
  console.log(`The field '${fin}' was changed from '${previous_value}' to '${new_value}'`);
}
```

2.5.1.16 Add a callback when a vLookup has loaded

You can use this method to run your own custom code when a vLookup has been loaded. You can find the id of the vLooup in the vLookup configurator.

```
function dffs_vLookup_callback(id, items){
  if(id === "vLookup_7943515991751215"){
    console.log(id, items);
  }
}
```

2.5.1.17 Reload the vLookup table from Custom js

If you want to reload the vLookup table from custom js you can call this function with the id of the vLookup.

```
vLookup_loadItems("vLookup_40882006992542896").then(items => {
    console.log(items);
});
```

2.5.1.18 Open a vLookup list item from Custom JS

You can open a list item in a vLookup table from custom js like this

```
vlookup_open('target_vLookupId', 'edit', item_id);
```

2.5.1.19 Set a vLookup are required from custom js

```
vLookup_toggleRequired("vLookup_7993792841490628", true); // true = required, false = optional
```

Please note that you can set a vLookup as required / optional using a rule.

2.5.1.20 List the attachments on a list item

You can show a list of the attachments on a list item using this variable in custom js:

```
dffs_attachments
```

You can find more code examples in the forum here: https://spjsblog.com/forums/topic/custom-js-examples/

2.6 Custom CSS

You can add your own Custom CSS style your custom HTML sections or to override any styling on your form.

You can load external CSS files by writing this in the Custom CSS field:

```
@import "/path_to_file/filename.css";
```

An auto-generated suffix is added to all default classes so if you want to overide for example the formLabel class you must write the CSS selector like this: div[class^='formLabel_']

2.7 Miscellaneous

This tab contains various settings.

2.7.1 Password

You should set a password to protect your configuration from others.

2.7.2 Override strings used in DFFS

Use this to override individual strings in the UI of DFFS. There is a link below the textarea where you can see all strings you can override.

2.7.3 Form panel type (only for vLookup child forms)

Choose from the following list of form types:

- Full width
- Large
- Medium
- Small

2.7.4 Show panel type menu in the form (only for vLookup child forms)

Use this setting to show a form panel-width selector in the form.

2.7.5 Currency symbol position for currency fields

Use this to place the currency symbol left or right on currency fields.

2.7.6 Comments

Check the **Show comments button in the top right corner of display form and edit form** to let the users comment on the list items. This uses the out-of-the-box SharePoint comment functionality.

2.7.7 Use Jodit Editor

Toggle this on to use a more advanced rich text editor with rich text fields in the form.

2.7.8 Show an icon to the left of the field label

Toggle this on to show an icon to the left of the field label in your form.

2.7.9 Show option to quick-save the form without closing it

Toggle this on to show a button to quick save without closing the form. Please note that emails and rules triggered before or after saving will only be processed when the form is manually saved and closed and not when the form is autosaved or quick saved.

2.7.10 Show button to turn on automatic saving in the footer of the form

Toggle this on to show a toggle button at the bottom of the form where auto-saving can be turned on. Please note that emails and rules triggered before or after saving will only be processed when the form is manually saved and closed and not when the form is autosaved or quick saved.

2.7.11 Tab colors

You can set the default colors for the tab sets you add to the form.

2.7.12 Load the configuration from another site or from a file

This feature lets you load the configuration from another site in your tenant, or from a file. This can be used when you have multiple sites created from a template, and you want to manage the configuration for all forms centrally.

You can find more information in the contextual information text in the Miscellaneous tab.

In addition to these two options that must be applied to each form you can also set a general "Load all configurations from another site" for all lists in the site by manually adding an item to the configuration list (see 2.1 for information on how to access the list).

Set the Title field to "LoadAllConfigsFromOtherSite" and the Misc field to the fully qualified URL of the site where the configurations are stored.

Please note that this is the URL to the site and not to the list. For example: https://contoso.com/sites/your_site

2.8 Note to self

This tab is for internal notes, helping you track your progress while building the form.

2.9 Install DFFS

Use this tab to activate DFFS as the renderer of forms in this list or library.

3 Linking to a DFFS form

If you want to create a custom link to a form you can use this format:

https://contoso.sharepoint.com/sites/YourSite/_layouts/15/SPListForm.aspx?PageType=4&List=[list_guid]&ID=[item_id]

The PageType attribute specifies the form type where 8 = NewForm, 6 = EditForm and 4 = DispForm. The ID attribute is not used for PageType 8 (NewForm).

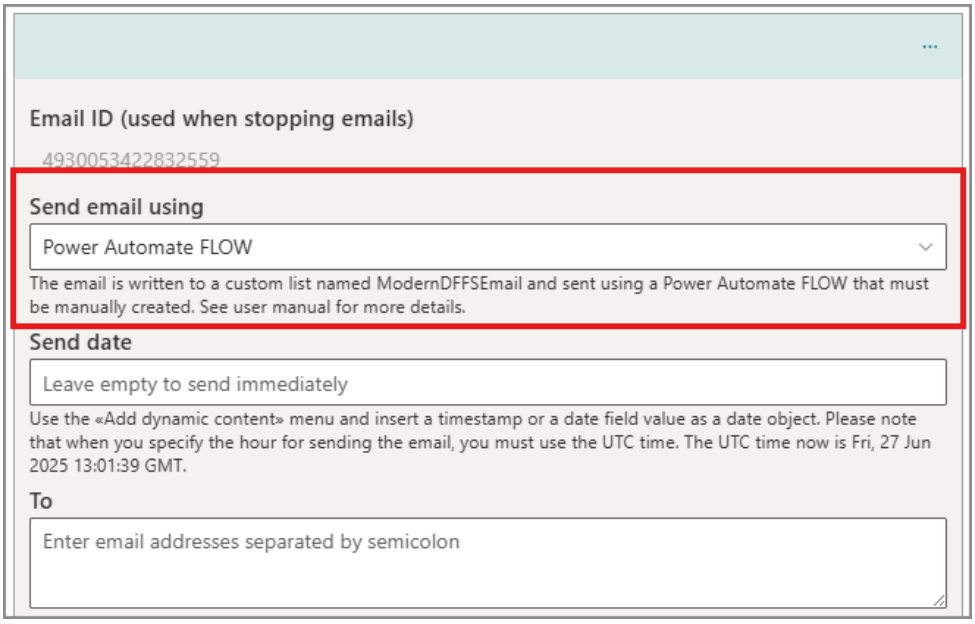
You can use the _DFFSID property to link to a form (instead of using the itemID). This can be used if you, for example, use vLookup to link a parent and a child and you want to create a link back to the parent item from the child. The format is the same as the default link, but the ID is replaced with DFFSID like this:

https://contoso.sharepoint.com/sites/YourSite/_layouts/15/SPListForm.aspx?PageType=4&List=[list_guid]&DFFSID=[DFFSID]

4 Sending emails using FLOW

Microsoft has deprecated the SendEmail API and therefore a new option has been added when sending emails from rules.

When you configure an email in a rule action, select ***Send email using: Power Automate FLOW*** like this:



When you first select this option in a site, a new custom list named ***ModernDFFSEmail*** will appear in ***Site contents***. When using the ***Send email using*** option, all emails will be added to this list.

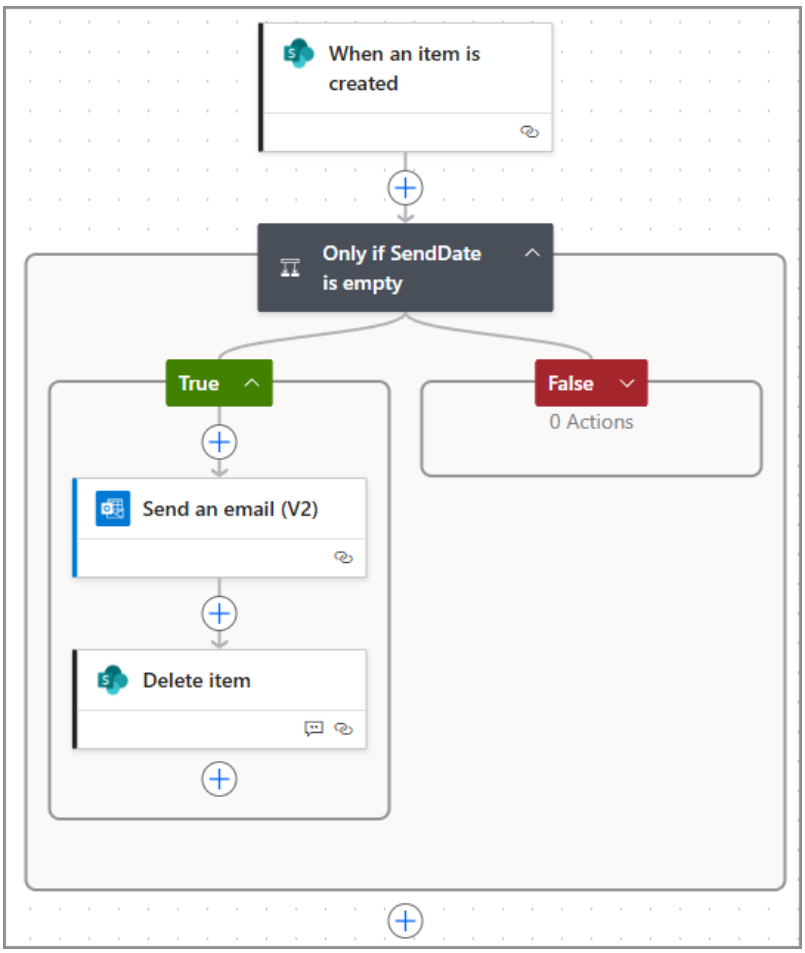
The sending of the emails is be done using a Power Automate FLOW that must be manually created and attached to the list.

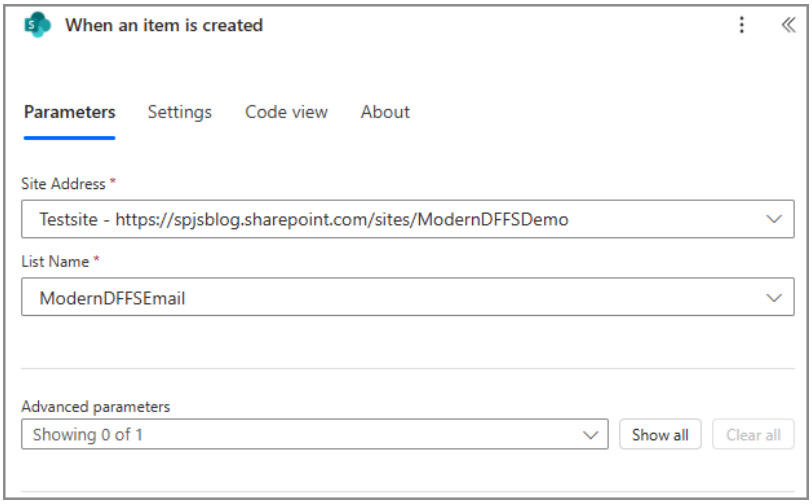
The example will show two examples, one for ***Send now*** (when SendDate is empty) and one for ***Send later*** (when SendDate contains a date). The Send later FLOW processes items with a send date (the ***Send date*** field in the email configuration is only visible when you use the ***Send email using: Power Automate FLOW*** option.

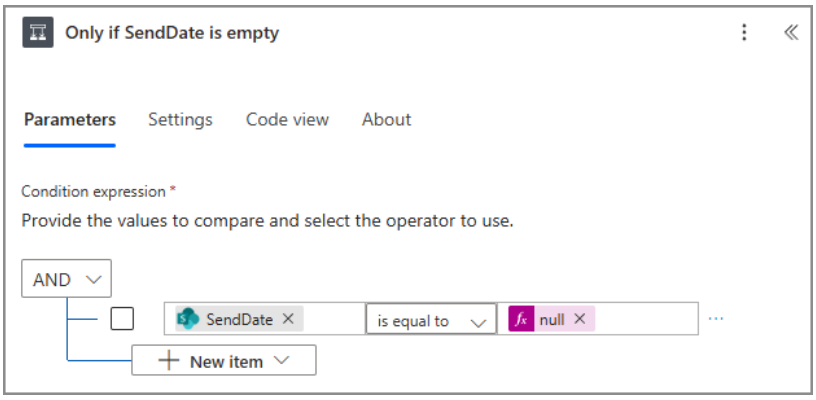
4.1 FLOW configuration for Send now

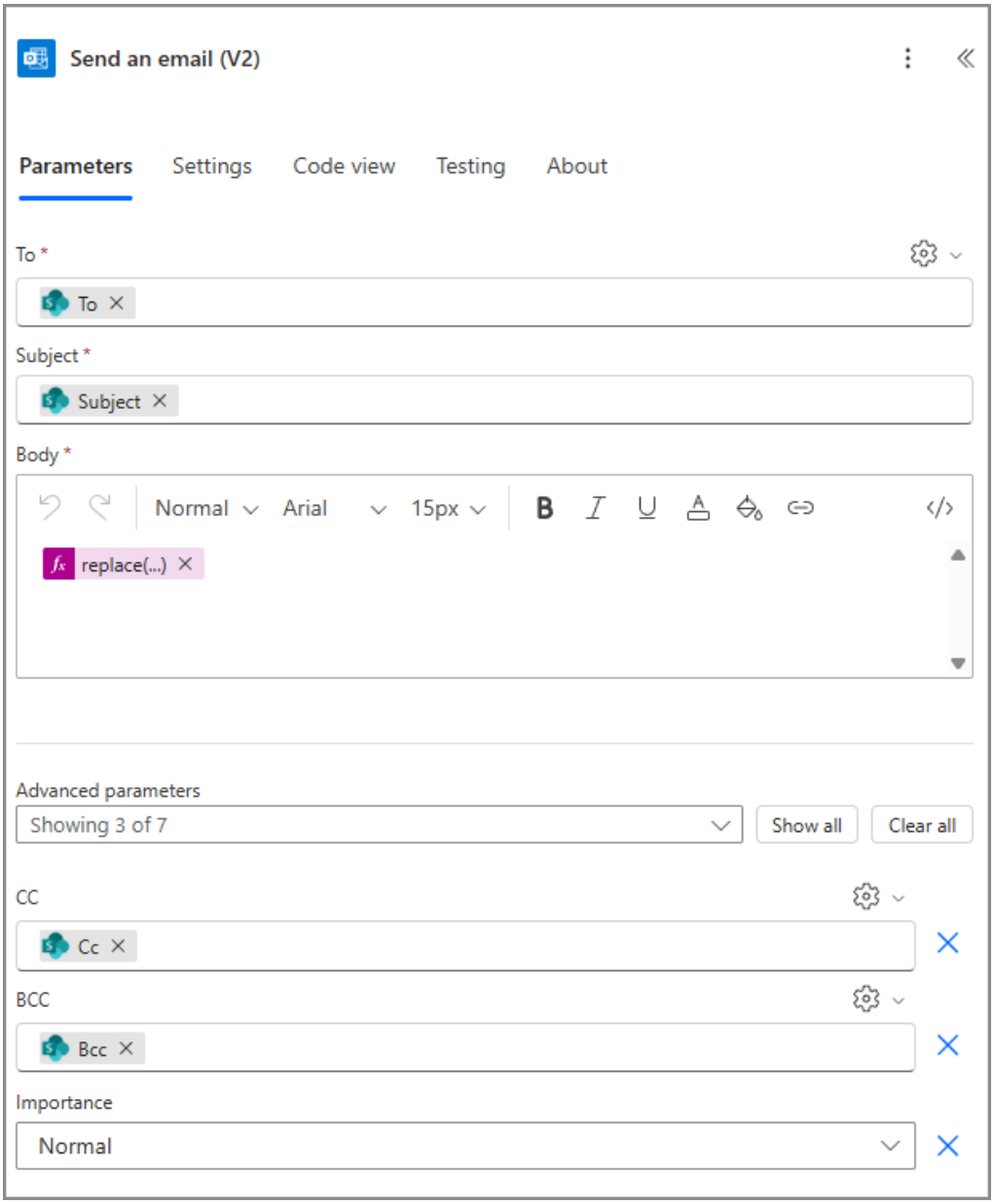
This FLOW is an ***Automated cloud flow*** that is set to trigger when a new item is added to the ***ModerDFFSEmail*** list.

Here are screenshots of all the steps in this FLOW.


When an item is created
Only if SendDate is empty
True
False
0 Actions
Send an email (V2)
Delete item

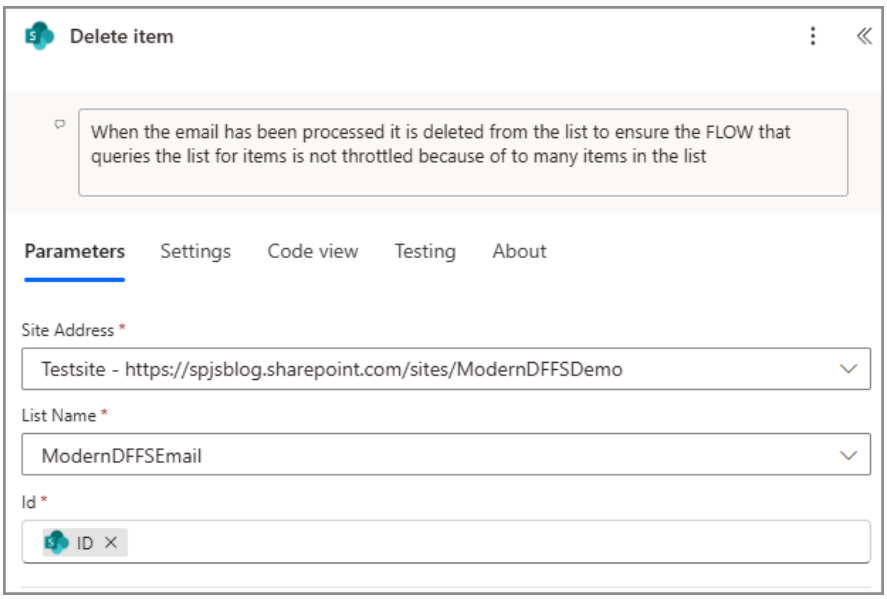

When an item is created
Parameters
Settings
Code view
About
Site Address *
Testsite - https://spjsblog.sharepoint.com/sites/ModernDFFSDemo
List Name *
ModernDFFSEmail
Advanced parameters
Showing 0 of 1
Show all
Clear all


Only if SendDate is empty
Parameters
Settings
Code view
About
Condition expression *
Provide the values to compare and select the operator to use.
AND
SendDate
is equal to
null
New item



To replace linefeed in Body with <br> you must insert this snippet by clicking the fx button:

```
replace(triggerBody()?['Body'],decodeUriComponent('%0A'), '<br>')
```

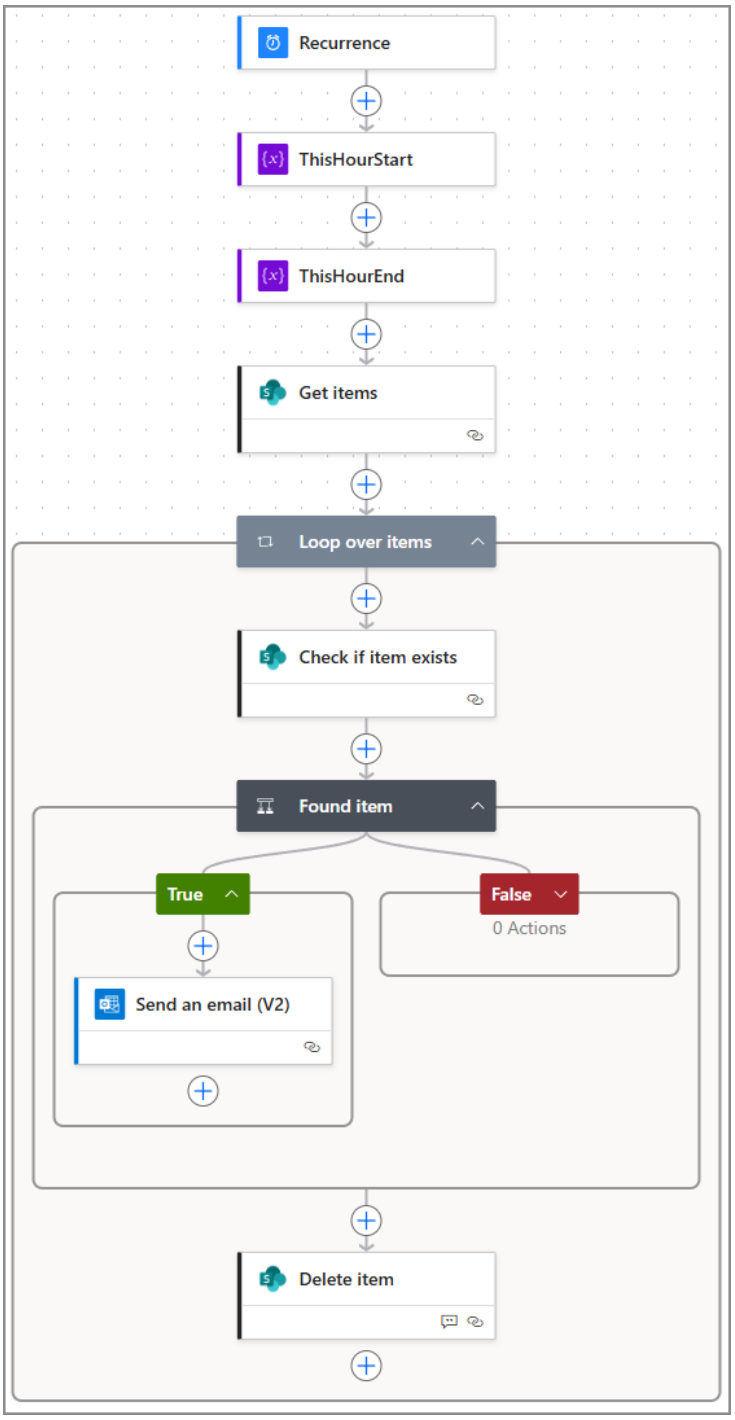

Delete item
When the email has been processed it is deleted from the list to ensure the FLOW that queries the list for items is not throttled because of to many items in the list
Parameters
Settings
Code view
Testing
About
Site Address *
Testsite - https://spjsblog.sharepoint.com/sites/ModernDFFSDemo
List Name *
ModernDFFSEmail
Id *
ID

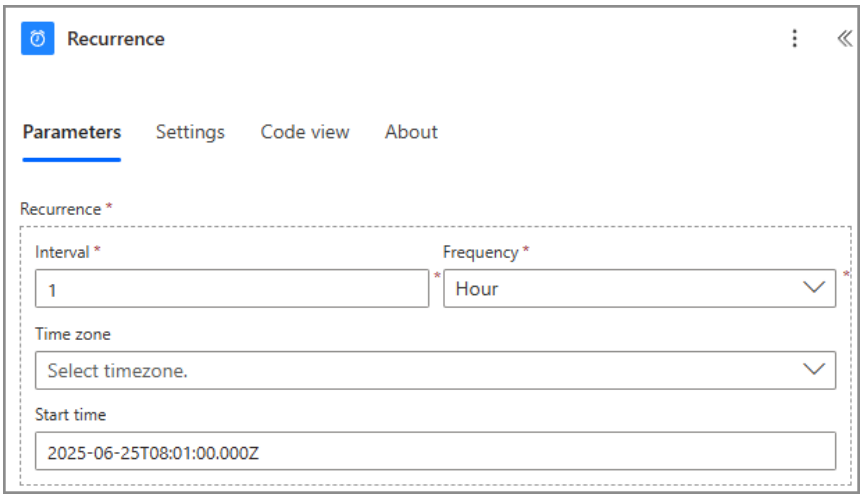
4.2 FLOW configuration for Send later

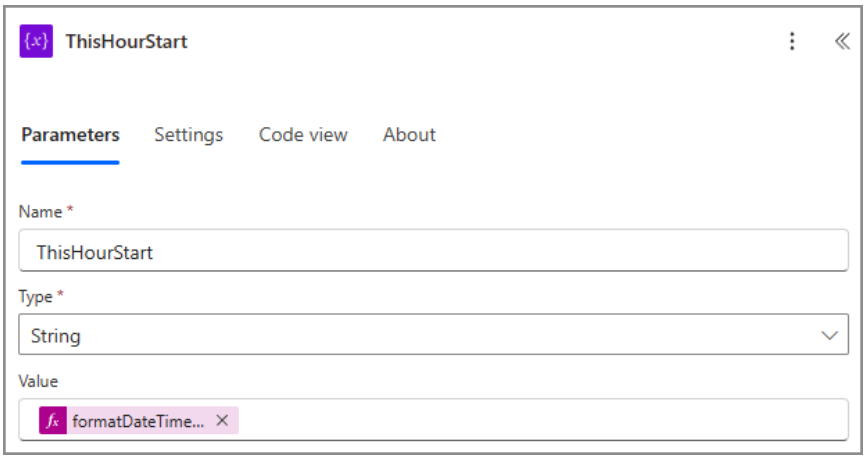
This FLOW is a ***Scheduled cloud flow*** that is set to run every hour, looking for items in the ***ModerDFFSEmail*** list where the SendDate and time (hour) match the current time.

Please note that it uses UTC time so the email Send date field must use UTC time.

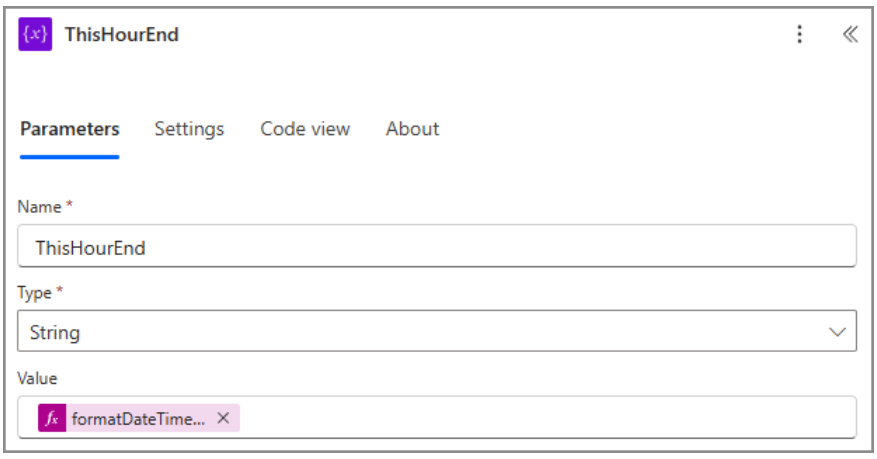
Here are screenshots of all the steps in this FLOW.



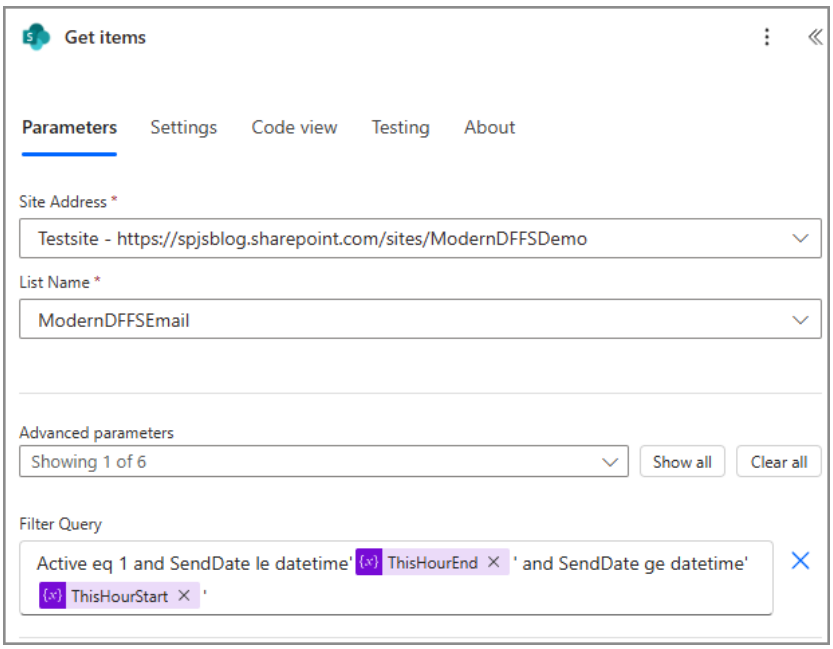


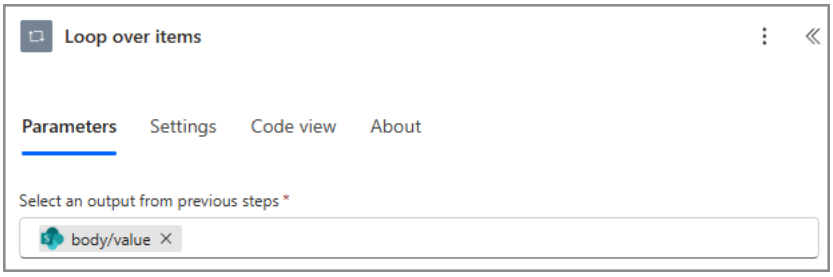


fx: formatDateTime(utcNow(),'yyyy-MM-ddTHH:00:00')



fx: formatDateTime(utcNow(),'yyyy-MM-ddTHH:59:59')





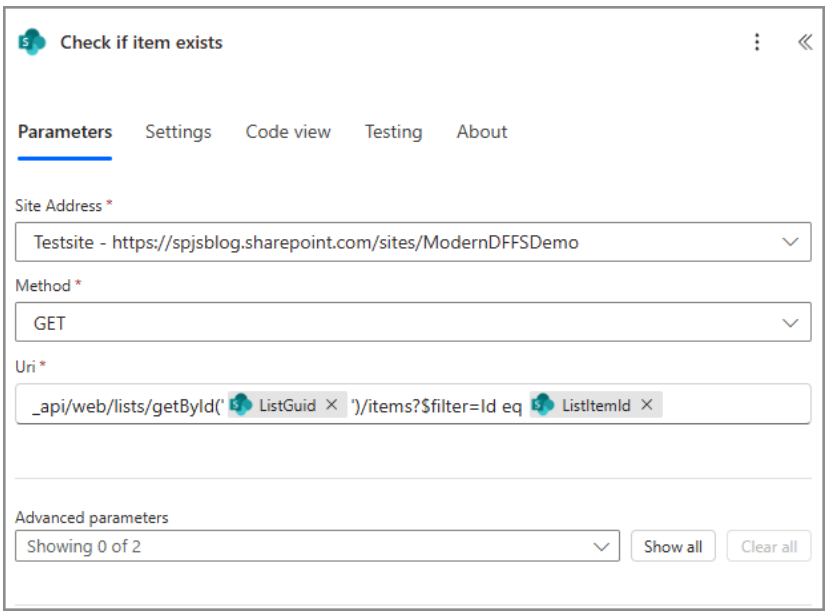
The ***Check if item exists*** is used to ensure that the parent item has not been deleted after this email was scheduled. It uses the action ***Send an HTTP request to SharePoint***.

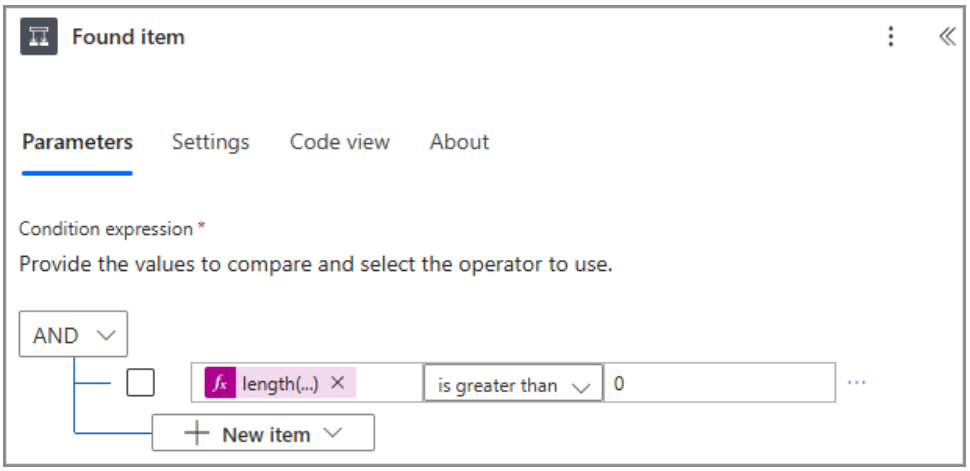
The parameter ListGuid must be inserted using fx like this:

```
item()?['ListGuid']
```

The parameter ListItemId must be inserted using fx like this:

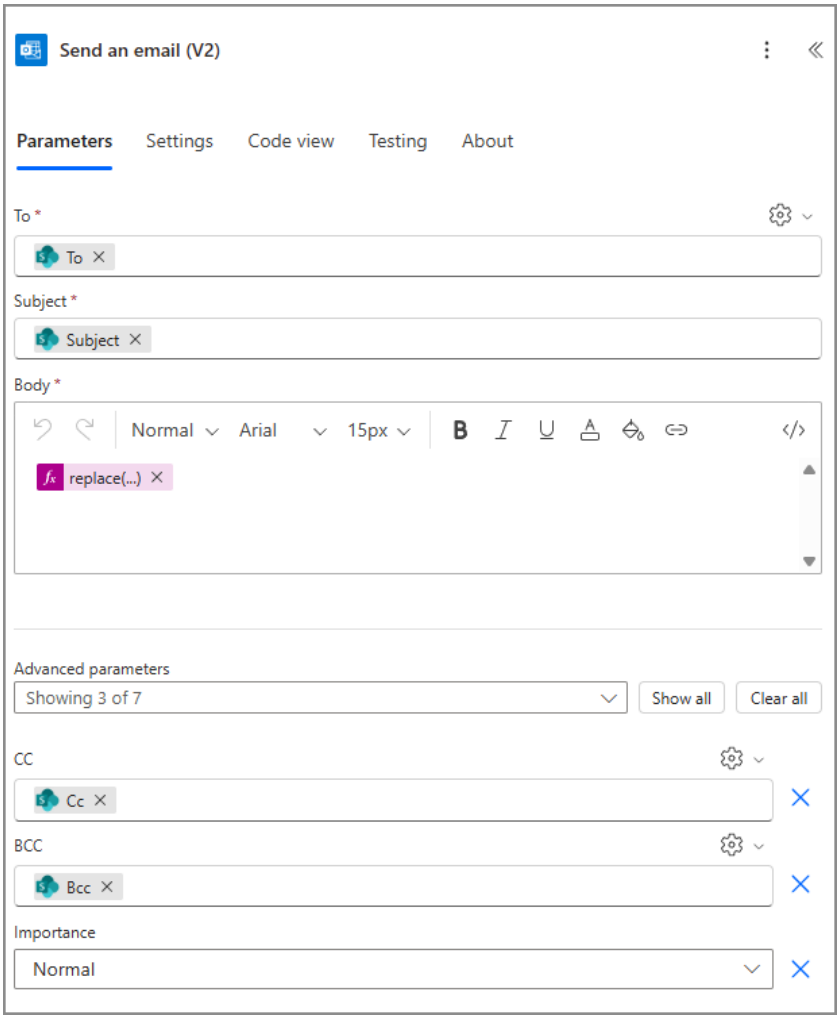
```
item()?['ListItemId']
```





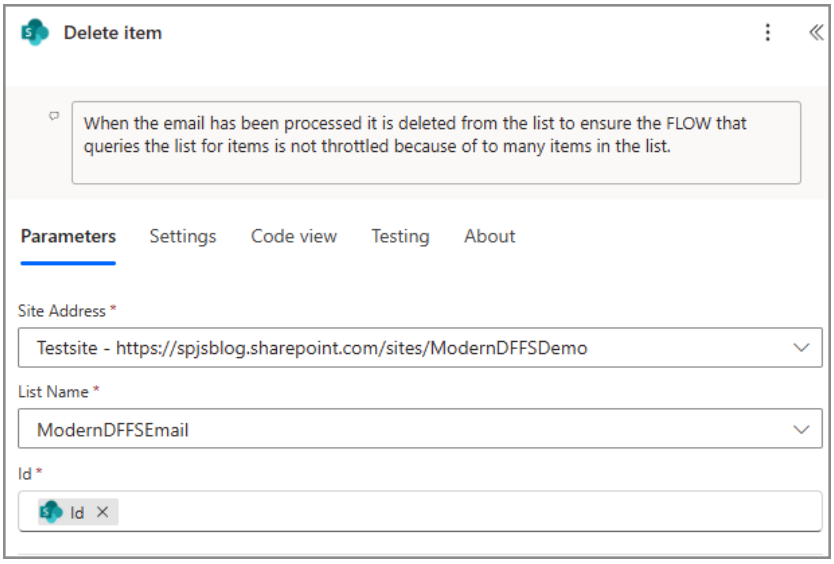
The condition must be written using fx like this:

```
length(body('Check_if_item_exists').d.results)
```



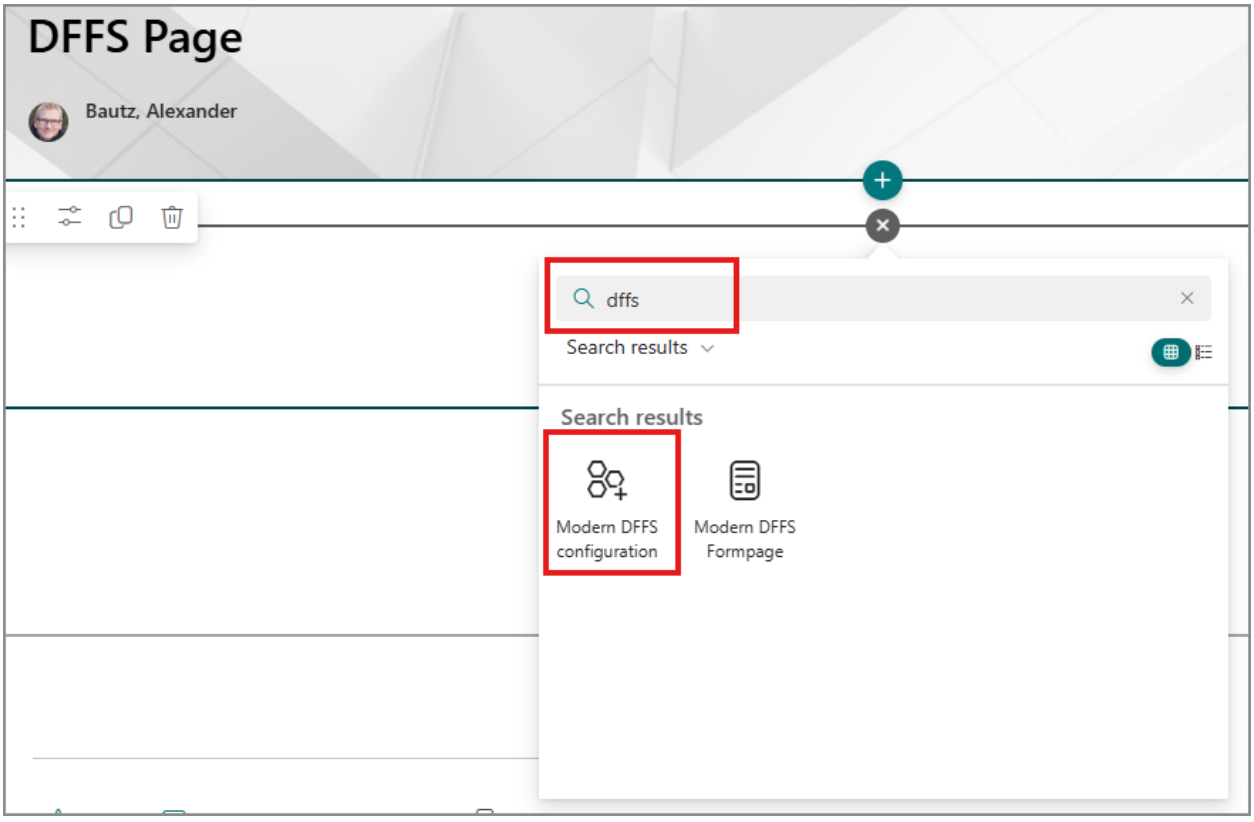
To replace linefeed in Body with <br> you must insert this snippet by clicking the fx button:

```
replace(item()?['Body'],decodeUriComponent('%0A'), '<br>')
```

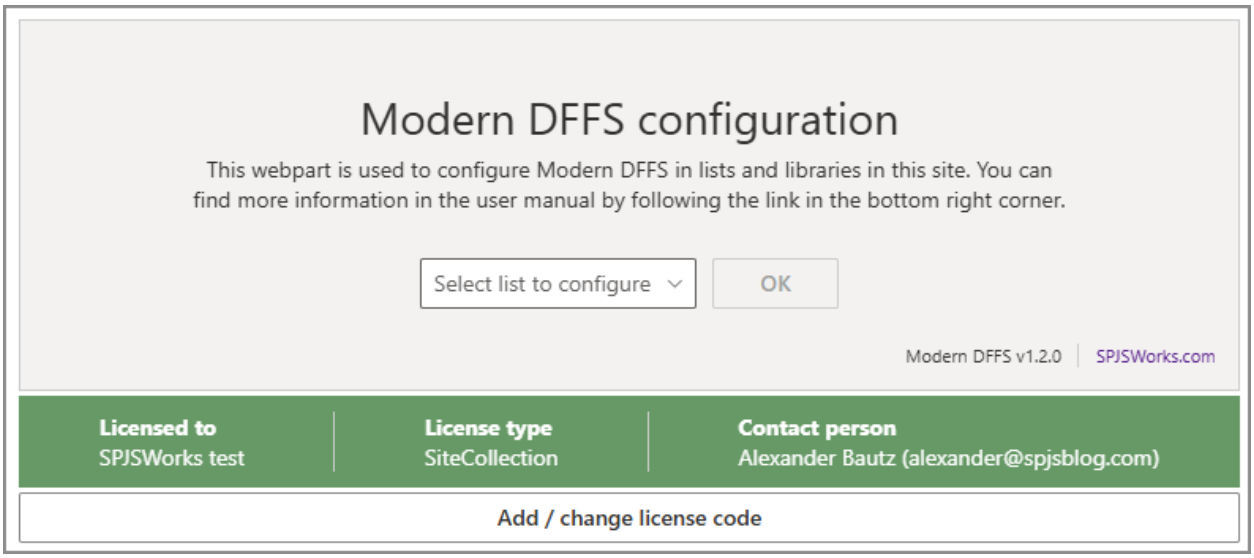


5 DFFS Configuration WebPart

Because Microsoft has some issues with showing the banner buttons associated with a list view command set (which is the original method Modern DFFS is loaded in a list view) I have decided to create a separate DFFS configuration WebPart. This webpart can be added to a modern site page (in site pages) like this:



The web part looks like this:



It lists all lists and libraries in the current site, but when selecting a list, it will check if the user has manage-lists-permissions to the list before proceeding.

6 Modern DFFS Formpage Web Part

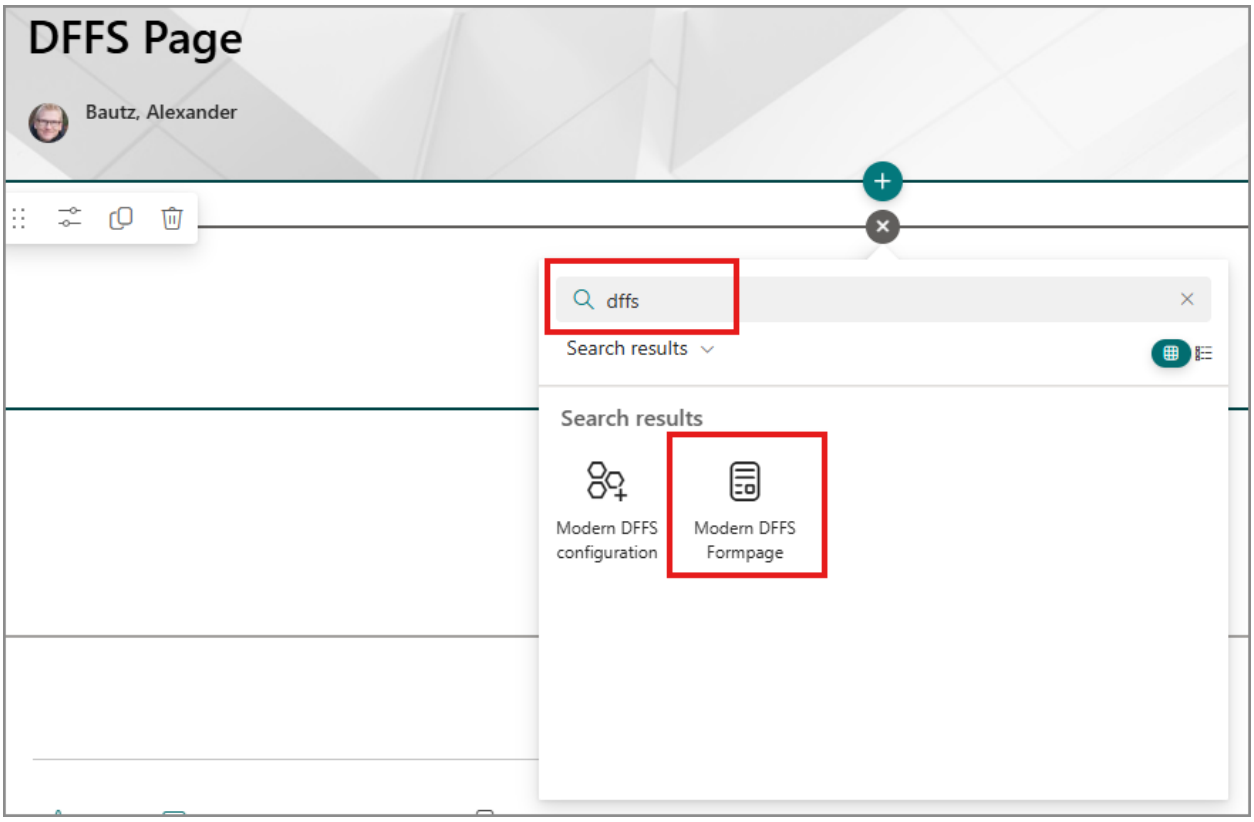
Requires Modern DFFS solution v1.0.14.0 or later.

This web part is used to show a Modern DFFS form on a site page. This can be handy if you want to show a new item form, or you want to let users edit a list item by for example passing them a link to an item in an email.

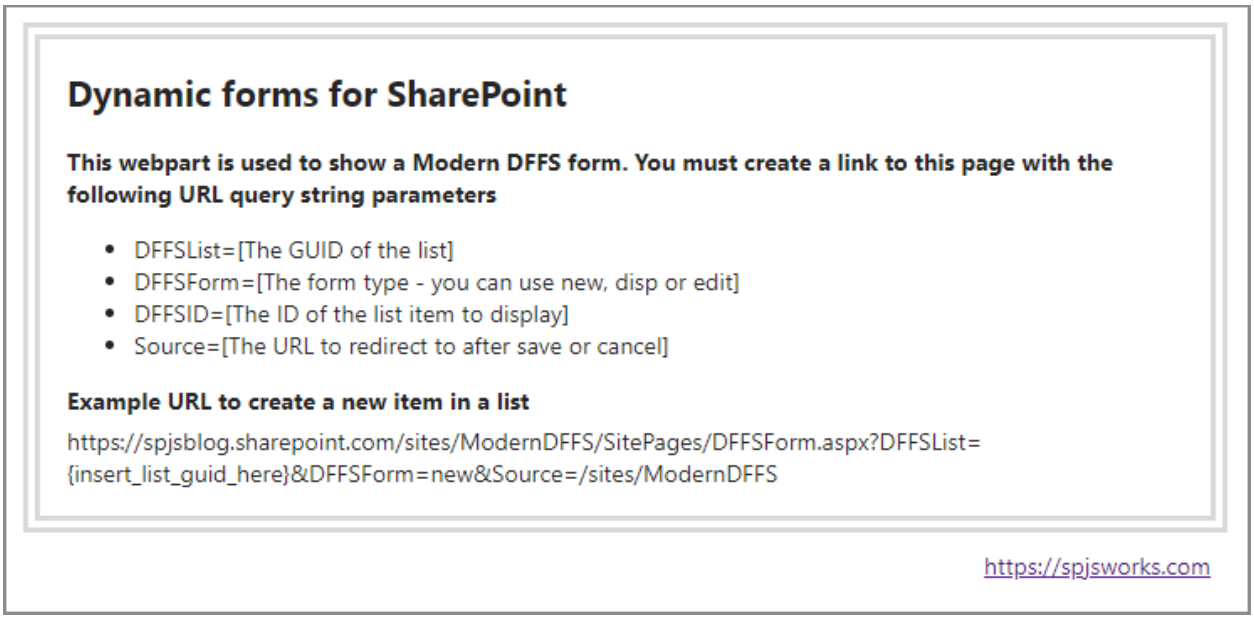
You can use the Moder DFFS Formpage Web Part to add, view or edit items in a list that has Modern DFFS Forms configured. To use it you must pass the **DFFSList** (GUID of the target list), the **DFFSForm** (new, disp or edit) and the **DFFSID** (if you want to display or edit an existing item). You can also add a **Source** attribute to have the user redirected to that address after they cancel or save the form.

See instructions when adding the web part.

Add the web part like this:



When you have added it, you can read the instructions on how to use it in the placeholder:



It is currently not possible to use this formpage when adding, viewing or editing a list item from a list view – only by using a custom link created on the format described above.

6.1 Setting a default redirect in the form URL

When using the Modern DFFS Formpage webpart you can use an URL parameter to set a default redirect. This can still be overridden in Custom JS if you use the function dffs_PostSaveAction or dffs_PostCancelAction.

Create the URL like this

https://spjsblog.sharepoint.com/sites/ModernDFFS/SitePages/DFFSForm.aspx?DFFSList={insert_list_guid_here}&DFFSForm=new&DFFSSource=/The_url_to_redirect_to

7 Upgrade from Classic DFFS

Use the ***Import / Export*** button to open the ***Import, export and restore configurations*** panel.

If you have the Classic DFFS version already in use on this site, you can browse the list of configurations and import them directly.

If you don't have the Classic DFFS version on this site, you must use the export functionality in the Classic DFFS and the import functionality in the Modern DFFS. This must be done in one form at the time

Please note that because the Classic DFFS and the Modern DFFS are built with completely different technologies, the Custom JS and Custom CSS parts will not be imported. You can do all the same things in Custom JS and Custom CSS in the Modern version, but code that interacts with SharePoint lists (using CRUD) must be rewritten using the REST API because the *spjs.dffs* and *spjs.utility* namespaces are not available.

There are several functions built into the Modern DFFS that can be used, but the format is a bit different than in the Classic version. You can find details about these functions in the forum here: https://spjsblog.com/forums/topic/custom-js-examples/

Tabs and rules are imported, but because the rule configuration is a bit different you must manually go through the rules to ensure that they do what you want them to do.

Please note that you must manually uninstall the Classic DFFS version on the list before configuring the list with the Modern DFFS to avoid conflicts.

8 Troubleshooting

8.1 Blocked web resources

If you are behind a company firewall you might have trouble loading some of the resources used in the Modern DFFS.

The license is loaded from an Azure CDN on this address: https://spjs.blob.core.windows.net. To be able to use the Modern DFFS you must also unblock this resource.

8.2 Other blocked resources

Because firewall rules are individually set up for different companies there might be other resources that must be unblocked in your company. If you have trouble loading the Modern DFFS you can use the developer tools to look for error messages in the Console or the Network tab.

If you have any questions, please contact support: https://spjsworks.com/support