Di Halaman Ini

Platform WhatsApp Business

Components

Components are like building blocks. They allow you to build complex UIs and display business data using attribute models. **The maximum number of components per screen is 50.** Please refer to best practices for components.

The following components are supported:

- Basic Text (Heading, Subheading, Caption, Body)
- RichText
- TextEntry
- CheckboxGroup
- RadioButtonsGroup
- Footer
- OptIn
- Dropdown
- EmbeddedLink
- DatePicker
- CalendarPicker
- Image
- If

- Media upload
- NavigationList
- Chips Selector
- Image Carousel

Text Components

Heading

This is the top level title of a page.

Parameter	Deskripsi
`type` **(required)** *string*	"TextHeading"
`text` **(required)** *string*	Dynamic "${data.text}"
`visible` *Boolean*	Dynamic "${data.is_visible}" Default: True

Subheading

Parameter	Deskripsi
`type` **(required)** *string*	"TextSubheading"
`text` **(required)** *string*	Dynamic "${data.text}"
`visible` *Boolean*	Dynamic "${data.is_visible}" Default: True

Body

`type` **(required)** *string*	TextBody
`text` **(required)** *string*	Dynamic "${data.text}"
`font-weight` *enum*	{'bold','italic','bold_italic','normal'} Dynamic "${data.font_weight}"
`strikethrough` *Boolean*	Dynamic "${data.strikethrough}"
`visible` *Boolean*	Dynamic "${data.is_visible}" Default: True
`markdown` *Boolean*	Default: False Requires Flow JSON V5.1+

Caption

Parameter	Deskripsi
`type` **(required)** *string*	"TextCaption"
`text` **(required)** *string*	Dynamic "${data.text}"
`font-weight` *enum*	{'bold','italic','bold_italic','normal'} Dynamic "${data.font_weight}"
`strikethrough` *Boolean*	Dynamic "${data.strikethrough}"
`visible` *Boolean*	Dynamic "${data.is_visible}" Default: True

`markdown` *Boolean*	Default: False Requires Flow JSON V5.1+

Limits and Restrictions

Component	Type	Limit / Restriction
Heading	Character Limit	80
Subheading		80
Body		4096
Caption		409
Heading	Text	Empty or Blank value is not accepted
Subheading		
Body		
Caption		

Additional capabilities for Text components

Supported starting with Flow JSON version 5.1

markdown syntax within these components.

```
{
   "type": "TextBody",
   "markdown": true,
   "text": [
     "This text is ~~***really important***~~",
   ]
}
```

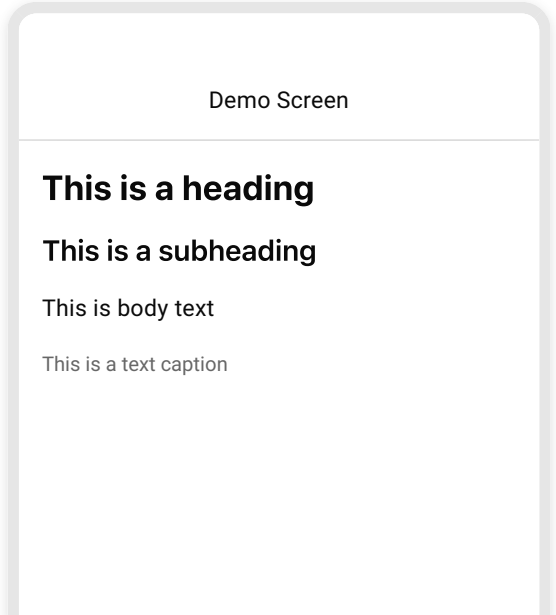
```
{
   "type": "TextCaption",
   "markdown": true,
   "text": [
     "This text is ~~***really important***~~",
   ]
}
```

For comparison purposes, we show how the text components look like next to one another:

Flow JSON

```
{
  "version": "7.3",
  "screens": [
    {
      "id": "DEMO_SCREEN",
      "title": "Demo Screen",
      "terminal": true,
      "layout": {
        "type": "SingleColumnLayout",
        "children": [
          {
            "type": "TextHeading",
            "text": "This is a heading"
            "visible": true
          },
          {
            "type": "TextSubheading",
            "text": "This is a subheadi
            "visible": true
          },
          {
            "type": "TextBody",
            "text": "This is body text"
          },
          {
            "type": "TextCaption",
```

Preview Run



```
      "type": "Footer",
      "label": "Continue",
      "on-click-action": {
        "name": "complete",
        "payload": {}
      }
    }
```

Rich Text

> Supported starting with Flow JSON version 5.1

Flow JSON 5.1 introduces a new component - `RichText`. The goal of the component is to provide rich formatting capabilities and introduce the way to render large texts (**Terms of Condition**, **Policy Documents**, **User Agreement** and etc) without facing limitations of basic text components (**TextHeading**, **TextSubheading**, **TextBody** and etc)

Parameter	Deskripsi
`type` **(required)** *string*	"RichText"
`text` **(required)** *string | string array*	Dynamic "${data.text}"
`visible` *Boolean*	Dynamic "${data.is_visible}" Default: True

`RichText` component utilizes a select subset of the `Markdown` specification. It adheres strictly to standard `Markdown` syntax without introducing any custom modifications. Content created for the `RichText` component is fully compatible with standard `Markdown` documents.

> **Note:**
>
> Until V6.2, the RichText component can only be used as a standalone component on the screen and cannot be combined with other components on the same screen.
>
> Starting with V6.3, the RichText component can be used in conjunction with the Footer component on same screen, allowing the Flow to navigate from or end at the screen with RichText.

and lists.

Supported Syntax

Headings

The current syntax supports only `Heading (h1)` and `Subheading (h2)`. Other heading levels will be parsed but rendered as normal text - `TextBody`.

Flow JSON	Flow Component

<pre>{
"type": "RichText",
"text": [
"# Heading level 1",
]
}</pre>
	`TextHeading`

<pre>{
"type": "RichText",
"text": [
"## Heading level 2",
]
}</pre>
	`TextSubheading`

<pre>{
"type": "RichText",
"text": [
"### Heading level 3",
"#### Heading level 4",
"##### Heading level 5",
"###### Heading level 6",
]
}</pre>
	`TextBody`

```
{
       "type": "RichText",
       "text": [
         "Paragraph 1",
        "Paragraph 2",
       ]
    }
```

or add a blank line in your markdown document that you bind using dynamic binding syntax `${data.your_dynamic_field}`

```
# Heading 1
Paragraph 1

Paragraph 2
```

```
{
       "type": "RichText",
       "text": "${data.text}"
    }
```

Text Formatting

Flow JSON	Flow Component
`{ "type": "RichText", "text": [ "Let's make a **bold** statement ] }`	TextBody (bold)

TextBody (italic)

```
{
   "type": "RichText",
   "text": [
     "Let's make this text *italic*",
   ]
}
```

TextBody (strikethrough)

```
{
   "type": "RichText",
   "text": [
     "Let's make this text ~~Striketh
   ]
}
```

TextBody (bold-italic-strikethrough)

```
{
   "type": "RichText",
   "text": [
     "This text is ~~***really import
   ]
}
```

Lists

You can organize items into ordered and unordered lists. At the moment, only single level lists are supported.

```
{
   "type": "RichText",
   "text": [
     "1. Item 1",
     "2. Item 2",
     "3. Item 3"
   ]
}
```

`OrderedList` (not available as standalone component)

```
{
   "type": "RichText",
   "text": [
     "- Item 1",
     "- Item 2",
     "- Item 3"
   ]
}
```

`UnorderedList` (not available as standalone component)

```
{
   "type": "RichText",
   "text": [
     "+ Item 1",
     "+ Item 2",
     "+ Item 3"
   ]
}
```

Images

You can also include images in the content. Please note, external URIs are not supported and you can only include base64 inline images

```
{
   "type": "RichText",
   "text": ["![Image alt text](<base64 content>)"]
}
```

Recommended image formats:

3. webp (please note, webp is only supported starting from IOS 14.6+, that corresponds to ~98% of IOS devices)

Links

To create a link, enclose the link text in brackets and then follow it immediately with the URL in parentheses

```
{
   "type": "RichText",
   "text": [
     "[Whatsapp Flows are awesome](https://business.whatsapp.com/products/whatsa
   ]
}
```

Tables

To add a table, use three or more hyphens (---) to create each column's header, and use pipes (|) to separate each column. For compatibility, you should also add a pipe on either end of the row.

Cell content can be combined with the following syntax:

1. Italic, bold, strikethrough
2. Images
3. Links

```
{
   "type": "RichText",
   "text": [
     "| Column Header 1     | Column Header 2
     "| -------------       |  -------------
     "| **Bold** text 1     | [Link](<URI>)
     "| **Bold** text 1     | ![Image alt text](<base64 co
   ]
}
```

Width of the columns:

Width of the column is based on the Header content size. Markdown specification doesn't provide a specific syntax for controlling a column width. If you want to make a certain column wider, simply add additional content to the header:

```
  "text": [
    "| Column Header 1 - Extended width  | Column Header 2       |",
    "| -------------                     |  -------------        |",
    "| **Bold** text 1                   | Cell text 2           |",
  ]
}
```

Working with large texts

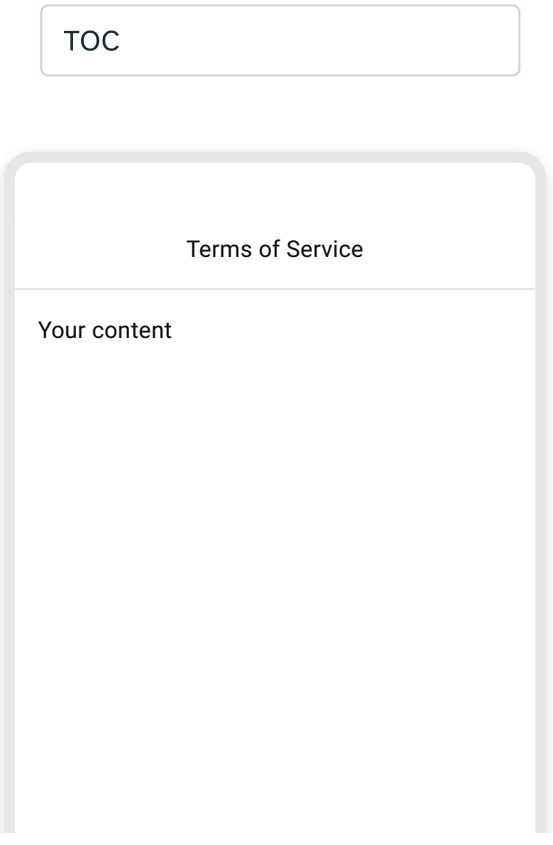
If your text content for markdown has a limited size, you can incorporate it as a static text as shown in all examples above, however if your text is large and you expect to update it often on your server, we recommend sending it as a part of dynamic data, this will improve overall readability of the JSON and allow to load always up to date text from your server.

> **Please note:** We use array text property for static cases since it's easier to read. However the components support both types: `Array of strings` and `string`. Your markdown can be sent as a normal string, you don't need to convert it to an array of strings.

Flow JSON

```
{
  "version": "7.3",
  "data_api_version": "3.0",
  "routing_model": {
    "TOC": [],
    "FIRST_SCREEN": [
      "TOC"
    ]
  },
  "screens": [
    {
      "id": "TOC",
      "data": {
        "text": {
          "type": "string",
          "__example__": "Your content"
        }
      },
      "title": "Terms of Service",
      "layout": {
        "type": "SingleColumnLayout",
        "children": [
          {
            "type": "RichText",
            "text": "${data.text}"
          }
        ]
      }
    },
    {
```

Preview Run



```
        "layout": {
          "type": "SingleColumnLayout",
          "children": [
```

Syntax cheatsheet

- Supported starting with Flow JSON version 5.1

Here is the quick overview of the syntax that's supported by RichText, TextBody and TextCaption components

Syntax	RichText	TextBody	TextCaption
`# Text Heading`	✅	❌	❌
`## Text Subheading`	✅	❌	❌
`**bold**`	✅	✅	✅
`*italic*`	✅	✅	✅
`~~strikethrough~~`	✅	✅	✅
`Normal Paragraph`	✅	✅	✅
`+ Item 1` `+ Item 2`	✅	✅	✅
`1. Item 1` `2. Item 2`	✅	✅	✅
`[Link text]` `(https://your-`	✅	✅	✅

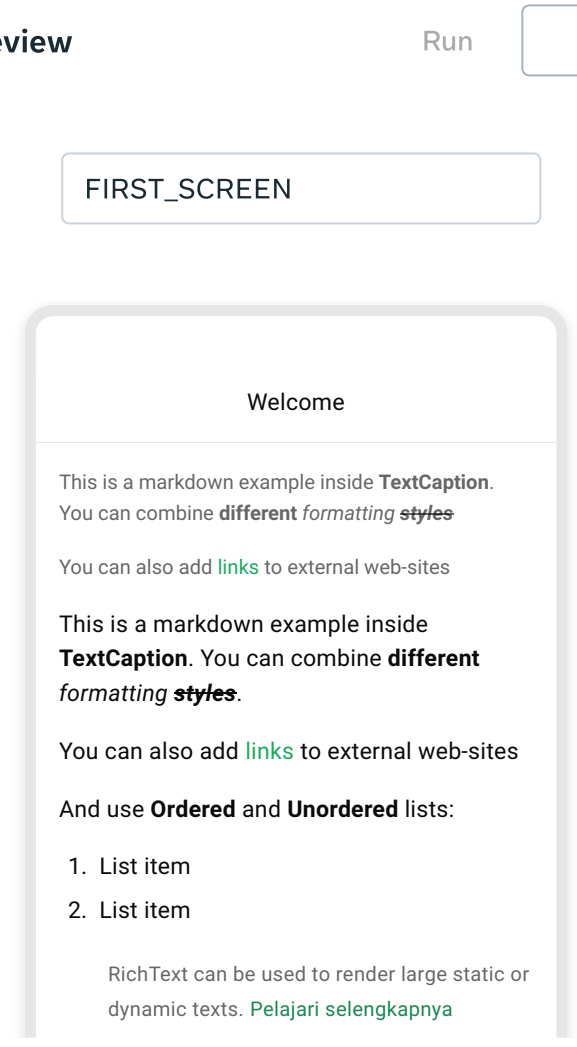
`![Image Alt]( base64-data)`	✅	❌	❌
`| Header 1 | Head | --- | --- | Row 1 | Data | Row 2 | Data | Row 3 | Data`	✅	❌	❌

Usage example

Flow JSON

```
{
  "version": "7.3",
  "screens": [
    {
      "id": "FIRST_SCREEN",
      "title": "Welcome",
      "terminal": true,
      "layout": {
        "type": "SingleColumnLayout",
        "children": [
          {
            "type": "TextCaption",
            "markdown": true,
            "text": [
              "This is a markdown examp
              "You can also add [links]
            ]
          },
          {
            "type": "TextBody",
            "markdown": true,
            "text": [
              "This is a markdown examp
              "You can also add [links]
              "And use **Ordered** and
              "1. List item",
              "2. List item"
            ]
          },
          {
            "type": "OptIn",
            "name": "toc",
```

Preview Run



```
"name": "navigate",
```

Text Entry Components

TextInput

Parameter	Deskripsi
`type` **(required)** *string*	"TextInput"
`label` **(required)** *string*	Dynamic "${data.label}"
`label-variant` *string*	"large" Label will have a more prominent style and will be displayed across multiple lines if needed. Supported starting with Flow JSON version 7.0
`input-type` *enum*	{'text','number','email', 'password', 'passcode', 'phone'}
`pattern` *String*	When specified, it is a regular expression which the input's value must match for the value to pass. • Supported starting with Flow JSON version 6.2 • Supported with input-type= {'text', 'number', 'password', 'passcode'} • Expects a raw regex string (e.g., hello, not /hello/). • When using the pattern field, helper-text is mandatory. • For input-type= {'number', 'passcode' }, a base regular expression is applied before the pattern validator, ensuring both validations are performed.
`required` *Boolean*	Dynamic "${data.is_required}"
`min-chars`	Dynamic "${data.min_chars}"

`max-chars` *String*	Dynamic "${data.max_chars}". Default value is 80 characters.
`helper-text` *String*	Dynamic "${data.helper_text}"
`name` **(required)** *String*	
`visible` *Boolean*	Dynamic "${data.is_visible}" Default: True
`init-value` *String*	Dynamic "${data.init-value}" Only available when component is outside Form component Optional Form • Supported starting with Flow JSON version 4.0
`error-message` *String*	Dynamic "${data.error-message}" Only available when component is outside Form component Optional Form • Supported starting with Flow JSON version 4.0

TextArea

Parameter	Deskripsi
`type` **(required)** *string*	"TextArea"
`label` **(required)** *string*	Dynamic "${data.label}"
`label-variant` *string*	"large"

	multiple lines if needed. Supported starting with Flow JSON version 7.0
`required` *Boolean*	Dynamic "${data.is_required}"
`max-length` *String*	Dynamic "${data.max_length}" Default value is 600 characters.
`name` **(required)** *String*	
`helper-text` *String*	Dynamic "${data.helper_text}"
`enabled` *Boolean*	Dynamic "${data.is_enabled}"
`visible` *Boolean*	Dynamic "${data.is_visible}" Default: True
`init-value` *String*	Dynamic "${data.init-value}" Only available when component is outside Form component Optional Form • Supported starting with Flow JSON version 4.0
`error-message` *String*	Dynamic "${data.error-message}" Only available when component is outside Form component Optional Form • Supported starting with Flow JSON version 4.0

Limits and Restrictions

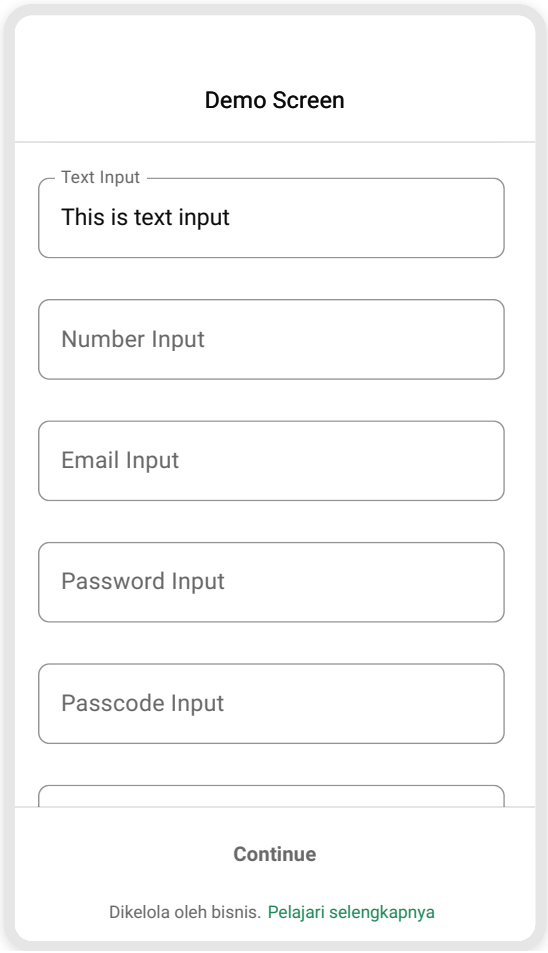
TextInput	Helper Text	80 characters
	Error Text	30 characters
	Label	20 characters
TextArea	Helper Text	80 characters
	Label	20 characters

Together, the text entry components look like as shown:

Flow JSON

```
{
  "version": "7.3",
  "screens": [
    {
      "id": "DEMO_SCREEN",
      "title": "Demo Screen",
      "terminal": true,
      "layout": {
        "type": "SingleColumnLayout",
        "children": [
          {
            "type": "Form",
            "name": "text_input_form",
            "init-values": {
              "text input": "This is te
              "text area": "This is tex
            },
            "children": [
              {
                "type": "TextInput",
                "required": true,
                "label": "Text Input",
                "name": "text input"
              },
              {
                "type": "TextInput",
                "required": true,
                "label": "Number Input"
                "input-type": "number",
                "name": "number input"
              },
              {
                "type": "TextInput",
                "required": true,
```

Preview Run



CheckboxGroup

CheckboxGroup component allows users to pick multiple selections from a list of options.

Parameter	Deskripsi
`type` **(required)** *string*	"CheckboxGroup"
`data-source` **(required)** *Array*	Dynamic "${data.data_source}" **Flow JSON versions before 5.0:** • *Array< id: String, title: String, description: String, metadata: String, enabled: Boolean>* **Flow JSON versions after 5.0:** • *Array< id: String, title: String, description: String, metadata: String, enabled: Boolean, image: Base64 of an image, alt-text: string, color: 6-digit hex color string >* **Flow JSON versions after 6.0:** • *Array< id: String, title: String, description: String, metadata: String, enabled: Boolean, image: Base64 of an image, alt-text: string, color: 6-digit hex color string, on-select-action: {name: 'update_data', payload: {...}}, on-unselect-action: {name: 'update_data', payload: {...}} >*
`name` **(required)** *String*	
`min-selected-items` *Integer*	Dynamic "${data.min_selected_items}"
`max-selected-items` *Integer*	Dynamic "${data.max_selected_items}"
`enabled` *Boolean*	Dynamic "${data.is_enabled}"
`label` *string*	Dynamic "${data.label}"

	• Flow JSON versions before 4.0: **optional** • Flow JSON versions after 4.0: **required**
`required` *Boolean*	Dynamic "${data.is_required}"
`visible` *Boolean*	Dynamic "${data.is_visible}" Default: True
`on-select-action` *Action*	`data_exchange` and `update_data` are supported. **update_data** • Supported starting with Flow JSON version 6.0
`on-unselect-action` *Action*	Only `update_data` is supported. • Supported starting with Flow JSON version 6.0 • In V6.0, if `on-unselect-action` is not added, `on-select-action` will continue to handle both selection and unselection events. However, if `on-unselect-action` is defined, it will exclusively handle unselection, while `on-select-action` will be used solely for selection.
`description` *String*	Dynamic "${data.description}" • Supported starting with Flow JSON version 4.0
`init-value` *Array<String>*	Dynamic "${data.init-value}" Only available when component is outside Form component • Supported starting with Flow JSON version 4.0
`error-message` *String*	Dynamic "${data.error-message}" Only available when component is outside Form component • Supported starting with Flow JSON version 4.0

media-size *enum*	{'regular', 'large'} Dynamic "${data.media-size}" • Supported starting with Flow JSON version 5.0

Images in WEBP format are not supported on iOS versions prior to iOS 14.

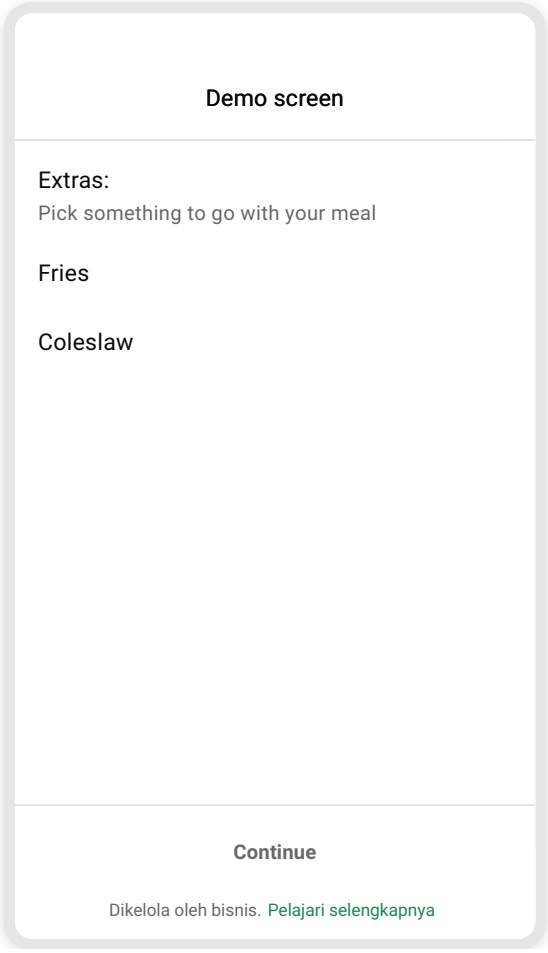
Example

Flow JSON

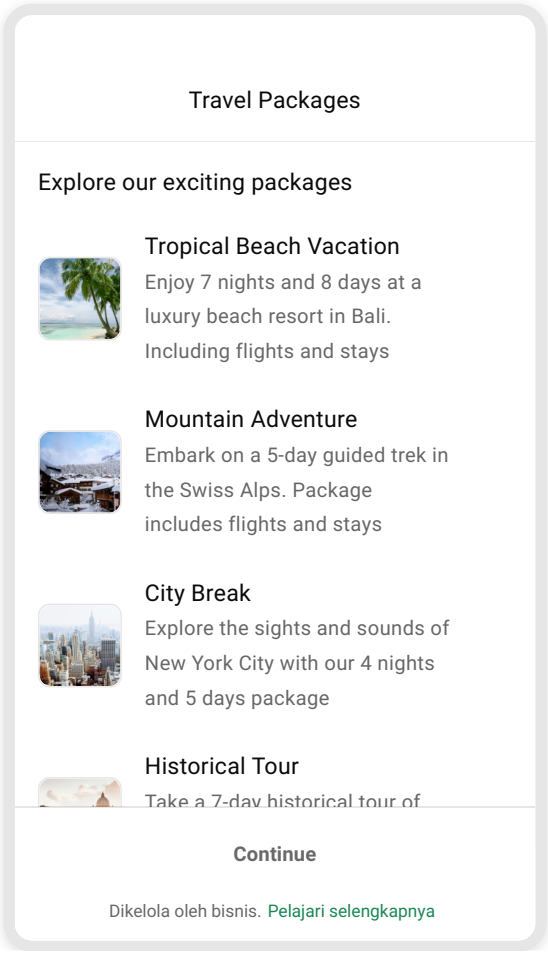
```
{
  "version": "7.3",
  "data_api_version": "3.0",
  "routing_model": {},
  "screens": [
    {
      "id": "DEMO_SCREEN",
      "terminal": true,
      "title": "Demo screen",
      "data": {
        "all_extras": {
          "type": "array",
          "items": {
            "type": "object",
            "properties": {
              "id": {
                "type": "string"
              },
              "title": {
                "type": "string"
              }
            }
          },
          "__example__": [
            {
              "id": "1",
              "title": "Fries"
            },
            {
              "id": "2",
              "title": "Coleslaw"
            }
          ]
        }
      },
      "layout": {
```

Preview

Run



```
{
  "version": "7.3",
  "screens": [
    {
      "id": "TRAVEL_PACKAGES",
      "layout": {
        "type": "SingleColumnLayout",
        "children": [
          {
            "type": "CheckboxGroup",
            "name": "packages",
            "required": true,
            "data-source": [
              {
                "id": "1",
                "title": "Tropical Beac
                "description": "Enjoy 7
                "alt-text": "beach vacc
                "image": "iVBORw0KGgoAA
              },
              {
                "id": "2",
                "title": "Mountain Adve
                "description": "Embark
                "image": "iVBORw0KGgoAA
              },
              {
                "id": "3",
                "title": "City Break",
                "description": "Explore
                "image": "iVBORw0KGgoAA
              },
              {
                "id": "4",
                "title": "Historical To
                "description": "Take a
```



For the `data-source` field, you can declare it dynamically or statically.

Static Example

This static example hardcodes the respective `id`'s and `title`'s for the `data-source` field.

Flow JSON **Preview** Run

```
{
  "version": "7.3",
  "screens": [
    {
      "id": "DEMO_SCREEN",
      "title": "Demo Screen",
```

```
        "children": [
          {
            "type": "Form",
            "name": "checkbox_example_f
            "children": [
              {
                "type": "CheckboxGroup'
                "name": "checkboxgroup'
                "label": "Checkbox grou
                "data-source": [
                  {
                    "id": "Checkbox 1",
                    "title": "Checkbox
                  },
                  {
                    "id": "Checkbox 2",
                    "title": "Checkbox
                  },
                  {
                    "id": "Checkbox 3",
                    "title": "Checkbox
                  }
                ]
              },
              {
                "type": "Footer",
                "label": "Continue",
```

Checkbox group example (opsional)

Checkbox 1

Checkbox 2

Checkbox 3

Dynamic Example

In this dynamic example, you can see that `data-source` references the `days_per_week_options` of type `array` defined before it using `days_per_week_options`. When defining such a structure, you need to specify `items` in the `array`, which will be of type `object`. Then inside the `items` object, you have a `properties` dictionary with `id` and `title` just like in the static declaration. Both `id` and `title` will always be of type `String`. Within the `days_per_week_options` array, you must define concrete examples in the `__example__` field.

Flow JSON

Preview Run

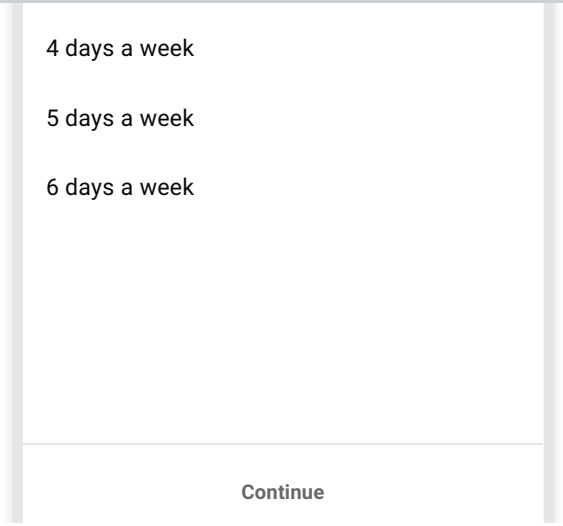
```
{
  "version": "7.3",
  "screens": [
    {
      "id": "DEMO_SCREEN",
      "title": "Demo Screen",
      "terminal": true,
      "data": {
        "days_per_week_heading": {
          "type": "string",
          "__example__": "How often wou
        },
        "days_per_week_options": {
          "type": "array",
          "items": {
```

Demo Screen

How often would you like to workout?

2 days a week


```
        "type": "string"
      },
      "title": {
        "type": "string"
      }
    }
  },
  "__example__": [
    {
      "id": "2",
      "title": "2 days a week"
    },
    {
      "id": "3",
      "title": "3 days a week"
    },
    {
      "id": "4",
```

Limits and Restrictions

Type	Limit / Restriction
Label Content	30 Characters
Title	30 Characters
Description	300 Characters
Metadata	20 Characters
Min # of options	1
Max # of options	20
Image	**Flow JSON versions before 6.0:** 300KB **Flow JSON versions after 6.0:** 100KB

RadioButtonsGroup

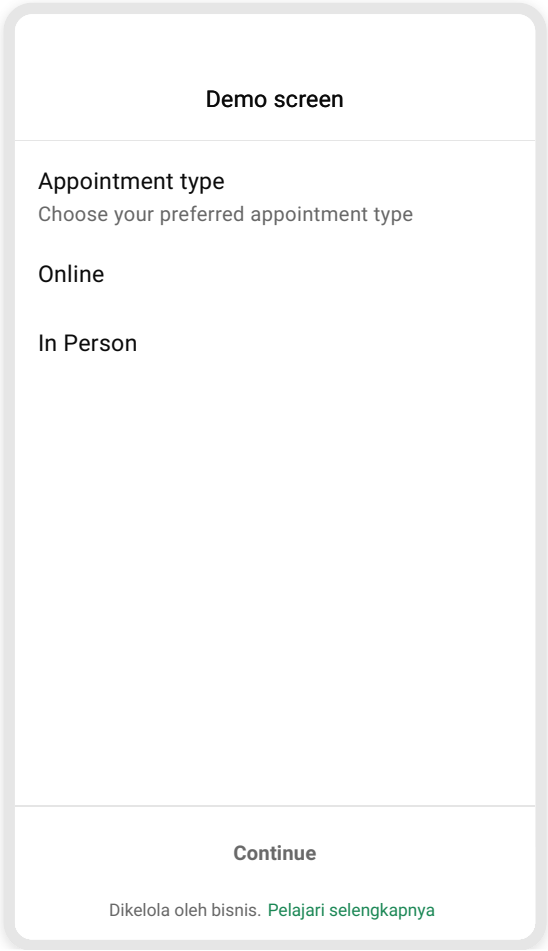
`type` **(required)** *string*	"RadioButtonsGroup"
`data-source` **(required)** *Array*	Dynamic "${data.data_source}" **Flow JSON versions before 5.0:** • *Array< id: String, title: String, description: String, metadata: String, enabled: Boolean>* **Flow JSON versions after 5.0:** • *Array< id: String, title: String, description: String, metadata: String, enabled: Boolean, image: Base64 of an image, alt-text: string, color: 6-digit hex color string >* **Flow JSON versions after 6.0:** • *Array< id: String, title: String, description: String, metadata: String, enabled: Boolean, image: Base64 of an image, alt-text: string, color: 6-digit hex color string, on-select-action: {name: 'update_data', payload: {...}}, on-unselect-action: {name: 'update_data', payload: {...}} >*
`name` **(required)** *String*	
`enabled` *Boolean*	Dynamic "${data.is_enabled}"
`label` *string*	Dynamic "${data.label}" • Flow JSON versions before 4.0: **optional** • Flow JSON versions after 4.0: **required**
`required` *Boolean*	Dynamic "${data.is_required}"
`visible` *Boolean*	Dynamic "${data.is_visible}" Default: True
`on-select-action` *Action*	`data_exchange` and `update_data` are supported. **update_data**

`on-unselect-action` *Action*	Only `update_data` is supported. • Supported starting with Flow JSON version 6.0 • In V6.0, if `on-unselect-action` is not added, `on-select-action` will continue to handle both selection and unselection events. However, if `on-unselect-action` is defined, it will exclusively handle unselection, while `on-select-action` will be used solely for selection.
`description` *String*	Dynamic "${data.description}" • Supported starting with Flow JSON version 4.0
`init-value` *Array<String>*	Dynamic "${data.init-value}" Only available when component is outside Form component • Supported starting with Flow JSON version 4.0
`error-message` *String*	Dynamic "${data.error-message}" Only available when component is outside Form component • Supported starting with Flow JSON version 4.0
`media-size` *enum*	{'regular', 'large'} Dynamic "${data.media-size}" • Supported starting with Flow JSON version 5.0

Images in WEBP format are not supported on iOS versions prior to iOS 14.

Example

```
  "version": "7.3",
  "data_api_version": "3.0",
  "routing_model": {},
  "screens": [
    {
      "id": "DEMO_SCREEN",
      "terminal": true,
      "title": "Demo screen",
      "data": {
        "all_appointment_types": {
          "type": "array",
          "items": {
            "type": "object",
            "properties": {
              "id": {
                "type": "string"
              },
              "title": {
                "type": "string"
              }
            }
          },
          "__example__": [
            {
              "id": "1",
              "title": "Online"
            },
            {
              "id": "2",
              "title": "In Person"
            }
          ]
        }
      },
      "layout": {
```

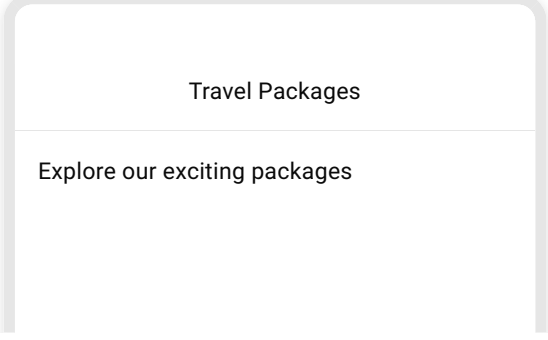


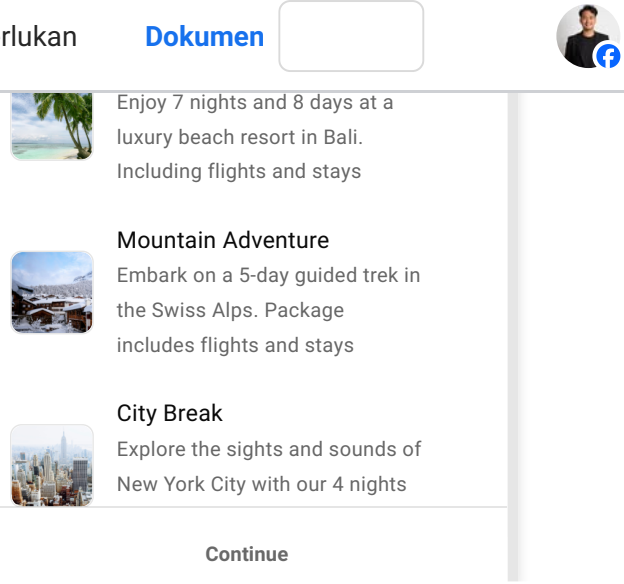
Flow JSON

Preview

Run

```
{
  "version": "7.3",
  "screens": [
    {
      "id": "TRAVEL_PACKAGES",
      "layout": {
        "type": "SingleColumnLayout",
        "children": [
          {
            "type": "RadioButtonsGroup"
            "name": "packages",
            "required": true,
            "data-source": [
              {
                "id": "1",
                "title": "Tropical Bea
```





For the `data-source` field, you can declare it dynamically or statically.

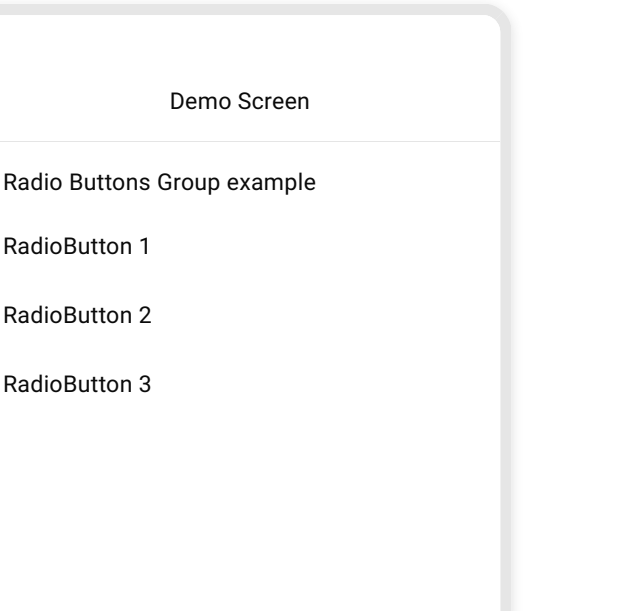
Static Example

This static example hardcodes the respective `id`'s and `title`'s for the `data-source` field.

Flow JSON

```
{
  "version": "7.3",
  "data_api_version": "3.0",
  "routing_model": {},
  "screens": [
    {
      "id": "DEMO_SCREEN",
      "title": "Demo Screen",
      "terminal": true,
      "layout": {
        "type": "SingleColumnLayout",
        "children": [
          {
            "type": "Form",
            "name": "text_input_form",
            "children": [
              {
                "type": "RadioButtonsGr
                "name": "radiobuttonsgr
                "label": "Radio Buttons
                "required": true,
                "data-source": [
                  {
                    "id": "RadioButton
                    "title": "RadioButt
                  },
```

Preview Run



```
      },
      {
        "id": "RadioButton
        "title": "RadioButt
      }
    ]
  },
```

Dynamic Example

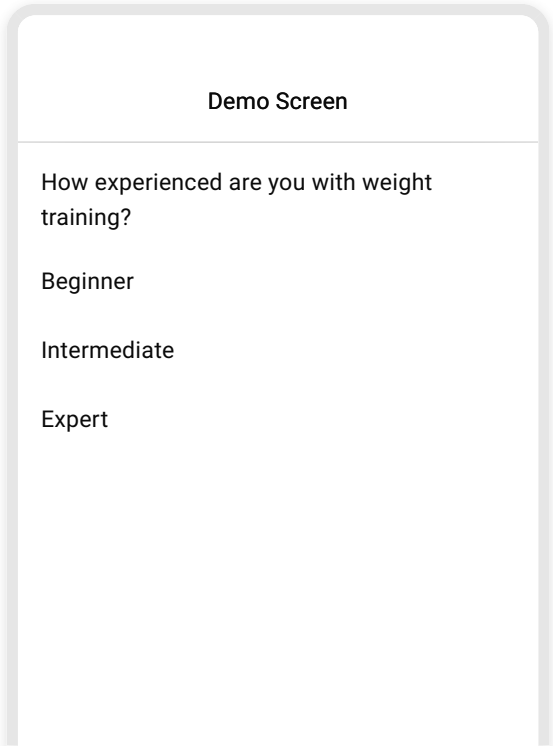
In this dynamic example, you can see that `data-source` references the `experience_level_options` of type `array` defined before it using `data.experience_level_options`. When defining such a structure, you need to specify `items` in the `array`, which will be of type `object`. Then inside the `items` object, you have a `properties` dictionary with `id` and `title` just like in the static declaration. Both `id` and `title` will always be of type `String`. Within in the `experience_level_options` array you must define concrete examples in the `__example__` field.

Flow JSON

```
{
  "version": "7.3",
  "screens": [
    {
      "id": "DEMO_SCREEN",
      "title": "Demo Screen",
      "terminal": true,
      "data": {
        "experience_level_heading": {
          "type": "string",
          "__example__": "How experienc
        },
        "experience_level_options": {
          "type": "array",
          "items": {
            "type": "object",
            "properties": {
              "id": {
                "type": "string"
              },
              "title": {
                "type": "string"
              }
            }
          },
          "__example__": [
            {
              "id": "beginner",
              "title": "Beginner"
            },
            {
```

Preview

Run



```
{
  "id": "expert",
```

Limits and Restrictions

Type	Limit / Restriction
Label Content	30 Characters
Title	30 Characters
Description	300 Characters
Metadata	20 Characters
Min # of options	1
Max # of options	20
Image	**Flow JSON versions before 6.0:** 300KB **Flow JSON versions after 6.0:** 100KB

Footer

Parameter	Deskripsi
`type` **(required)** *string*	"Footer"
`label` **(required)** *string*	Dynamic "${data.label}"
`left-caption` *String*	Dynamic "${data.left_caption}" Can set left-caption **and** right-caption **or** only center-

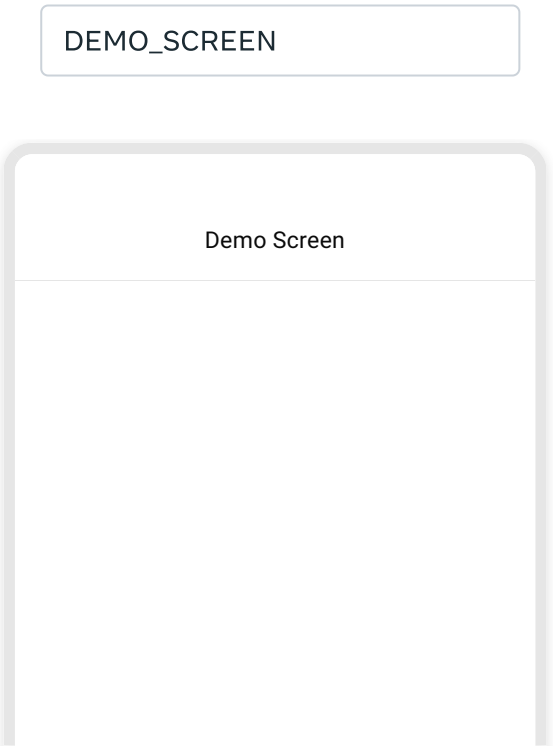
`center-caption` *String*	Dynamic "${data.center_caption}" Can set center-caption **or** left-caption **and** right-caption, but not all 3 at once
`right-caption` *String*	Dynamic "${data.right_caption}" Can set right-caption **and** left-caption **or** only center-caption, but not all 3 at once
`enabled` *Boolean*	Dynamic "${data.is_enabled}"
`on-click-action` **(required)** *Action*	Action

Flow JSON

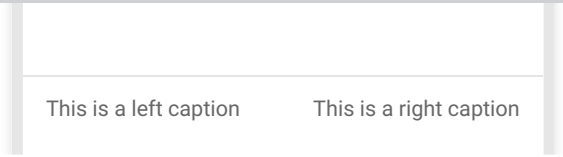
```
{
  "version": "7.3",
  "screens": [
    {
      "id": "DEMO_SCREEN",
      "title": "Demo Screen",
      "layout": {
        "type": "SingleColumnLayout",
        "children": [
          {
            "type": "Form",
            "name": "text_input_form",
            "children": [
              {
                "type": "Footer",
                "label": "This is a foo
                "left-caption": "This i
                "right-caption": "This
                "on-click-action": {
                  "name": "navigate",
                  "next": {
                    "type": "screen",
                    "name": "NEXT_SCREE
                  },
                  "payload": {}
                }
              }
```

Preview

Run



```
        }
      },
      {
        "id": "NEXT_SCREEN",
        "title": "Next Screen",
        "terminal": true,
```



Limits and Restrictions

Type	Limit / Restriction
Label Max Character Limit	35
Captions Max Character Limit	15

OptIn

Parameter	Deskripsi
`type` **(required)** *string*	"OptIn"
`label` **(required)** *string*	Dynamic "${data.label}"
`required` *Boolean*	Dynamic "${data.is_required}"
`name` **(required)** *String*	
`on-click-action` *Action*	Action that is executed on clicking "Read more". "Read more" is only visible when an on-click-action is specified. Allowed values are `data_exchange` and `navigate`. From Flow JSON version 6.0 and later, allowed values are `data_exchange`, `navigate` and `open_url`.

`on-select-action` *Action*	Only `update_data` is supported. • Supported starting with Flow JSON version 6.0
`on-unselect-action` *Action*	Only `update_data` is supported. • Supported starting with Flow JSON version 6.0
`visible` *Boolean*	Dynamic "${data.is_visible}" Default: True
`init-value` *Boolean*	Dynamic "${data.init-value}" Only available when component is outside Form component Optional Form • Supported starting with Flow JSON version 4.0

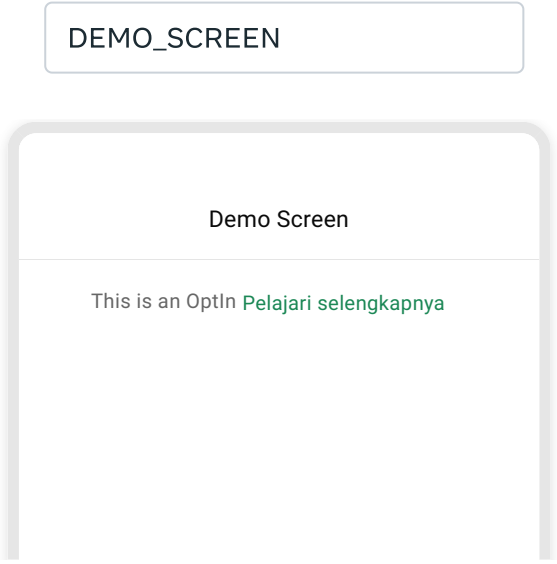
Example

Flow JSON

```
{
  "version": "7.3",
  "screens": [
    {
      "id": "DEMO_SCREEN",
      "title": "Demo Screen",
      "layout": {
        "type": "SingleColumnLayout",
        "children": [
          {
            "type": "Form",
            "name": "text_input_form",
            "children": [
              {
                "type": "OptIn",
                "name": "OptIn",
                "label": "This is an Op
                "required": true,
                "on-click-action": {
                  "name": "navigate",
```

Preview

Run



```
                "payload": {}
              }
            }
          ]
        }
      ]
    }
  },
  {
    "id": "NEXT_SCREEN",
    "title": "Next Screen",
    "terminal": true,
```

Limits and Restrictions

Type	Limit / Restriction
Content Max Character Limit	120
Max number of Opt-Ins Per Screen	5

Dropdown

Parameter	Deskripsi
`type` **(required)** *string*	"Dropdown"
`label` **(required)** *string*	
`data-source` **(required)** *Array*	Dynamic "${data.data_source}" **Flow JSON versions before 5.0:** • *Array< id: String, title: String, description: String, metadata: String, enabled: Boolean>* **Flow JSON versions after 5.0:** • *Array< id: String, title: String, description: String, metadata: String, enabled: Boolean, image: Base64 of*

	>` **Flow JSON versions after 6.0:** • *Array< id: String, title: String, description: String, metadata: String, enabled: Boolean, image: Base64 of an image, alt-text: string, color: 6-digit hex color string, on-select-action: {name: 'update_data', payload: {...}}, on-unselect-action: {name: 'update_data', payload: {...}} >*
`required` *Boolean*	
`enabled` *Boolean*	Dynamic "${data.is_enabled}"
`required` *Boolean*	Dynamic "${data.is_required}"
`visible` *Boolean*	Dynamic "${data.is_visible}" Default: True
`on-select-action` *Action*	`data_exchange` and `update_data` are supported. **update_data** • Supported starting with Flow JSON version 6.0
`on-unselect-action` *Action*	Only `update_data` is supported. • Supported starting with Flow JSON version 6.0 • In V6.0, if `on-unselect-action` is not added, `on-select-action` will continue to handle both selection and unselection events. However, if `on-unselect-action` is defined, it will exclusively handle unselection, while `on-select-action` will be used solely for selection.
`init-value` *String*	Dynamic "${data.init-value}" Only available when component is outside Form component
`error-message` *String*	Dynamic "${data.error-message}" Only available when component is outside Form component

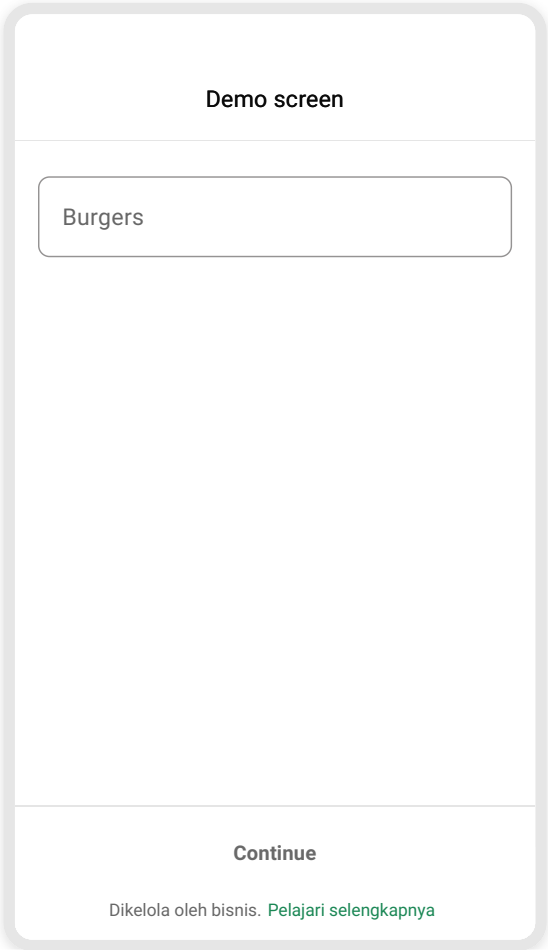
Example

Flow JSON

```
{
  "version": "7.3",
  "data_api_version": "3.0",
  "routing_model": {
    "DEMO_SCREEN": []
  },
  "screens": [
    {
      "id": "DEMO_SCREEN",
      "terminal": true,
      "title": "Demo screen",
      "data": {
        "all_burgers": {
          "type": "array",
          "items": {
            "type": "object",
            "properties": {
              "id": {
                "type": "string"
              },
              "title": {
                "type": "string"
              },
              "description": {
                "type": "string"
              },
              "metadata": {
                "type": "string"
              }
            }
          },
          "__example__": [
            {
              "id": "1",
              "title": "Beef burger",
              "description": "Beef, red
```

Preview

Run



Flow JSON

```
{
  "version": "7.3",
  "screens": [
    {
      "id": "TRAVEL_PACKAGES",
      "layout": {
        "type": "SingleColumnLayout",
```

Preview

Run


```
"name": "packages",
"required": true,
"data-source": [
  {
    "id": "1",
    "title": "Life Insuranc
    "description": "Get com
    "metadata": "Annual Fee
    "image": "iVBORw0KGgoAA
  },
  {
    "id": "2",
    "title": "Health Insura
    "description": "Explore
    "metadata": "Annual Fee
    "image": "iVBORw0KGgoAA
  },
  {
    "id": "3",
    "title": "Home Insuranc
    "description": "Protect
    "metadata": "Annual Fee
    "image": "iVBORw0KGgoAA
  }
],
"label": "Insurance options
```


Limits and Restrictions

Type	Limit / Restriction
Label	20 characters
Title	30 characters
Min dropdown options	1
Max dropdown options	200 if no images are present in the `data-source`, 100 otherwise
Description	300 characters
Metadata	20 characters
Image	**Flow JSON versions before 6.0:** 300KB **Flow JSON versions after 6.0:**

For the `data-source` field, you can declare it dynamically or statically.

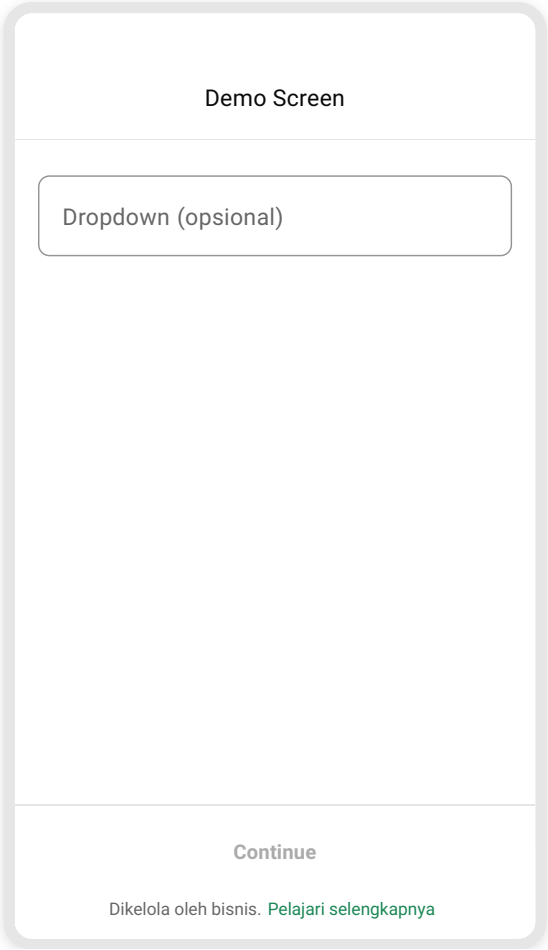
Static Example

This static example hardcodes the respective `id`'s and `title`'s for the `data-source` field.

Flow JSON

```
{
  "version": "7.3",
  "screens": [
    {
      "id": "DEMO_SCREEN",
      "title": "Demo Screen",
      "terminal": true,
      "layout": {
        "type": "SingleColumnLayout",
        "children": [
          {
            "type": "Form",
            "name": "text_input_form",
            "children": [
              {
                "type": "Dropdown",
                "label": "Dropdown",
                "name": "Dropdown",
                "data-source": [
                  {
                    "id": "Dropdown Opt
                    "title": "Dropdown
                  },
                  {
                    "id": "Dropdown Opt
                    "title": "Dropdown
                  },
                  {
                    "id": "Dropdown Opt
                    "title": "Dropdown
                  }
                ]
              },
              {
                "type": "Footer",
                "label": "Continue",
```

Preview Run



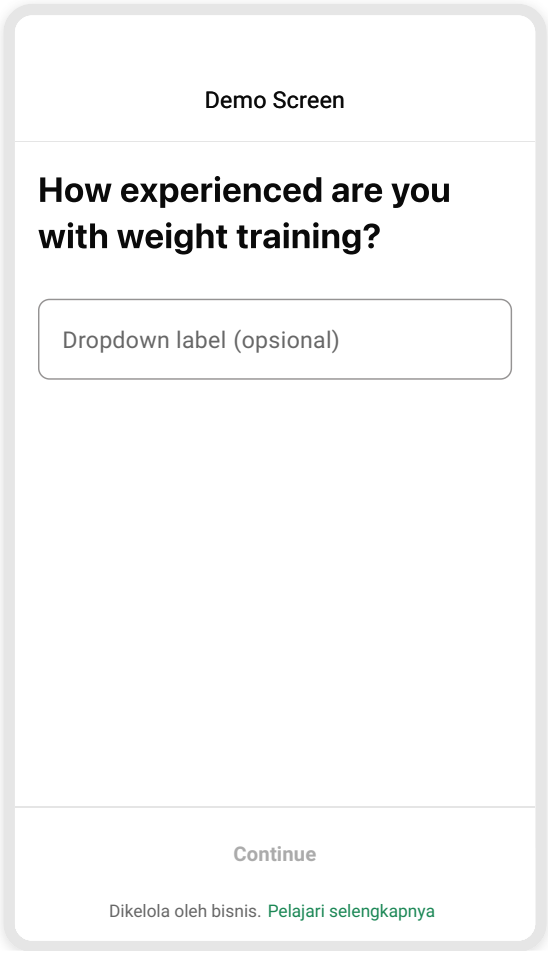
Dynamic Example

you need to specify `items` in the `array`, which will be of type `object`. Then inside the `items` object, you have a `properties` dictionary with `id` and `title` just like in the static declaration. Both `id` and `title` will always be of type `String`. Within the `experience_level_options` array you must define concrete examples in the `__example__` field.

Flow JSON

```
{
  "version": "7.3",
  "screens": [
    {
      "id": "DEMO_SCREEN",
      "title": "Demo Screen",
      "terminal": true,
      "data": {
        "experience_level_heading": {
          "type": "string",
          "__example__": "How experienc
        },
        "experience_level_options": {
          "type": "array",
          "items": {
            "type": "object",
            "properties": {
              "id": {
                "type": "string"
              },
              "title": {
                "type": "string"
              }
            }
          },
          "__example__": [
            {
              "id": "1",
              "title": "Beginner"
            },
            {
              "id": "2",
              "title": "Intermediate"
            },
            {
              "id": "3",
```

Preview Run



Embedded Link

Parameter	Deskripsi
`type`	"EmbeddedLink"

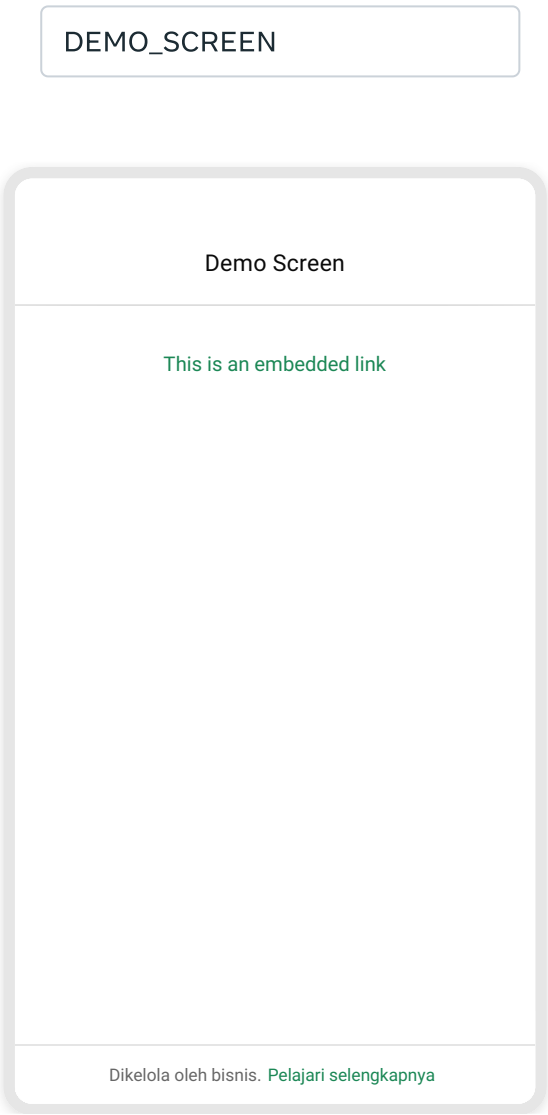
`text` **(required)** *string*	Dynamic "${data.text}"
`on-click-action` **(required)** *Action*	Action Allowed values are `data_exchange` and `navigate`. From Flow JSON version 6.0 and later, allowed values are `data_exchange`, `navigate` and `open_url`.
`visible` *Boolean*	Dynamic "${data.is_visible}" Default: True

Flow JSON

```
{
  "version": "7.3",
  "screens": [
    {
      "id": "DEMO_SCREEN",
      "title": "Demo Screen",
      "layout": {
        "type": "SingleColumnLayout",
        "children": [
          {
            "type": "EmbeddedLink",
            "text": "This is an embedde
            "on-click-action": {
              "name": "navigate",
              "next": {
                "type": "screen",
                "name": "FINISH"
              },
              "payload": {
                "test_payload": "This i
              }
            }
          }
        ]
      }
    },
    {
      "id": "FINISH",
      "data": {
        "test_payload": {
          "type": "string",
          "__example__": "CTA title"
        }
      },
      "title": "Final screen",
```

Preview

Run



Limits and Restrictions

Type	Limit / Restriction
Character limit	25
Case	No restriction on formatting
Max Number of Embedded Links Per Screen	2
Text	Empty or Blank value is not accepted

DatePicker

The DatePicker component allows users to input dates through an intuitive date selection interface.

> Before Flow JSON version 5.0, the DatePicker doesn't support scenarios where the business and the end user are in different time zones. We recommend only using the component if you plan to send your Flows to users in a specific timezone. For details, please refer to section Guidelines for Usage
>
> Starting from Flow JSON version 5.0, the DatePicker has been updated to use a formatted date string in the format "YYYY-MM-DD", such as "2024-10-21", for setting and retrieving date values. This update makes the date values of the date picker unrelated to time zones, allowing businesses to send messages and collect dates from users in any time zone.

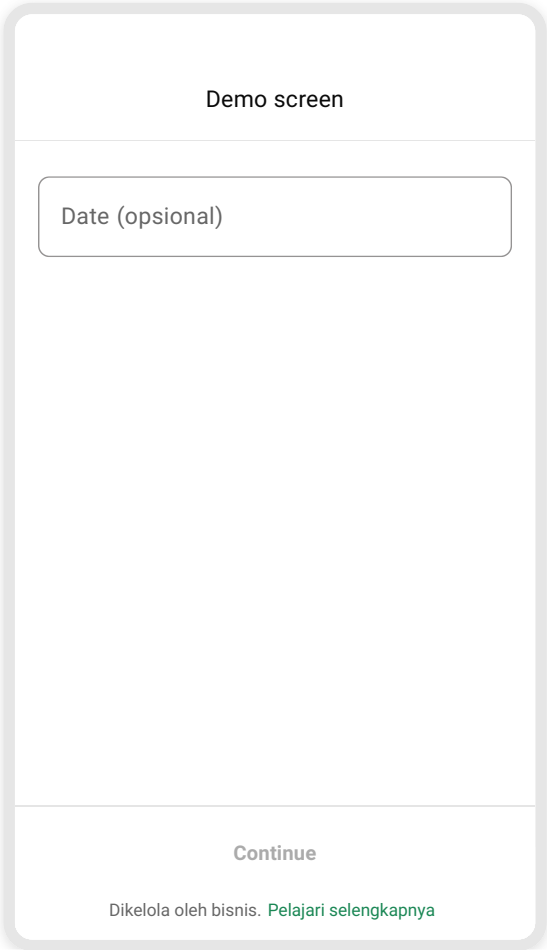
Parameter	Deskripsi
`type` **(required)** *string*	"DatePicker"
`label` **(required)** *string*	Dynamic "${data.label}"

`min-date` *String (timestamp in milliseconds)*	Dynamic "${data.min_date}". Please refer to section Guidelines for Usage
`max-date` *String (timestamp in milliseconds)*	Dynamic "${data.max_date}". Please refer to section Guidelines for Usage
`name` **(required)** *string*	
`unavailable-dates` *Array < timestamp in milliseconds: String >*	Dynamic "${data.unavailable_dates}". Please refer to section Guidelines for Usage
`visible` *Boolean*	Dynamic "${data.is_visible}" Default: True
`helper-text` *String*	Dynamic "${data.helper_text}"
`enabled` *Boolean*	Dynamic "${data.is_enabled}" Default: True
`on-select-action` *Action*	Only `data_exchange` is supported.
`init-value` *String*	Dynamic "${data.init-value}" Only available when component is outside Form component Optional Form • Supported starting with Flow JSON version 4.0
`error-message` *String*	Dynamic "${data.error-message}" Only available when component is outside Form component Optional Form • Supported starting with Flow JSON version 4.0

Flow JSON

```
{
  "version": "7.3",
  "data_api_version": "3.0",
  "routing_model": {},
  "screens": [
    {
      "id": "DEMO_SCREEN",
      "terminal": true,
      "title": "Demo screen",
      "layout": {
        "type": "SingleColumnLayout",
        "children": [
          {
            "type": "DatePicker",
            "name": "date",
            "label": "Date",
            "min-date": "2024-10-21",
            "max-date": "2024-11-12",
            "unavailable-dates": [
              "2024-10-28",
              "2024-11-01"
            ],
            "on-select-action": {
              "name": "data_exchange",
              "payload": {
                "date": "${form.date}"
              }
            }
          },
          {
            "type": "Footer",
            "label": "Continue",
            "on-click-action": {
              "name": "data_exchange",
              "payload": {}
            }
```

Preview Run



Guidelines for Usage

Before flow JSON version 5.0

Due to current system limitations, the DatePicker functions correctly and as intended(that is, correct selection range is shown to the User, and accurate user-selection value is returned to the Business) as long as

> Correct behavior is not guaranteed if businesses and end-users are in different time zones. For example, if a business operating in Sao Paulo (UTC-3) sends a Flow to a user in Manaus (UTC-4), the DatePicker may not work as expected. We don't recommend using it if your users are in different time zones than you.

Handling of Dates for Businesses and Users in the Same Time Zone

DatePicker allows setting of date range for user selection through `min-dates` and `max-dates` fields, and also prevents selection of specific dates using the `unavailable-dates` field. If you have not supplied the date range , then by default, the component allows the user to select dates from `1 January 1900` to `31 December 2100`.

Setting Date Parameters in the Component

When you specify the date range or set unavailable dates, you should convert your local dates with midnight (00:00:00) as a base time to UTC timestamps.

For example, if you are a business based in India who wants to collect a date in the range `21 March 2024` to `25 March 2024`, then you should set `min-dates` and `max-dates` as `1710958020000` and `1711303620000`, respectively.

`21 March 2024, 00:00:00.000 IST` converts to `20 March 2024, 18:30:00.000 UTC` which is represented by timestamp `1710958020000`.

`25 March 2024, 00:00:00.000 IST` converts to `24 March 2024, 18:30:00.000 UTC` which is represented by timestamp `1711303620000`.

Component Integration

DatePicker will read the timestamps in `min-dates`, `max-dates` and `unavailable-dates` fields and convert it to the end user's local date for displaying on the UI. In the example we discussed above, a user in India will see dates from `21 March 2024` to `25 March 2024` in the DatePicker component.

Processing User Selection

Businesses will receive a UTC timestamp, which should be converted back to the business's local time zone. Importantly, businesses should focus solely on the date portion of the resulting timestamp , disregarding the time portion. This ensures that the date remains consistent with the user's selection. Unfortunately, this conversion will only work correctly when the business and user are in the same time zone.

should treat `21st March 2024` as the user selected date.

Recommendation for navigating Time Zone differences

If you need to send flow messages to users in time zones different from yours despite reviewing the above guidelines, follow these steps to overcome the limitation:

- If you are a business based in Brazil and want to serve flows to your users across the country, then your time zone range will be `UTC-2 (Fernando de Noronha)` to `UTC-5 (Rio Branco)`.
- Add a `Dropdown` component within your Flow that allows users to select their current time zone.
- Identify the westernmost time zone from your time zone range. In our example, it is `UTC-5`.
- Provide the dates you want to collect in the westernmost time zone, using midnight as the reference time. For example, if you want to collect dates from `March 20th, 2024` to `March 25th, 2024`, then provide the timestamp in milliseconds for `March 20th, 2024 at 5 AM UTC` and `March 25th, 2024 at 5 AM UTC`.
- Convert the timestamps received from the user to their respective time zone and use the corresponding date. For example, if a user is in Sao Paulo(UTC-3) and you receive a timestamp of `1710910800000`, then convert it to `UTC-3` to get `March 20th, 2024`.

Start from flow JSON version 5.0

DatePicker component has been updated to use a formatted date string in the format "YYYY-MM-DD", such as "2024-10-21", for setting and retrieving date values. This update makes the date values of the date picker unrelated to time zones, allowing businesses to send messages and collect dates from users in any time zone in a consistent manner.

Limits and Restrictions

Type	Limit / Restriction
Label Max Length	40 characters
Helper Text Max Length	80 characters
Error Message Max Length	80 characters

CalendarPicker

The CalendarPicker component allows users to select a single date or a range of dates from a full calendar interface.

Parameter	Deskripsi
`type` **(required)** *String*	"CalendarPicker"
`name` **(required)** *String*	
`title` *String*	Dynamic "${data.title}" Only available when 'mode' is set to 'range'
`description` *String*	Dynamic "${data.description}" Only available when 'mode' is set to 'range'
`label` **(required)** *String*	Dynamic "${data.label}" When 'mode' is set to 'range' the value should be in '{"start-date": String, "end-date": String}' format
`helper-text` *String*	Dynamic "${data.helper_text}" When 'mode' is set to 'range' the value should be in '{"start-date": String, "end-date": String}' format
`required` *Boolean*	Dynamic "${data.is_required}" Default: False When 'mode' is set to 'range' the value should be in '{"start-date": Boolean, "end-date": Boolean}' format
`visible` *Boolean*	Dynamic "${data.is_visible}" Default: True
`enabled` *Boolean*	Dynamic "${data.is_enabled}" Default: True
`mode` *enum*	{"single", "range"} Dynamic "${data.mode}" Default: "single" Allows to select one date in 'single' mode or start and end dates in 'range' mode

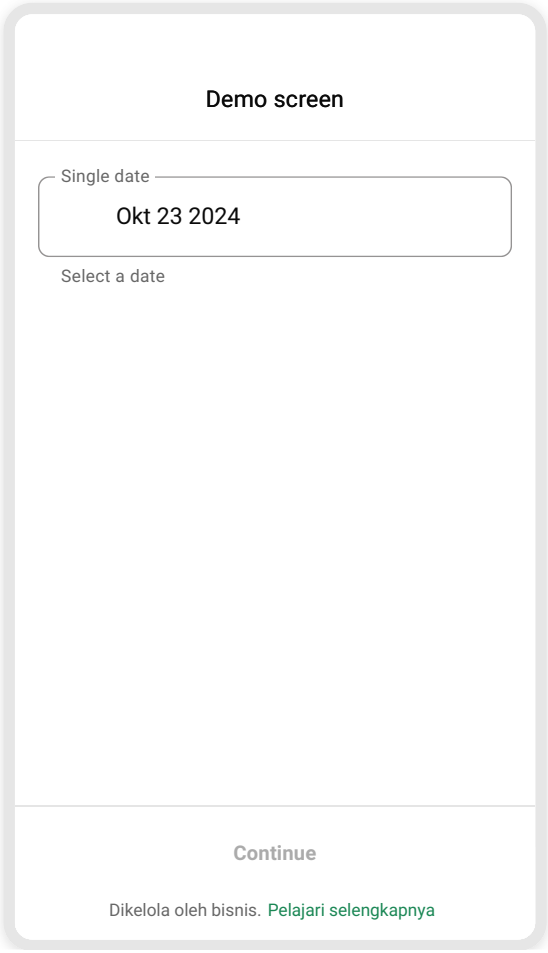
`min-date` *String*	Dynamic "${data.min_date}" Formatted date string in the format "YYYY-MM-DD" Disallows selecting dates before specified min-date
`max-date` *String*	Dynamic "${data.max_date}" Formatted date string in the format "YYYY-MM-DD" Disallows selecting dates after specified max-date
`unavailable-dates` *Array<String>*	Dynamic "${data.unavailable_dates}" Formatted date strings in the format "YYYY-MM-DD" Disallows selecting specific dates, should be in the range between min-date and max-date if specified
`include-days` *Array<enum>*	{"Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"} Dynamic "${data.include_days}" Default: all weekdays - ["Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"] Enables specific weekdays, for example to enable only working days Monday through Friday and disallow selecting Saturdays and Sundays
`min-days` *Integer*	Dynamic "${data.min_days}" Available only in 'range' mode to set the minimum number of days between start and end dates
`max-days` *Integer*	Dynamic "${data.max_days}" Available only in 'range' mode to set the maximum number of days between start and end dates
`on-select-action` *Action*	Only 'data_exchange' is supported. Payload that is sent to a data channel business endpoint is a string in "YYYY-MM-DD" format for 'single' mode or dictionary in {"start-date":"YYYY-MM-DD","end-date":"YYYY-MM-DD"} format for 'range' mode
`init-value` *String*	Dynamic "${data.init-value}" When 'mode' is set to 'range' the value should be in '{"start-date": String, "end-date": String}' format Only available when component is outside Form component
`error-message` *String*	Dynamic "${data.error-message}" When 'mode' is set to 'range' the value should be in '{"start-date": String, "end-date": String}' format Only available when component is outside Form component

CalendarPicker single mode example

Flow JSON

```
{
  "version": "7.3",
  "data_api_version": "3.0",
  "routing_model": {},
  "screens": [
    {
      "id": "DEMO_SCREEN",
      "terminal": true,
      "title": "Demo screen",
      "layout": {
        "type": "SingleColumnLayout",
        "children": [
          {
            "type": "CalendarPicker",
            "name": "calendar",
            "label": "Single date",
            "helper-text": "Select a da
            "required": true,
            "mode": "single",
            "min-date": "2024-10-21",
            "max-date": "2025-12-12",
            "unavailable-dates": [
              "2024-11-28",
              "2024-11-01"
            ],
            "include-days": [
              "Mon",
              "Tue",
              "Wed",
              "Thu",
              "Fri"
            ],
            "init-value": "2024-10-23",
            "on-select-action": {
              "name": "data_exchange",
              "payload": {
```

Preview Run

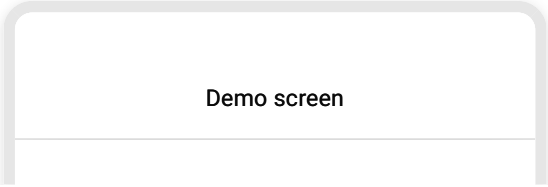


CalendarPicker range mode example

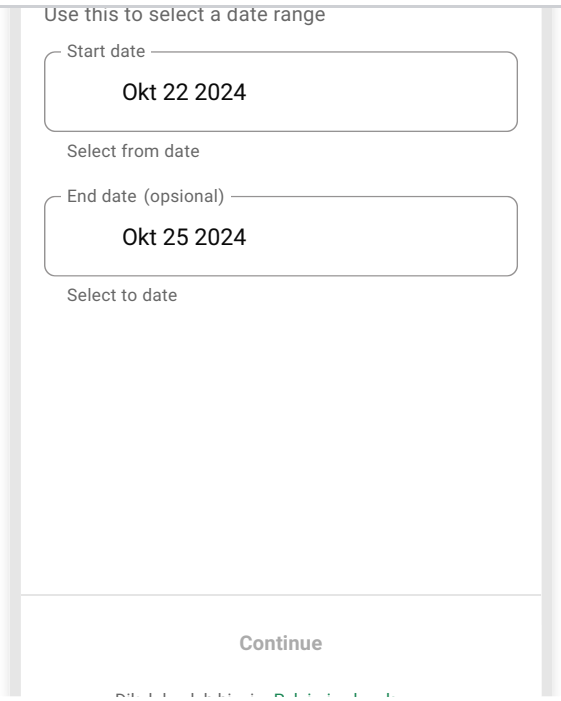
Flow JSON

```
{
  "version": "7.3",
  "data_api_version": "3.0",
  "routing_model": {},
  "screens": [
    {
      "id": "DEMO_SCREEN",
      "terminal": true,
      "title": "Demo screen",
      "layout": {
```

Preview Run



```
"type": "CalendarPicker",
"name": "calendar_range",
"title": "Range calendar",
"description": "Use this to
"label": {
  "start-date": "Start date
  "end-date": "End date"
},
"helper-text": {
  "start-date": "Select fro
  "end-date": "Select to da
},
"required": {
  "start-date": true,
  "end-date": false
},
"mode": "range",
"min-date": "2024-10-21",
"max-date": "2025-12-12",
"unavailable-dates": [
  "2024-11-28",
  "2024-11-01"
],
```



Limits and Restrictions

Type	Limit / Restriction
Title Max Length	80 characters
Description Max Length	300 characters
Label Max Length	40 characters
Helper Text Max Length	80 characters
Error Message Max Length	80 characters

Image

`type` **(required)** *string*	"Image"
`src` **(required)** *string*	Base64 of an image. Dynamic "${data.src}"
`width` *Integer*	Dynamic "${data.width}"
`height` *Integer*	Dynamic "${data.height}"
`scale-type` *string*	`cover` or `contain` Default value: `contain`
`aspect-ratio` *Number*	Default value: 1 Dynamic "${data.aspect_ratio}"
`alt-text` *string*	Alternative Text is for the accessibility feature, eg. Talkback and Voice over Dynamic "${data.alt_text}"

Image Scale Types

Scale Type	Description
`cover`	Image is clipped to fit the image container. If there is no height value (which is the default), the image will be displayed to its full width with its original aspect ratio. If the height value is set, the image is cropped within the fixed height. Depending on the image whether it is portrait or landscape, image is clipped vertically or horizontally.
`contain`	Image is contained within the image container with the original aspect ratio. If there is no height value (which is the default), the image will be displayed to its full width with its original aspect ratio.

container with the fixed height and the original aspect ratio.

> Developers should consider setting a specific height, width and aspect ratio for images whenever using `contain`. On Android devices WhatsApp sets a default height value of 400, which may create some unwanted spacing.

Example

Flow JSON

```
{
  "version": "7.3",
  "screens": [
    {
      "id": "DEMO_SCREEN",
      "title": "Demo",
      "terminal": true,
      "layout": {
        "type": "SingleColumnLayout",
        "children": [
          {
            "type": "Image",
            "src": "iVBORw0KGgoAAAANSUh
            "width": 200,
            "height": 200
          },
          {
            "type": "Footer",
            "label": "Continue",
            "on-click-action": {
              "name": "complete",
              "payload": {}
            }
          }
        ]
      }
    }
  ]
}
```

Preview

Run

Limits and Restrictions

Type	Limit / Restriction
Max number of images per screen	3
Recommended image size	Up to 300kb
Total data channel payload size	1 Mb
Supported images formats	JPEG PNG

If

> Supported starting with Flow JSON version 4.0

Parameter	Deskripsi
`type` **(required)** *string*	"If"
`condition` **(required)** *string*	Boolean expression, it allows both dynamic and static data. Check section below for more info.
`then` **(required)** *Array of Components*	The components that will be rendered when `condition` is `true`. Allowed components: "TextHeading", "TextSubheading", "TextBody", "TextCaption", "CheckboxGroup", "DatePicker", "Dropdown", "EmbeddedLink", "Footer", "Image", "OptIn", "RadioButtonsGroup", "Switch", "TextArea", "TextInput" and "If"*. It is allowed to nest up to 3 "If" components. > From V7.1 ChipsSelector is also allowed together with all the previous listed components.
`else` *Array of Components*	The components that will be rendered when `condition` is `false`. Allowed components: "TextHeading", "TextSubheading", "TextBody",

"EmbeddedLink", "Footer", "Image", "OptIn", "RadioButtonsGroup", "Switch", "TextArea", "TextInput" and "If"*. It is allowed to nest up to 3 "If" components.

> From V7.1 ChipsSelector is also allowed together with all the previous listed components.

Supported Operators

Operator	Symbol	Types allowed	Description and examples

`Parentheses`	`()`	`boolean`
		`number`
		`string`
			It is used to define the precedence of operations. Or if you want to perform boolean operations that one of the sides is a result of a number or string comparison. It always require an operation within it. One expression can contain multiple parentheses. Examples:
			• `${form.opt_in} \|\| (${data.num_value} > 5)`
			• `${form.opt_in} && (${form.address} != '')`
			• `!${form.value1}`

`Equal to`	`==`	`boolean`
		`number`
		`string`
			It is used to compare booleans, numbers and strings. Both sides should have the same type and at least one of them should contain a dynamic variable. Examples:
			• `${form.opt_in} == true`
			• `${data.num_value} == 5`
			• `${form.city} == 'London'`

`Not equal to`	`!=`	`boolean` `number`	It is used to compare booleans, numbers and strings. Both sides should have the same type and at least one of them should contain a dynamic variable. Examples: • `${form.opt_in} != true`

		`string`	• `${data.num_value} != 5` • `${form.city} != 'London'`
`AND`	`&&`	`boolean`	It performs the boolean `AND` operation. It evaluates as true only if both sides are true. This operator has high priority, i.e. it will be evaluated before other operators. The exception is parentheses, if one of the sides contain an opening or closing parenthesis, then the parenthesis is evaluated first. Example: • `${form.opt_in} && ${data.boolean_value}`
`OR`	`||`	`boolean`	It performs the boolean `OR` operation. It evaluates as true if at least one side is true. Example: • `${form.opt_in} || ${data.boolean_value}`
`NOT`	`!`	`boolean`	It performs the boolean `NOT` operation. It negates the statement after it. It can be used before immediately `boolean` values or parentheses (that will result into boolean values) Examples: • `!(${form.opt_in} || ${data.boolean_value})` • `!(${data.num_value} > 5)` • `!${form.value1}`
`Greater than`	`>`	`number`	It is used to compare to numbers. At least one of them should be a dynamic variable. Examples: • `${data.num_value} > 5` • `${data.num_value} > ${data.num_value2}`
`Greater than or`	`>=`	`number`	It is used to compare to numbers. At least one of them should be a dynamic variable. Examples:

equal to			• ${data.num_value} >= 5 • ${data.num_value} >= ${data.num_value}
Less than	<	number	It is used to compare to numbers. At least one of them should be a dynamic variable. Examples: • ${data.num_value} < 5 • ${data.num_value} < ${data.num_value2}
Less than or equal to	<=	number	It is used to compare to numbers. At least one of them should be a dynamic variable. Examples: • ${data.num_value} == 5 • ${data.num_value} <= ${data.num_value}

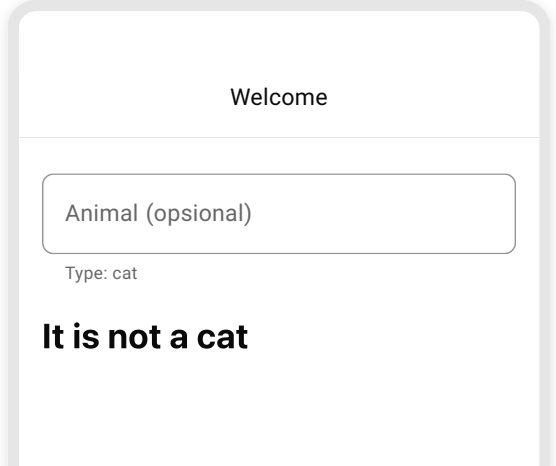
Example

Flow JSON

```
{
  "version": "7.3",
  "screens": [
    {
      "data": {
        "value": {
          "type": "boolean",
          "__example__": true
        }
      },
      "id": "SCREEN",
      "layout": {
        "type": "SingleColumnLayout",
        "children": [
          {
            "type": "TextInput",
            "label": "Animal",
            "name": "animal",
            "helper-text": "Type: cat"
          },
```

Preview

Run



```
        {
          "type": "TextHeading",
          "text": "It is a cat"
        }
      ],
      "else": [
        {
          "type": "TextHeading",
          "text": "It is not a ca
        }
      ]
    },
```


Rules

Condition

- Should have at least one dynamic value (e.g. `${data...}` or `${form...}`).
- Should always be resolved into a boolean (i.e. no strings or number values).
- Can be used with literals but should not only contain literals.

Footer

- `Footer` can be added within `If` only in the first level, not inside a nested `If`.
- If there is a `Footer` within `If`, it should exist in both branches (i.e. `then` and `else`). This means that `else` becomes mandatory.
- If there is a `Footer` within `If` it cannot exist a footer outside, because the max count of `Footer` is 1 per screen.

Limitations and restrictions

The table below show examples of limitations and validation errors that will be shown for certain cases.

Scenario	Validation error shown
• `Given` there is a footer component inside `then` • `And else` is not defined • `When` validating the flow • `Then` it should show a validation error	Missing Footer inside one of the if branches. Branch "else" should exist and contain one Footer.

• Given there is a footer component inside then • And there is no footer inside else • When validating the flow • Then it should show a validation error	Missing Footer inside one of the if branches.
• Given there is no footer component inside then • And there is a footer inside else • When validating the flow • Then it should show a validation error	Missing Footer inside one of the if branches.
• Given there is a footer component inside then • And there is a footer component inside else • And there is a footer component outside the If • When validating the flow • Then it should show a validation error	You can only have 1 Footer component per screen.
• Given there is an empty array defined for then • When validating the flow • Then it should show a validation error	Invalid value found at: "$root/screens/path_to_your_component/then" due to empty array. It should contain at least one component.

Switch

Supported starting with Flow JSON version 4.0

Parameter	Deskripsi
`type` **(required)** *string*	"Switch"
`value` **(required)** *string*	A variable that will have its value evaluated during runtime. Example - `${data.animal}`

`cases` **(required)** *Map of Array of Components*	Each property is a key (string) that maps to an Array of Components. When the `value` matches the key, it renders its array of components. Allowed components: "TextHeading", "TextSubheading", "TextBody", "TextCaption", "CheckboxGroup", "DatePicker", "Dropdown", "EmbeddedLink", "Footer", "Image", "OptIn", "RadioButtonsGroup", "TextArea", "TextInput". From V7.1 ChipsSelector is also allowed together with all the previous listed components.

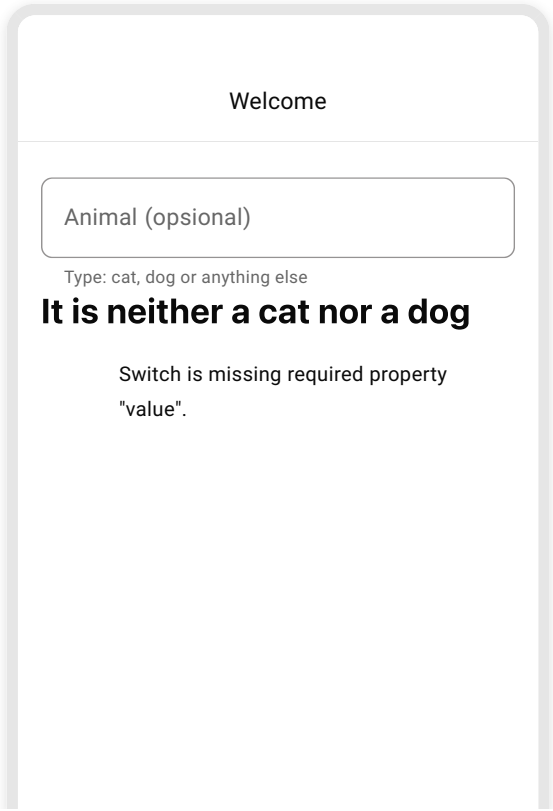
Example

Flow JSON

```
{
  "version": "7.3",
  "screens": [
    {
      "data": {
        "value": {
          "type": "string",
          "__example__": "cat"
        }
      },
      "id": "SCREEN",
      "layout": {
        "type": "SingleColumnLayout",
        "children": [
          {
            "type": "TextInput",
            "label": "Animal",
            "name": "animal",
            "helper-text": "Type: cat,
          },
          {
            "type": "Switch",
            "value": "${form.animal}",
            "cases": {
              "cat": [
                {
                  "type": "TextHeading"
                  "text": "It is a cat"
                }
              ],
              "dog": [
                {
                  "type": "TextHeading"
                  "text": "It is a dog"
```

Preview

Run



Rules

Cases

- Should have at least one value. It cannot be empty (e.g. `"cases": {}`)

Limitations and restrictions

The table below show examples of limitations and validation errors that will be shown for certain cases.

Scenario	Validation error shown
• `Given` there is a `Switch` component • `And` its `cases` property is empty • `When` validating the flow • `Then` it should show a validation error	Invalid empty property found at: "$root/screens/path_to_your_component/cases".

Media upload

Please refer to the specific page for media upload components.

Navigation List

Supported from Flows v6.2+.

The NavigationList component allows users to navigate effectively between different screens in a Flow, by exploring and interacting with a list of options. Each list item can display rich content such as text, images and tags.

Parameter	Deskripsi
`type` **(required)** *string*	"NavigationList"
`name`	

`list-items` **(required)** *array*	Dynamic "${data.list_items}"
`label` *string*	Dynamic "${data.label}"
`description` *string*	Dynamic "${data.description}"
`media-size` *enum*	{'regular','large'} Default: 'regular' Dynamic "${data.media-size}"
`on-click-action` *action*	`data_exchange` and `navigate` are supported.

Each item in the list of items supports the following properties:

Parameter	Deskripsi
`main-content` **(required)** *object*	• **(required)** title <string> • description <string> • metadata <string>
`end` *object*	• title <string> • description <string> • metadata <string>
`start` *object*	• **(required)** image <base64 encoding of an image> • alt-text <string>
`badge` *string*	
`tags` *Array<string>*	
`on-click-action` *action*	`data_exchange` and `navigate` are supported.

The `on-click-action` is required for the component, and it can be defined either:

- Once at component-level and it will apply the same action for all items in the list.
- Individually, on each item in the list to allow for different actions to be triggered.

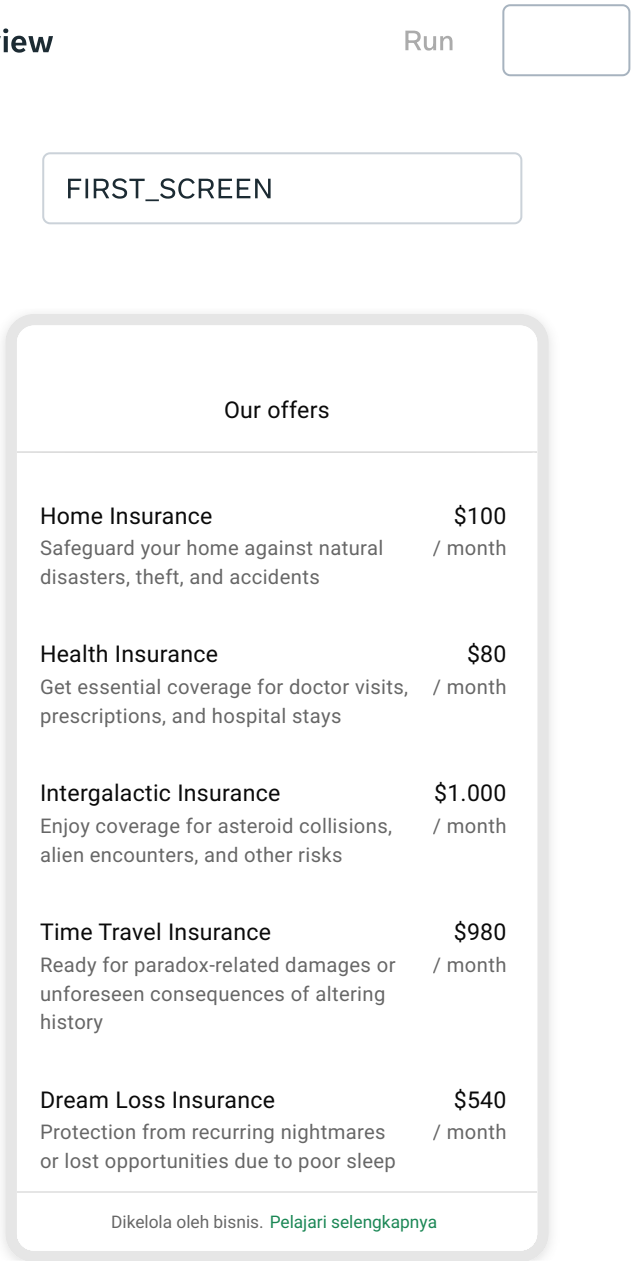
Example

Flow JSON

```
{
  "version": "7.3",
  "routing_model": {
    "FIRST_SCREEN": [
      "SECOND_SCREEN",
      "THIRD_SCREEN",
      "FIFTH_SCREEN"
    ],
    "SECOND_SCREEN": [
      "CONTACT"
    ],
    "THIRD_SCREEN": [
      "CONTACT"
    ],
    "FIFTH_SCREEN": [
      "CONTACT"
    ],
    "CONTACT": []
  },
  "screens": [
    {
      "id": "FIRST_SCREEN",
      "title": "Our offers",
      "data": {},
      "layout": {
        "type": "SingleColumnLayout",
        "children": [
          {
            "type": "NavigationList",
            "name": "insurances",
            "list-items": [
              {
                "id": "home",
                "main-content": {
                  "title": "Home Insura
                  "metadata": "Safeguar
```

Preview

Run



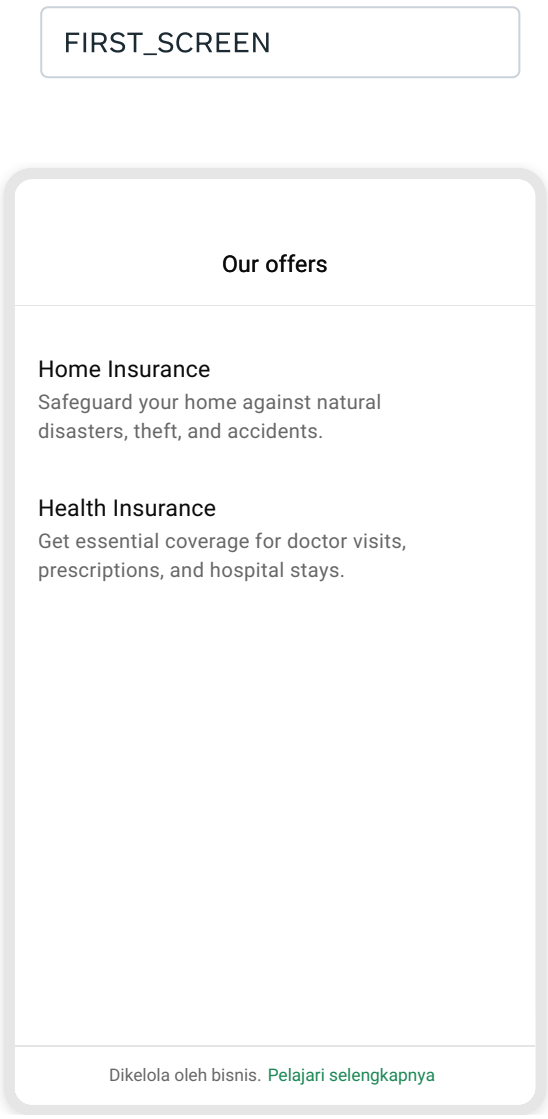
Dynamic Example

`array`, which will be of type `object`. Then inside the `items` object, you have a `properties` dictionary with `id` and `main-content` just like in the static declaration. Both `id` will always be of type `string` and `main-content` will always be of type `object`, and accompanied by a definition of its structure. Within the `insurances` array, you must define concrete examples in the `__example__` field.

Flow JSON

```
{
  "version": "7.3",
  "data_api_version": "3.0",
  "routing_model": {
    "FIRST_SCREEN": [
      "SECOND_SCREEN",
      "THIRD_SCREEN"
    ],
    "SECOND_SCREEN": [
      "CONTACT"
    ],
    "THIRD_SCREEN": [
      "CONTACT"
    ],
    "CONTACT": []
  },
  "screens": [
    {
      "id": "FIRST_SCREEN",
      "title": "Our offers",
      "data": {
        "insurances": {
          "type": "array",
          "items": {
            "type": "object",
            "properties": {
              "id": {
                "type": "string"
              },
              "main-content": {
                "type": "object",
                "properties": {
                  "title": {
                    "type": "string"
                  }
                }
```

Preview Run



Limits and Restrictions

- The `Navigation List` component cannot be used on a terminal screen.
- There can be at most 2 `Navigation List` components per screen.

- There can be only one item with a `badge` per list.
- The `end` add-on cannot be used in combination with `media-size` set to `large`.
- The `on-click-action` cannot be defined simultaneously on component-level and on item-level.

Component restrictions

Property	Limit / Restriction
list-items	minimum 1 and maximum 20 items Content will not be rendered if the limit is reached
label	80 characters Content will truncate if the limit is reached
description	300 characters Content will truncate if the limit is reached

List items restrictions

Content over the limit specified will not be rendered.

Add-on / property	Property	Limit / Restriction
start	image	100KB Images over the limit will be replaced by a placeholder
main-content	title	30 characters

	description	20 characters
	metadata	80 characters
end	title	10 characters
	description	10 characters
	metadata	10 characters
badge		15 characters
tags		15 characters
		3 items

Chips Selector

Chips Selector component allows users to pick multiple selections from a list of options.

> Supported starting with Flow JSON version 6.3

Parameter	Deskripsi
`type` **(required)** *string*	"ChipsSelector"
`data-source` **(required)** *Array*	Dynamic "${data.data_source}" • *Array< id: String, title: String, enabled: Boolean, on-select-action: {name: 'update_data', payload: {...}}, on-unselect-action: {name: 'update_data', payload: {...}} >*

`name` **(required)** *String*	
`min-selected-items` *Integer*	Dynamic "${data.min_selected_items}"
`max-selected-items` *Integer*	Dynamic "${data.max_selected_items}"
`enabled` *Boolean*	Dynamic "${data.is_enabled}"
`label` **(required)** *string*	Dynamic "${data.label}"
`required` *Boolean*	Dynamic "${data.is_required}"
`visible` *Boolean*	Dynamic "${data.is_visible}" Default: True
`description` *String*	Dynamic "${data.description}"
`init-value` *Array<String>*	Dynamic "${data.init-value}" Only available when component is outside Form component
`error-message` *String*	Dynamic "${data.error-message}" Only available when component is outside Form component
`on-select-action` *Action*	`data_exchange` and `update_data` are supported. **update_data** • Supported starting with Flow JSON version 7.1
`on-unselect-action` *Action*	Only `update_data` is supported. • Supported starting with Flow JSON version 7.1 • In V7.1, if `on-unselect-action` is not added, `on-select-action` will continue to handle both selection and unselection events. However, if `on-unselect-action` is defined, it will exclusively handle unselection, while `on-select-action` will be used solely for selection.

exclusively handle unselection, while `on-select-action` will be used solely for selection.

Limits and Restrictions

Type	Limit / Restriction
Label	80 Characters
Description	300 Characters
Min # of options	2
Max # of options	20

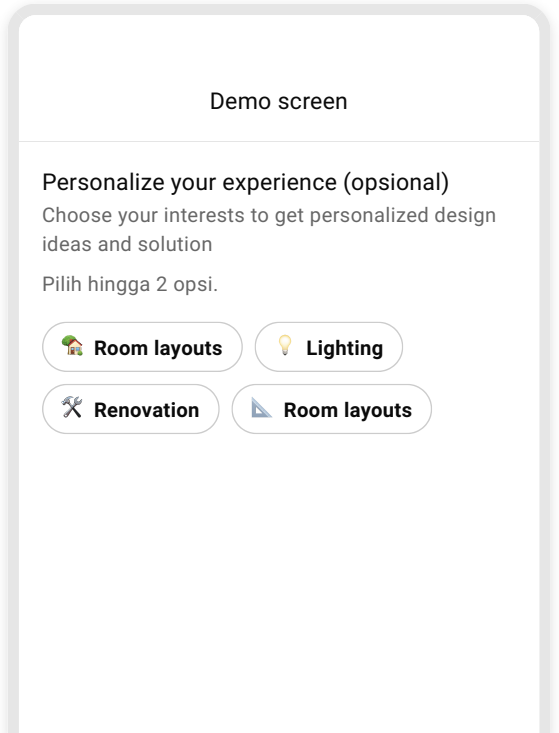
Example

Flow JSON

```
{
  "version": "6.3",
  "screens": [
    {
      "id": "DEMO_SCREEN",
      "terminal": true,
      "title": "Demo screen",
      "layout": {
        "type": "SingleColumnLayout",
        "children": [
          {
            "type": "ChipsSelector",
            "name": "chips",
            "label": "Personalize your
            "description": "Choose your
            "max-selected-items": 2,
            "data-source": [
              {
                "id": "room_layout",
                "title": "🏠 Room layou
              },
              {
                "id": "lighting",
                "title": "💡 Lighting"
              },
              {
                "id": "renovation",
                "title": "🛠️ Renovatior
              },
              {
                "id": "furnitures"
```

Preview

Run



```
        },
        {
```

Image Carousel

The Image Carousel component allows users to slide through multiple images.

> Supported from Flows v7.1+.

Parameter	Deskripsi
`type` **(required)** *string*	"ImageCarousel"
`images` **(required)** *array*	Dynamic "${data.images}"
`aspect-ratio` *string*	Either "4:3" or "16:9". Default: "4:3".
`scale-type` *string*	Either "contain" or "cover". Default: "contain".

Each item in the list of images supports the following properties:

Parameter	Deskripsi
`src` **(required)** *string*	Base64 of an image.
`alt-text` **(required)** *string*	Alternative text for for accessibility purposes.

Limits and Restrictions

Type	Limit / Restriction
Min # of images	1
Max # of images	3

screen Max # of ImageCarousel per Flow	3

Example

Flow JSON

```
{
  "version": "7.1",
  "screens": [
    {
      "id": "DEMO_SCREEN",
      "terminal": true,
      "title": "Demo screen",
      "layout": {
        "type": "SingleColumnLayout",
        "children": [
          {
            "type": "ImageCarousel",
            "scale-type": "cover",
            "images": [
              {
                "alt-text": "Landscape
                "src": "iVBORw0KGgoAAAA
              },
              {
                "alt-text": "Square ima
                "src": "iVBORw0KGgoAAAA
              },
              {
                "alt-text": "Portrait i
                "src": "iVBORw0KGgoAAAA
              }
            ]
          },
          {
            "type": "Footer",
            "label": "Continue",
            "on-click-action": {
              "name": "complete",
              "payload": {}
            }
          }
```

Preview

Run

Here's a corrected version:

If you check the attribute model of certain components (`Dropdown`, `DatePicker`, `RadioGroup` and `CheckboxGroup`), you will find that some of them accept the `on-xxxx-action` attribute. This attribute allows the component to trigger a data-exchange action. It can be used in the following scenarios:

1. When a user selects a date in the DatePicker component.
2. When the business needs to fetch available data (such as table slots, tickets, etc.) for this selected date by calling a data_exchange action.
3. Once the data is received, the user will see an updated screen with new data.

Prerequisites

The following steps require communication between the client and the business server. Please ensure that you have configured the data channel before attempting to use this feature.

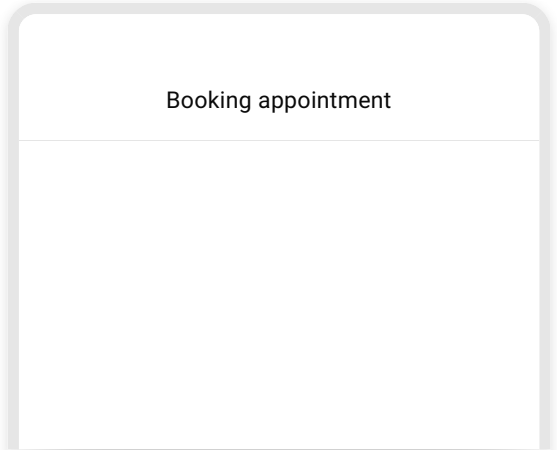
Step 1 - Defining the layout

Let's begin with a minimal example, consisting of an empty form and a CTA button, and gradually add more components.

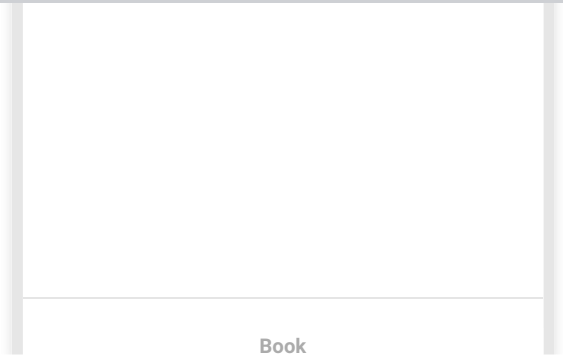
Flow JSON

```
{
  "version": "7.3",
  "routing_model": {
    "BOOKING": []
  },
  "screens": [
    {
      "id": "BOOKING",
      "terminal": true,
      "title": "Booking appointment",
      "layout": {
        "type": "SingleColumnLayout",
        "children": [
          {
            "type": "Form",
            "name": "booking-form",
            "children": [
              {
                "type": "Footer",
                "label": "Book".
```

Preview Run



```
                }
              }
            ]
          }
        ]
      }
    }
  ]
}
```



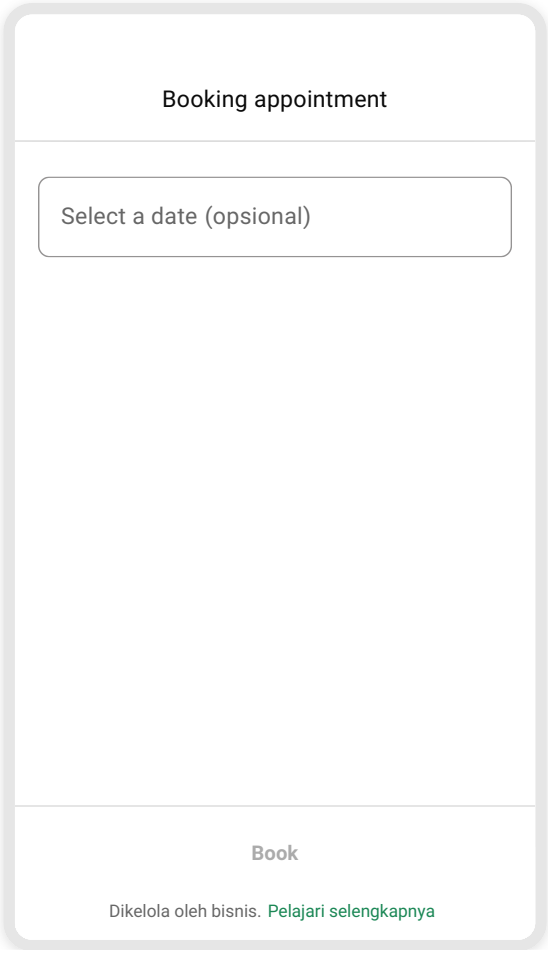
So, we want to build a simple form that takes a date and displays the list of available time slots. First, we'll add a `DatePicker` component:

Flow JSON

```
{
  "version": "7.3",
  "routing_model": {
    "BOOKING": []
  },
  "screens": [
    {
      "id": "BOOKING",
      "terminal": true,
      "title": "Booking appointment",
      "layout": {
        "type": "SingleColumnLayout",
        "children": [
          {
            "type": "Form",
            "name": "booking-form",
            "children": [
              {
                "type": "DatePicker",
                "label": "Select a date
                "name": "date"
              },
              {
                "type": "Footer",
                "label": "Book",
                "on-click-action": {
                  "name": "complete",
                  "payload": {}
                }
              }
            ]
          }
        ]
      }
    }
```

Preview

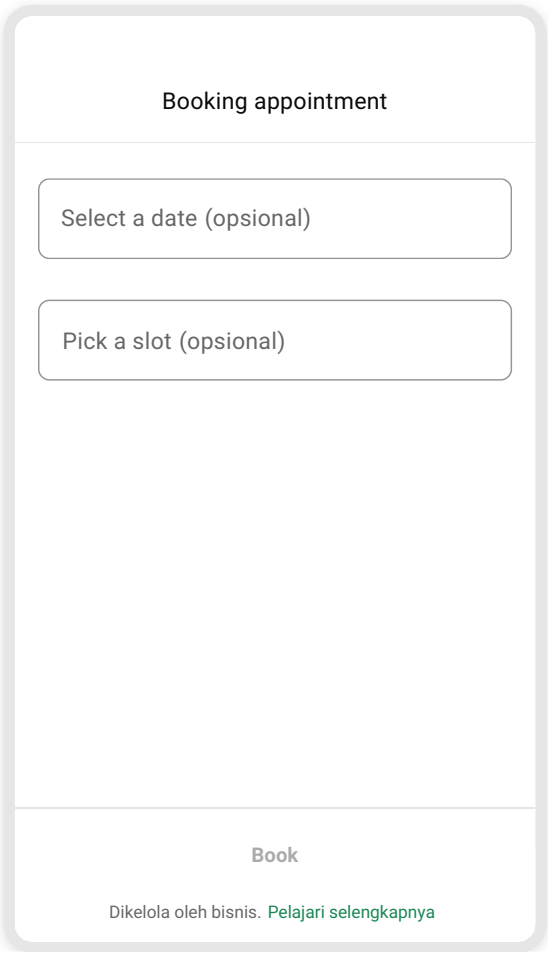
Run



Flow JSON

```
{
  "version": "7.3",
  "routing_model": {
    "BOOKING": []
  },
  "screens": [
    {
      "id": "BOOKING",
      "terminal": true,
      "title": "Booking appointment",
      "layout": {
        "type": "SingleColumnLayout",
        "children": [
          {
            "type": "Form",
            "name": "booking-form",
            "children": [
              {
                "type": "DatePicker",
                "label": "Select a date
                "name": "date"
              },
              {
                "type": "Dropdown",
                "label": "Pick a slot",
                "name": "selected_slot"
                "data-source": [
                  {
                    "id": "1",
                    "title": "13.00"
                  }
                ]
              },
              {
                "type": "Footer",
                "label": "Book",
```

Preview Run



Step 2 - Defining 3P Data

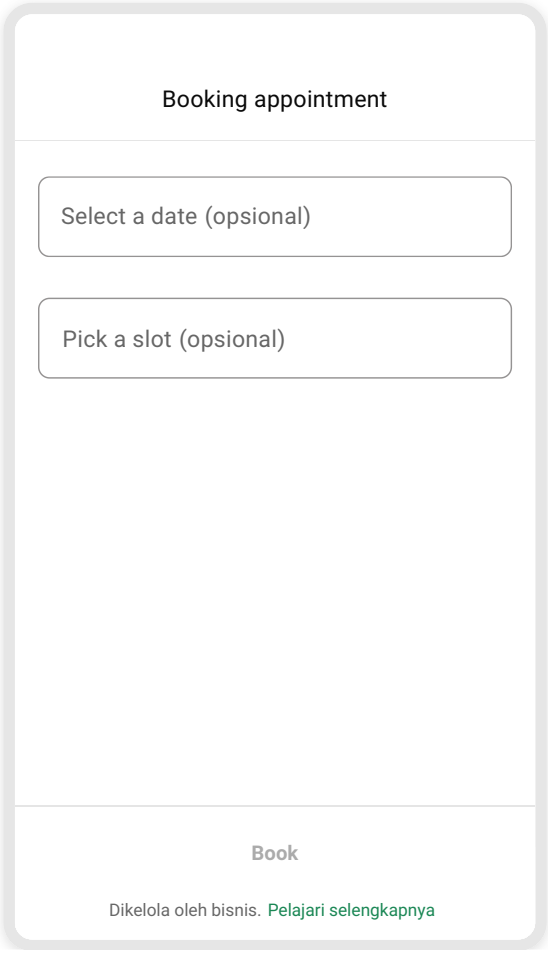
Until now, we've been incorporating static mock data, but now we aim to connect a screen with dynamic data. Dynamic data can originate from various sources:

1. Initial message payload
2. `navigate` - transitioning from the previous screen using a `navigate` action
3. `data_exchange` - a request to the business server

Flow JSON

```
{
  "version": "7.3",
  "data_api_version": "3.0",
  "routing_model": {
    "BOOKING": []
  },
  "screens": [
    {
      "id": "BOOKING",
      "terminal": true,
      "title": "Booking appointment",
      "layout": {
        "type": "SingleColumnLayout",
        "children": [
          {
            "type": "Form",
            "name": "booking-form",
            "children": [
              {
                "type": "DatePicker",
                "label": "Select a date
                "name": "date"
              },
              {
                "type": "Dropdown",
                "label": "Pick a slot",
                "name": "selected_slot'
                "data-source": [
                  {
                    "id": "1",
                    "title": "13.00"
                  }
                ]
              },
              {
                "type": "Footer",
```

Preview Run



Step 3 - Allowing DatePicker to Make a Request to the Server

Let's provide `"on-select-action"` to the `DatePicker` component so we can execute the call to the business server. In the `payload`, we can pass any data we want to the business server to understand the type of request.

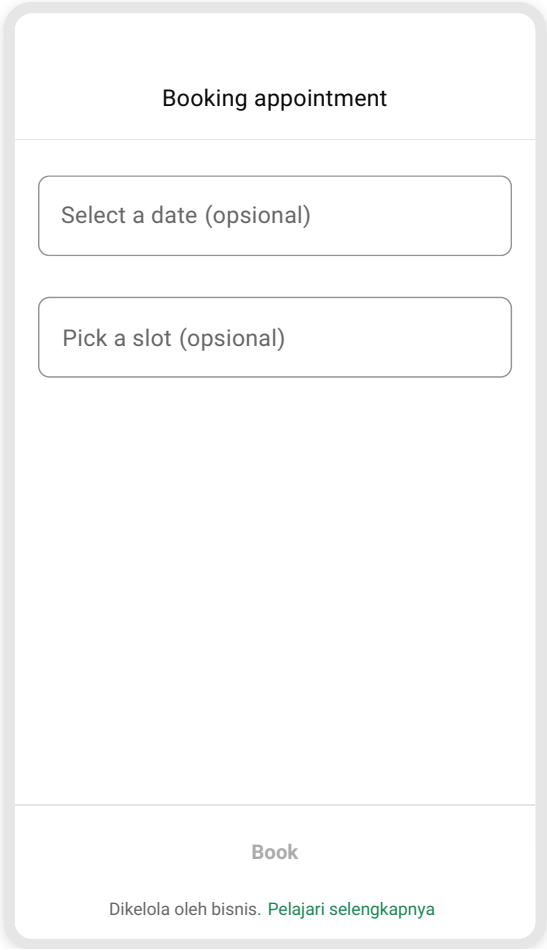
```
{
   "on-select-action":{
      "name":"data_exchange",
```

```
      }
    }
}
```

Flow JSON

```
{
  "version": "7.3",
  "data_api_version": "3.0",
  "routing_model": {
    "BOOKING": []
  },
  "screens": [
    {
      "id": "BOOKING",
      "terminal": true,
      "title": "Booking appointment",
      "layout": {
        "type": "SingleColumnLayout",
        "children": [
          {
            "type": "Form",
            "name": "booking-form",
            "children": [
              {
                "type": "DatePicker",
                "label": "Select a date
                "name": "date"
              },
              {
                "type": "Dropdown",
                "label": "Pick a slot",
                "name": "selected_slot"
                "data-source": [
                  {
                    "id": "1",
                    "title": "13.00"
                  }
                ],
                "on-select-action": {
                  "name": "data_exchang
                  "payload": {
```

Preview Run



In this example, we'll send the value of the field `date` to the action payload, and we'll also add some static data `"component_action": "update_date"` to help the server recognize the type of request. There is no strict format here; you can choose whatever works for your case.

Now when you try to select a date, a `data_exchange` request will be executed. The server may return the data that can change the UI. For now, our Flow doesn't expect or use any data from the server. Let's fix it by first defining the data model that we expect for a screen.

Step 4 - Define a Server Data Model

Let's declare a `data` property for the screen outlining the data that we expect to receive from the server. So, we want to receive an `available_slots` array with timeslot options.

It should have the following model. The `__example__` field is mock data used to display the data within the web preview.

```
{
    "available_slots": {
        "type": "array",
        "items": {
            "type": "object",
            "properties": { "id": {"type": "string"}, "title": {"type": "string"
        },
        "__example__": [ {"id": "1", "title": "08:00"}, {"id": "2", "title": "09
    }
}
```

It means that the expected payload to be returned from server can look like the following:

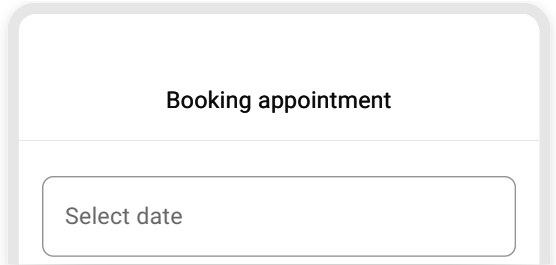
```
{
    "version": "3.0",
    "screen": "BOOKING",
    "data": {
       "available_slots": [ {"id": "1", "title": "08:00"}, {"id": "2", "title":
    }
}
```

So you Flow JSON now should look like the following:

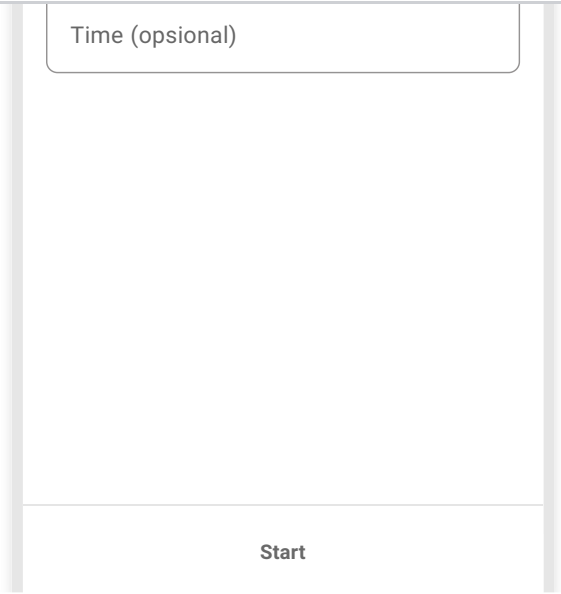
Flow JSON

Preview Run

```
{
  "version": "7.3",
  "data_api_version": "3.0",
  "routing_model": {
    "BOOKING": []
  },
  "screens": [
    {
      "id": "BOOKING",
      "terminal": true,
      "title": "Booking appointment",
      "data": {
        "available_slots": {
```



```
      "properties": {
        "id": {
          "type": "string"
        },
        "title": {
          "type": "string"
        }
      }
    },
    "__example__": [
      {
        "id": "1",
        "title": "08:00"
      },
      {
        "id": "2",
        "title": "09:00"
      }
    ]
  }
```



Step 5 - Control Visibility of the Component

Now, when we select a date in `DatePicker`, the application will send a request to the business server to get available timeslots. However, we don't want a `Dropdown` to be visible until there is data to display. How can we hide it?

For this purpose, we can use the `visible` attribute on `Dropdown` and connect it with server data. The business server can control the visibility of the component based on a set condition.

So, we need to make the following changes:

1. Define `is_dropdown_visible` in the `data` model of the screen.
2. Connect a property via dynamic binding `"visible": "${data.is_dropdown_visible}"`.
3. Ensure that the server returns the correct data.

Let's update our code:

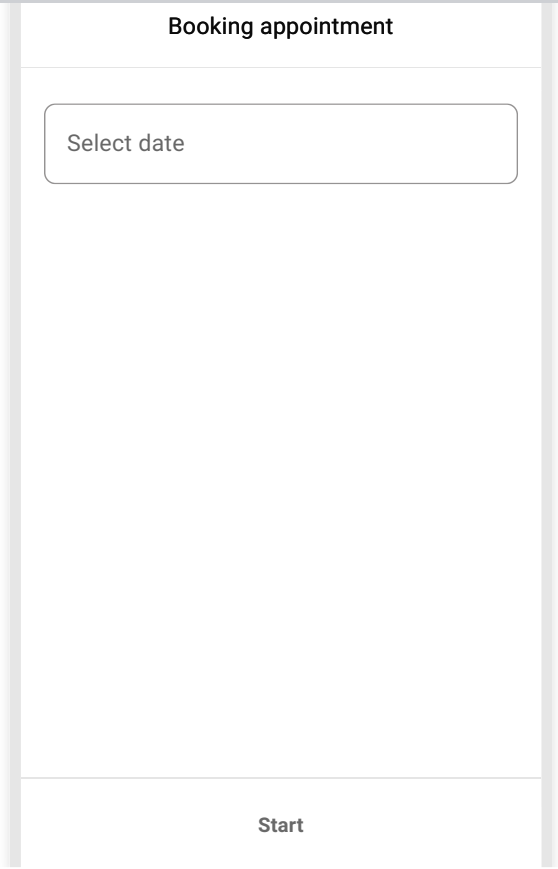
NOTE: The current version of the playground doesn't support endpoint requests

Flow JSON

Preview Run

```
{
  "version": "7.3",
  "data_api_version": "3.0",
  "routing_model": {
```

```
{
  "id": "BOOKING",
  "terminal": true,
  "title": "Booking appointment",
  "data": {
    "available_slots": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "id": {
            "type": "string"
          },
          "title": {
            "type": "string"
          }
        }
      },
      "__example__": [
        {
          "id": "1",
          "title": "08:00"
        },
        {
          "id": "2",
          "title": "09:00"
        }
      ]
    },
```



Summary

That's it! Now you have a dynamic component set up. If you're facing any challenges, feel free to ask a question on the developer forum. We'll be happy to help!

Sebelumnya
Flow JSON

Berikutnya
Flows API