Flow JSON

Introduction

Flow JSON enables businesses to create workflows in WhatsApp by accessing the features of WhatsApp Flows using a custom JSON object developed by Meta.

These workflows are initiated, run, and managed entirely inside WhatsApp. It can include multiple screens, data flows, and response messages.

Flow JSON consists of the following sections:

Flow JSON Section	Description
Screen Data Model	Commands to define static types that power the screen.
Screens	Used to compose layouts using standard UI library components.
Components	Individual building blocks that make up a screen (text fields, buttons, and so on).
Routing Model	Defines the rules for the screen by limiting the possible state transition. For example, developers can define that from Screen 1 you can only transition to Screen 2 and 3, but not Screens 4 and 5. These rules are used during server / client side payload validations.
Actions	Special type of syntax to invoke pre-defined logic on the client. Allowed actions are: `navigate`, `data_exchange`, or `complete`. From Flow JSON version 6.0 and later, `open_url` and `update_data`, is also allowed.

Top-level Flow JSON Properties

Flow JSON has several required and optional properties that are used in the process of compilation and validation of the Flow.

Required properties

- `version` - represents the version of Flow JSON to use during the compilation. Please refer to the list of versions for more details.

- `screens` - represents an array or screen as part of the user experience. This is like a set of different pages on your website.

Optional properties

- `routing_model` - represents a routing ruling system. The routing model is generated automatically if your Flow doesn't use a Data Endpoint. If it does, the validation system will ask you to provide a routing model.

- `data_api_version` - represents the version to use during communication with the WhatsApp Flows Data Endpoint. Currently, it is `3.0`. If flow uses the data-channel capability, the validation system will ask to provide this property.

- `data_channel_uri` - represents the URL of the WhatsApp Flows Data Endpoint. If a Flow uses the data-channel capability, the validation system will ask to provide this property.

`data_channel_uri` **is not supported by Flow JSON as of version** `3.0`**. For Flow JSON** `3.0`**, please configure the URL of your Flow Data Endpoint using the** `endpoint_uri` **field provided by the** [**Flows API**]()**.**

```
{
 "version": "2.1",
 "data_api_version": "3.0",
 "routing_model": {"MY_FIRST_SCREEN": ["MY_SECOND_SCREEN"] 
 "screens": [...],
 "data_channel_uri": "https://example.com"
}
```

```
{
 "version": "3.1",
 "data_api_version": "3.0",
 "routing_model": {"MY_FIRST_SCREEN": ["MY_SECOND_SCREEN"] 
 "screens": [...]
}
```

Screens

Screens are the main unit of a Flow. Each screen represents a single node in the state machine you define. These properties then make up the Flows screen property model:

```
"screen" : {
  "id": string,
  "terminal": ?boolean,
  "success": ?boolean,
  "title": ?string,
  "refresh_on_back": ?boolean,
  "data": ?object,
  "layout": object
}
```

Required properties

- `id` - unique identifier of the screen which works as a page url. SUCCESS is a reserved keyword and should not be used as a screen id.

- `layout` - associated screen UI Layout that is shown to the user. Layout can be predefined or it can represent a container with fully customizable content built using WhatsApp Flows Library.

Optional properties

- `terminal` (optional) - the business flow is the end state machine. It means that each Flow should have a terminal state where we terminate the experience and have the Flow completed. Multiple screens can be marked as terminal. It's mandatory to have a Footer component on the terminal screen.

- `data` (optional) - declaration of dynamic data that fills the components field in the Flow JSON. It uses JSON Schema to define the structure and type of the properties. Below you can find the simple example.

```
{
  "data": {
    "first_name": {
      "type": "string",
      "__example__": "John"
    }
  }
}
```

- `title` (optional) - screen level attribute that is rendered in the top navigation bar.

- `success` - (optional, only applicable on terminal screens) - Defaults to `true`. A Flow can have multiple terminal screens with different business outcomes. This property marks whether terminating on a terminal screen should be considered a successful business outcome.

- `refresh_on_back` (optional, and only applicable for Flows with a Data Endpoint) - Defaults to `false`. This property defines whether to trigger a data exchange request with the WhatsApp Flows Data Endpoint when using the back button while on this screen. The property is useful when you need to reevaluate the screen data when returning to the previous screen (example below).

- `sensitive` - (optional, only applicable for Flows version 5.1 and above) - Defaults to an empty array. When a Flow is completed, users will see a new message UI, which can be clicked and users will be able to view the responses submitted by them. This array contains the names of the fields in the screen that contain sensitive data, and should be hidden in the response summary for the consumers.

Additional information on `refresh_on_back`

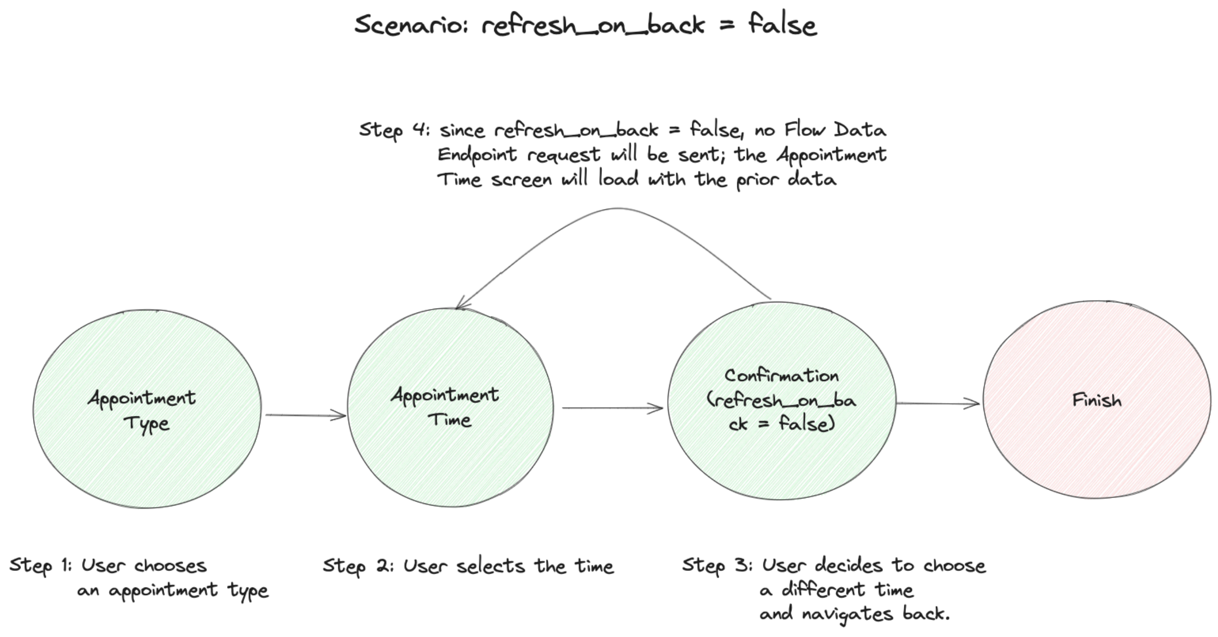
Given a simple Flow example with the following screens:

1. User chooses an `Appointment Type`.
2. User selects `Appointment Time`.
3. User `Confirms` an appoinment.

The user may navigate back from the confirmation page to re-select an appointment time. By using the `refresh_on_back` property in the `Confirmation` screen's definition, you can control whether to refresh the list of available times, or to reload the previously shown list.

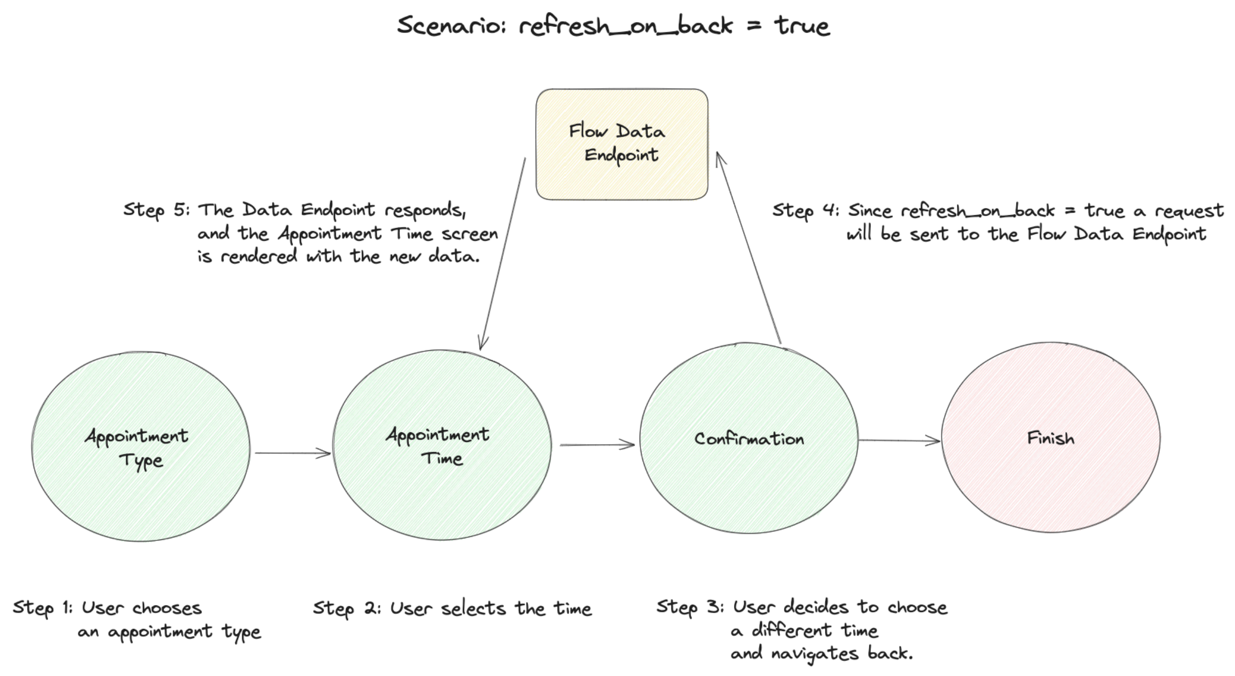
refresh_on_back=false (default)

If `refresh_on_back=false`, when the user goes back to the `Appointment Time` screen the Flow will not request the Flow Data Endpoint and the screen will be loaded with the previously provided data, and the user's prior input. This is the preferred behavior in most cases, given it avoids a roundtrip to the Flow Data Endpoint and provides a snappier experience for the user.


Scenario: refresh_on_back = false
Step 4: since refresh_on_back = false, no Flow Data Endpoint request will be sent; the Appointment Time screen will load with the prior data
Appointment Type
Appointment Time
Confirmation (refresh_on_back = false)
Finish
Step 1: User chooses an appointment type
Step 2: User selects the time
Step 3: User decides to choose a different time and navigates back.

refresh_on_back=true

If, however, you need to revalidate or update the data for the screen, set `refresh_on_back=true` on the screen from which the back action is triggered (the `Confirmation` screen in this example). Once the user navigates back to the `Appointment Time` screen, the Flow will send a request to the Flow Data Endpoint and display the screen with the data from the response.


Scenario: refresh_on_back = true
Flow Data Endpoint
Step 5: The Data Endpoint responds, and the Appointment Time screen is rendered with the new data.
Step 4: Since refresh_on_back = true a request will be sent to the Flow Data Endpoint
Appointment Type
Appointment Time
Confirmation
Finish
Step 1: User chooses an appointment type
Step 2: User selects the time
Step 3: User decides to choose a different time and navigates back.

Flow Data Endpoint Request payload in case of `refresh_on_back=true`

The complete payload structure is [defined here](#) - in this case the `action` field will be `BACK`, and the `screen` will be set to the name of the `Confirmation` screen.

Additional information on `sensitive` fields

Given a Flow in which we mark certain fields as `sensitive`, we use the following masking configuration to display the summary upon the Flow's completion:

Component	Masking	Consumer experience
Text Input	✅	Masked input value (••••••••••••)
Password / OTP	❌	Hidden completely
Text Area	✅	Masked input value (••••••••••••)

Date Picker	✅	Masked input value (••••••••••••)
Dropdown	✅	Masked input value (••••••••••••)
Checkbox Group	✅	Masked input value (••••••••••••)
Radio Buttons Group	✅	Masked input value (••••••••••••)
Opt In	❌	Display as-is (no masking needed)
Document Picker	✅	Hidden uploaded documents completely
Photo Picker	✅	Hidden uploaded media completely