



TIBCO Flogo® Extension for Visual Studio Code

User Guide

Version 1.3.2 | July 2025

Contents

Contents	2
App Development	6
Creating and Managing a Flogo App	6
Creating a Flogo App	6
Validating Your App Flow	8
Editing an App	9
Renaming an App	9
Editing the Version of an App	10
Using Notes	11
Deleting an App	12
Importing an App into an Existing App	12
Creating Flows and Triggers	15
Flows	16
Triggers	50
Playing a Test Case Once	50
Synchronizing a Schema Between Trigger and Flow	54
Data Mappings	55
Data Mappings Interface	55
Mapping Data from the Data Mappings Interface	58
Scopes in Data Mappings	61
Data Types	63
Reserved Keywords to be Avoided in Schemas	64
Mapping Different Types of Data	66
Mapping Data by Using if/else Conditions	98
Using Functions	102
Using Expressions	103
Combining Schemas Using Keywords	105

Supported Operators	109
General Category Triggers, Activities, and Connections	109
Triggers	110
Activities	128
Connections	167
File Category Trigger and Activities	170
File Poller Trigger	171
Create File	172
Read File	174
List Files	175
Write File	176
Copy File	177
Rename File	179
Remove File	180
Archive Files	181
Unarchive Files	182
Developing APIs	183
Using an OpenAPI Specification	183
Using GraphQL Schema	187
Using App Properties, Schemas, and Specs	188
App Properties	188
App Schemas	204
App Specs	208
Using Connectors	213
Supported Connectors	213
Prerequisites for Connectors	213
Creating Connections	227
Editing Connections	228
Deleting Connections	228
Developing for Lambda	229
Creating a Connection with the AWS Connector	229
Creating a Flow with Receive Lambda Invocation Trigger	232

Receive Lambda Invocation	235
InvokeLambdaFunction	237
Using Extensions	239
Configuring Extensions	239
Deleting Extensions	241
Unit Testing	241
Creating and Running a Test Case	242
Creating and Running a Test Suite	250
Unit Testing for the CI/CD	251
App Configuration Management	253
Consul	254
Using Consul	254
Consul Connection Parameters	255
Setting the Consul Connection Parameters	257
AWS Systems Manager Parameter Store	259
Using the Parameter Store	259
Parameter Store Connection Parameters	260
Setting the Parameter Store Connection Parameters	262
Overriding an App Property at Runtime	265
Overriding Properties without Restarting or Redeploying the App	265
Overriding Values by Specifying New Values	266
Overriding Values by Specifying New Values in the API Directly	266
Azure Key Vault Secrets	267
Running Apps Locally	270
Configuring the Runtime	270
Setting Environment Variables	275
Environment Variables	276
Building and Running your App Locally	280
Building the App Locally	281
Running the App Locally	282


Exporting App JSON from an Executable	283
Building Flogo App Executable and Docker Image Using Flogo - App Build CLI	284
Editing a Runtime	284
Deleting a Runtime	285
Generating an Application Docker Image Locally	285
Deployment and Configuration	287
Deploying Flogo Apps from Visual Studio Code	287
View Flogo App List in Runtime Explorer	287
Developing for Lambda	288
Creating a Connection with the AWS Connector	288
Creating a Flow with Receive Lambda Invocation Trigger	291
Receive Lambda Invocation	294
InvokeLambdaFunction	296
TIBCO Documentation and Support Services	298
Legal and Third-Party Notices	300

App Development

Flogo Extension for Visual Studio Code offers a wizard-driven approach to app development. You can create apps using only a browser. It is powered by Project Flogo®, a lightweight integration engine.

The following features from TIBCO Cloud Integration Flogo Web UI are not supported.

- Creating from trigger
- Creating an app from API spec or GraphQL Schema

 **Note:** If the Flogo® UI is not responding, then close the application tab and reopen it from the Explorer panel instead of reloading the Webview from the Visual Studio Code command palette.

For more information about Project Flogo®, see <http://www.flogo.io/>.

Creating and Managing a Flogo App

This section describes how to create and manage Flogo® apps.

Creating a Flogo App

Creating a Flogo app only creates a .flogo file. You can create a Flogo app from the **Explorer view** using Ctrl+Shift+E.

Before you begin

Ensure you meet the following prerequisites:

- Create a workspace in Visual Studio Code.
- Enable the **Auto-save** option from the **File Menu** in Visual Studio Code.

i Note: Use the path of a local directory instead of a shared location path. Entering a shared location might not work.

Procedure

1. Open the **Explorer view** (Ctrl+Shift+E) from the **Activity Bar** of Visual Studio Code.
2. Navigate to the workspace where you want to create a Flogo app and click the dropdown icon next to that workspace folder.
3. Click **Create a New Flogo App**, using the Flogo icon available right next to the workspace folder.
4. Add the name of your app in the top-most text bar that pops up.

i Note: The app name must not contain any spaces. It must start with a letter or a digit. The app name can contain letters, digits, periods, dashes, and underscores. When deploying an app on TIBCO Control Center, you can use letters and alphanumeric characters. Hyphens (-) and underscores (_) can be added in the middle of the app name.

5. Press enter to create your Flogo app.

Generating a POM File for Flogo App

TIBCO Flogo® Enterprise supports Maven as a tool for the CI-CD workflow. To get started with a Maven build, TIBCO Flogo® Extension for Visual Studio Code provides a context menu for generating the pom.xml file. To generate a pom.xml file, perform the following steps:

Procedure

1. Right-click the Flogo app.
2. Select the **Generate POM file for Flogo App** option.

For more details on pom.xml files, related configurations, how to use the Maven build for a Flogo app, running unit tests, and generating the test reports, see <https://github.com/TIBCOSoftware/flogo-maven-plugin>.

Result

The app is created and the App Details page is displayed for the new app. You can see the following tabs:





- **FLOWS:** A flow with the same name as the app is automatically created. You can now create one or more flows for the app. For more information, see [Creating Flows and Triggers](#).
- **APP PROPERTIES:** You can modify the properties of the app. For more information, see [App Properties](#).
- **CONNECTIONS:** You can view and edit the connections. For more information, see [Creating Connections](#).
- **SCHEMAS:** You can add, delete, or modify schemas. For more information, see [App Schemas](#).
- **SPECS:** You can import specs locally or from runtime. For more information, see [App Specs](#).

i Note: If you copy a Flogo app that has a password property from one workspace to another and then open that app in the Flogo editor view, the password type values persist. As a workaround, remove the password type values manually before sharing the app.

Validating Your App Flow

After creating the flows in your app, you must validate them before you finish building the app.

To validate your flow, click **Validate** on the flow details page. This validates each flow and activity. Based on the validation output, the following error or warning icons are displayed:

- If an activity has an error or a warning, it displays the  or  icon on its lower-right corner.
- If a flow has an error or a warning, it displays the  or  icon next to the flow name.

Important Considerations

- After a flow's validation is completed, the validation details are cached and remains present until you move out of the flow to the Flow List page or you refresh the page.
- If you add or change triggers and activities in a flow or any change in the canvas, no validation is triggered. To observe the latest validation, click **Validate**.
- If a subflow is appended, validation is triggered when the subflow is clicked. However, validation is not triggered if you call a subflow already present on the Flows tab.

For more information, see [Viewing Errors and Warnings](#).

Editing an App

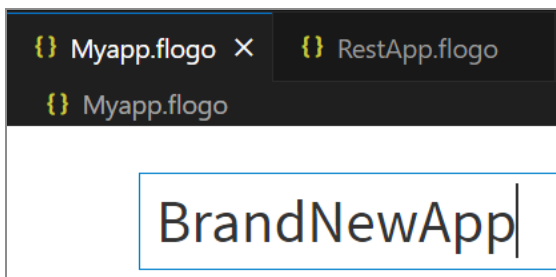
You can edit your app from the **File Explorer** tab of Visual Studio Code. Click any app to edit flows, triggers, and so on.

Renaming an App

To rename an existing app:

Procedure

1. Click any app in the **File Explorer** tab of Visual Studio Code.
2. Click inside the app name field to edit the name.



i Note: When deploying an app on TIBCO Control Center, you can use letters and alphanumeric characters. Hyphens (-) and underscores (_) can be added in the middle of the app name.

3. Click outside the app name field to save your changes.

Editing the Version of an App

When you create an app, the default version of the app is 1.0.0. You can edit the version of an app.

The format of a valid app version is:

```
xxx.xxx.xxx
```

i Note: Alphabets or special characters are not allowed in an app version.

Some examples of valid app versions are:

```
1.1.1  
11.22.13  
111.222.333
```

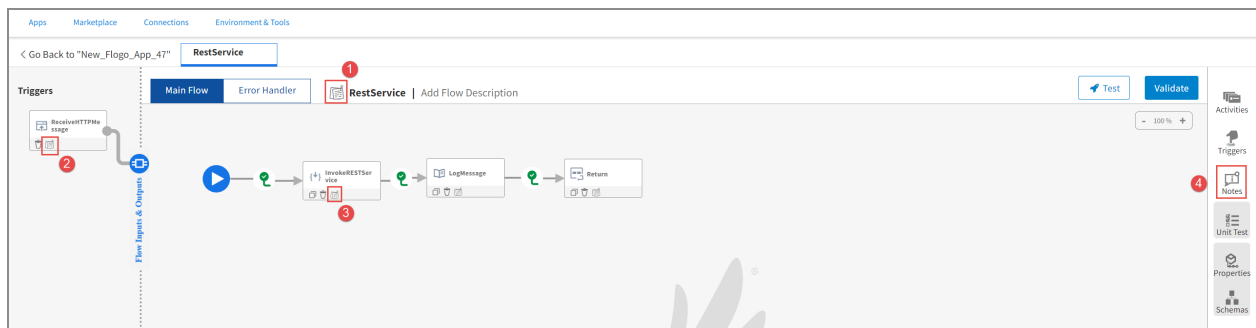
Procedure

1. Open the app details page.
Besides the name of the app, the version of the app is displayed as follows:
`New_Flogo_App_<sequential_number> v: 1.0.0`
For a newly created app, the version is 1.0.0.
2. To edit the version of the app, click the version number and edit the new version.
The new version of the app is reflected everywhere. For example, in runtime logs.

Using Notes

You can use Notes to keep a track of information about any flow, activity, or trigger. It helps you in keeping updates and important references, especially when the flow is very large and complex. In addition, this feature is available in the Error Handler tab.

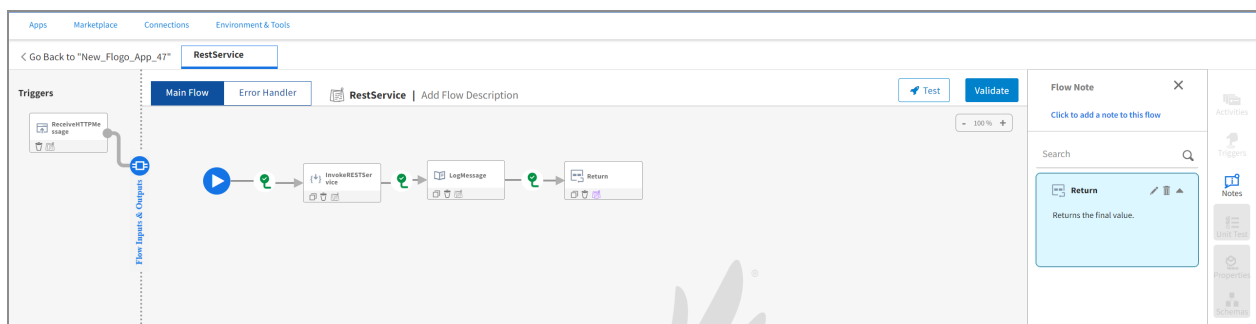
Let us take an example of a Flogo app that invokes a REST service. You use the **ReceiveHTTPMessage** Trigger, and **InvokeRESTService**, **LogMessage**, and **Return** Activities to create the app.







Here, you can add notes in the following manner:

- **Flow Note:** This note gives information about the flow. In the above case, the Flow Note can be - "This app invokes a Rest service and generates a log message that shows the status of the invoked Rest service".
- **Trigger Note:** This note is used to add information about the trigger. For the above example, add the Trigger Note that says - "This trigger listens to incoming REST requests".
- **Activity Note:** This note displays information about the activity. For the above example, the note for InvokeRESTService Activity can be - "This Activity invokes an external service".

To view all the notes, click the  icon on the right-hand sidebar.



i Note:

- Use the  icon on the Activities and Triggers to add notes.
- Use the  icon next to the Error Handler tab to add a Flow Note.
- In all cases,  icon implies notes are not added and  icon indicates that notes are added.
- The Save option on any note is enabled only after some content is added in it.
- If you are a read-only user, you cannot add, delete, or edit a note.

Deleting an App

You can delete an app by removing its .flogo file from the workspace or alternatively, you can delete the app from the **File Explorer**.

To delete an app from the **File Explorer**:

Procedure

1. Right-click the app to be deleted from the **File Explorer** tab.
2. Select **Delete** from the options.
3. On the confirmation dialog, click **Move to Recycle Bin**.

Result

The selected app is deleted.

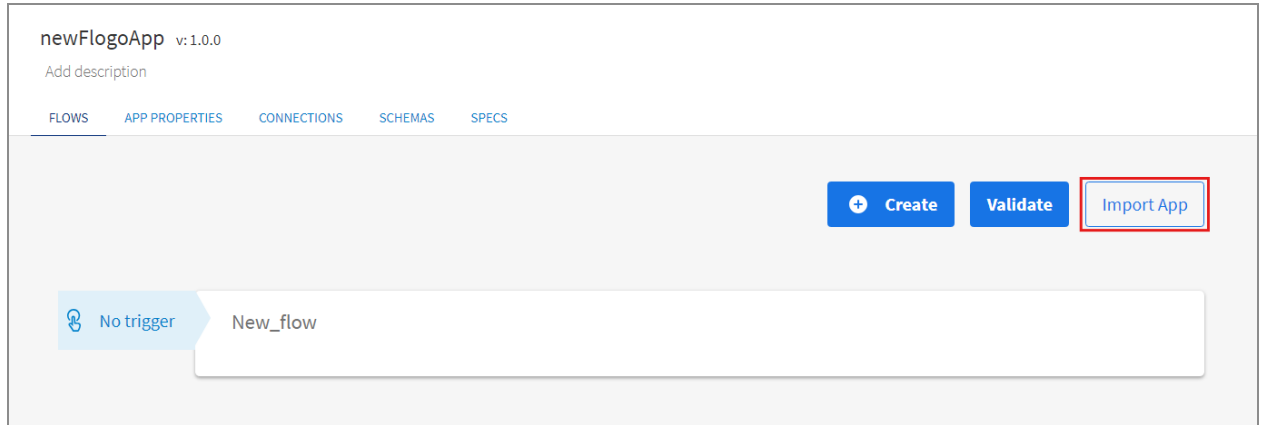
Importing an App into an Existing App

You can import a .json or .flogo file of an app to reuse its flows, triggers, connections, schema, and specs. You can import the file to a new app without flows or an existing app that contains flows.

To import an app, perform the following steps:

Procedure

1. Open an existing app.
2. Click the **Import App** button.



3. In the **Import App** dialog, upload the .json or .flogo file of the app that you want to import, then click **Upload**.
4. Select the entities that you want to import from the source app:
 - a. Select or clear the checkboxes for specific flows, triggers, schema, or spec.

i Note: By default, all flows, triggers, connections, schema, and specs are selected for importing.

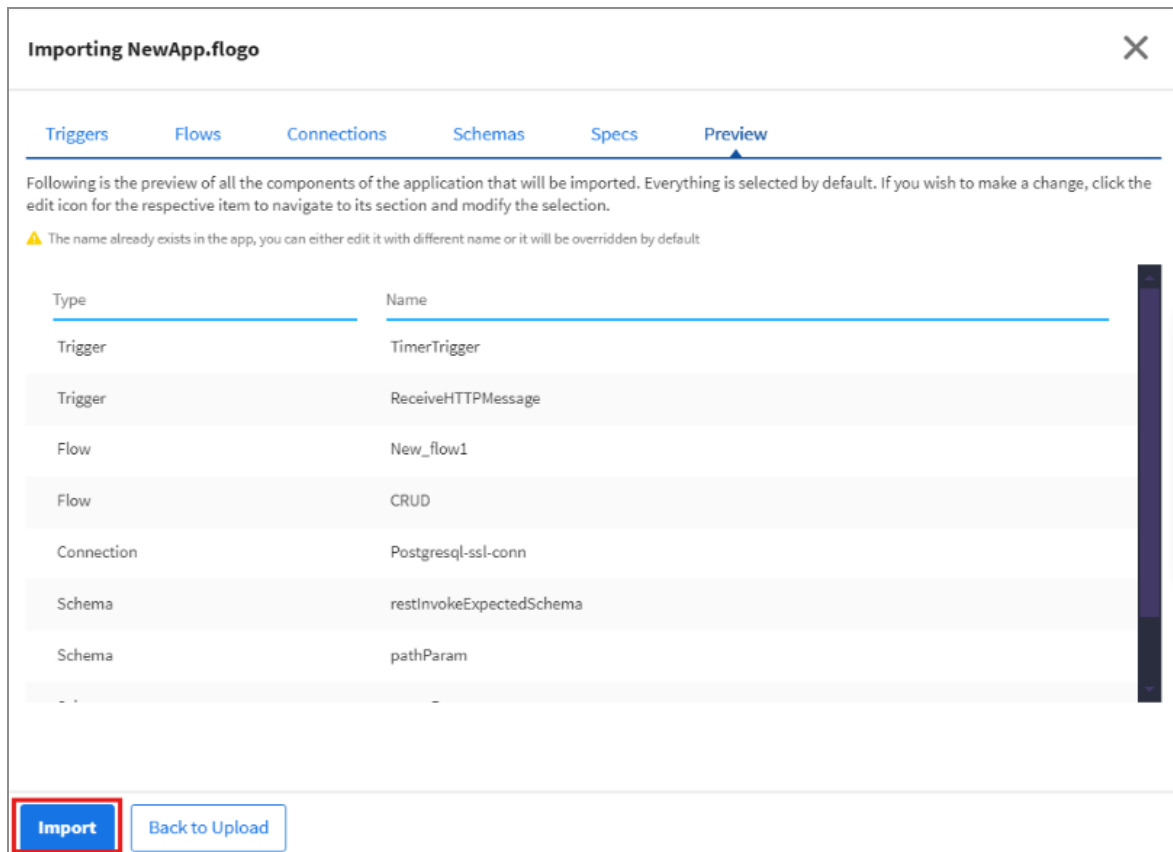
- b. If you select a trigger, all flows, schema, and specs associated with that trigger are selected by default.
- c. To import a flow without importing the attached trigger, select the flow only and not the attached trigger.

i Note: In case of same names, existing application properties are overwritten by the imported application properties.

- d. If you want to remove the flows, schema, and specs associated with the trigger, you must clear the trigger checkbox.
- e. If a trigger, flow, schema, or spec is already a part of the existing application,

you see a warning icon. Rename the trigger, flow, schema, or spec.

5. Confirm your selections and click **Import**.



Additional Information

- After uploading a file, you can see all the connections used in the application and the connections that are available but unused.
- The imported application reuses an existing connection if one of the same type is already available. If you must change a connection, you can either update it manually or create a new one.
- If you import a new application with the same name as an existing one, the existing application is overwritten.
- When importing the app, if you rename the trigger within the same flow, a new flow is created with the renamed trigger. Both the original and the renamed trigger are added to the new flow.

- When importing the app, if you rename both the trigger and the flow, a new flow is created with the renamed trigger and flow.

Creating Flows and Triggers

An app can have one or more flows and a flow could be attached to one or more triggers. Similarly, a trigger can have multiple flows attached to it.

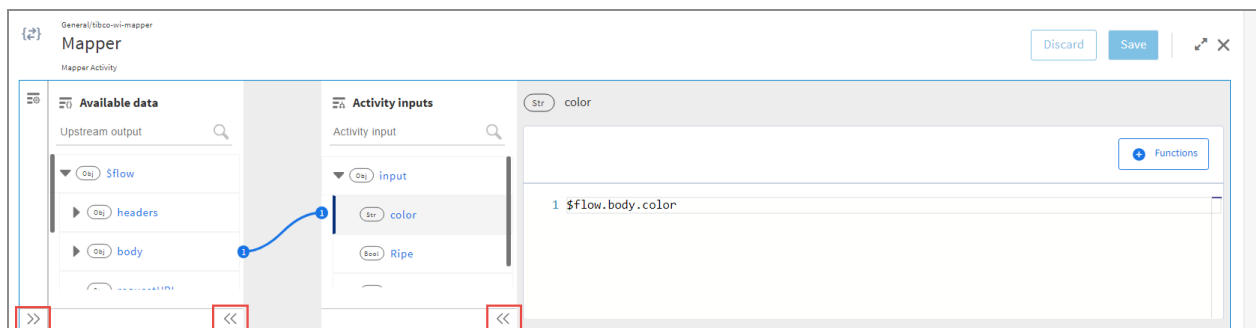
Flows

Each flow represents a specific business logic in an app. A flow contains one or more activities. The flow execution is started by a trigger. A new flow can be created only from the app details page. A flow can be attached to one or more triggers. You can attach the flow to a trigger at the time of flow creation by selecting the **Start with a trigger** option or selecting one of the options under **Start with** in the **Add triggers and flows** dialog. Optionally, you can create a blank flow without a trigger by selecting **Configure flow inputs and outputs** to begin with, then attach it to one or more triggers at any time after the flow has been created.

Activities and Triggers Panel

You can add a trigger or an activity in a flow from **Activities** or **Triggers** palettes available in this panel.

After adding a trigger or activity, you can configure it. The configuration panel for activities, triggers, and branches is resizable. You can expand, collapse, or resize the various columns in this panel. You can also resize the configuration panel in the mocking and assertion of activities or flow inputs and flow outputs in unit testing.



Flows

This section contains information about creating and managing flows in your app.

Creating a Flow

Every app consists of at least one flow. Each flow can be attached to one or more triggers. You have the option to first create a blank flow (a flow without a trigger) and then attach the flow to one or more triggers. On the **App Details** page, click **Create** to create the first flow in an app.

Before creating a flow that uses connectors, ensure that you create the required connections. For more information, see [created the necessary connections](#).



Warning: In an app with multiple triggers, the port number must be unique for all the triggers that require a port number. For example, REST and/or GraphQL triggers. Two triggers in the same app cannot run on the same port.

For flows that are attached to multiple triggers, you cannot disable a trigger. Specify a particular trigger to run. Or, specify the order in which the triggers run. When a flow runs, all triggers get initialized in the order that they appear within the flow.

The output of a trigger provides the input to the flow. Hence, it must be mapped to the flow input. When creating a flow without a trigger, there must be a well-defined contract within the flow that specifies the input to the flow and the output expected after the flow completes execution. You define this contract in the **Flow Inputs & Outputs** dialog. The **Flow Inputs & Outputs** contract works as a bridge between the flow and the trigger, hence every trigger has to be configured to map its output to the **Input** parameters defined in **Flow Inputs & Outputs**. You do this on the **Map to Flow Inputs** tab of the trigger.

Likewise, for triggers (such as the **ReceiveHTTPMessage** REST trigger) that send back a reply to the caller, the trigger reply must be mapped to the flow outputs (parameters configured on the **Output** tab of the **Flow Inputs & Outputs** tab). You do this mapping on the **Map from Flow Outputs** tab of the trigger.

A **Return** Activity is not added by default. Depending on your requirements, you must add and configure the **Return** Activity manually. For example, if any trigger needs to send a response back to a server, its output must be mapped to the output of the **Return** Activity in the flow.


The **Map Outputs** tab of the **Return** Activity displays the flow output schema that you configured on the **Output** tab of the **Flow Inputs & Outputs** tab. The **Map from Flow Output** tab of the trigger constitutes the trigger reply. This tab also displays the flow output schema that you configured on the **Output** tab of the **Flow Inputs & Outputs** tab.

Perform the following steps when using a **ReceiveHTTPMessage** REST trigger:

- Add a **Return** Activity at the end of the flow.
- On the **Map Outputs** tab of the **Return** Activity, map the elements in the schema to the data coming from the upstream activities.
- On the **Map from Flow Output** tab of the trigger, map the trigger reply elements to the flow output elements.

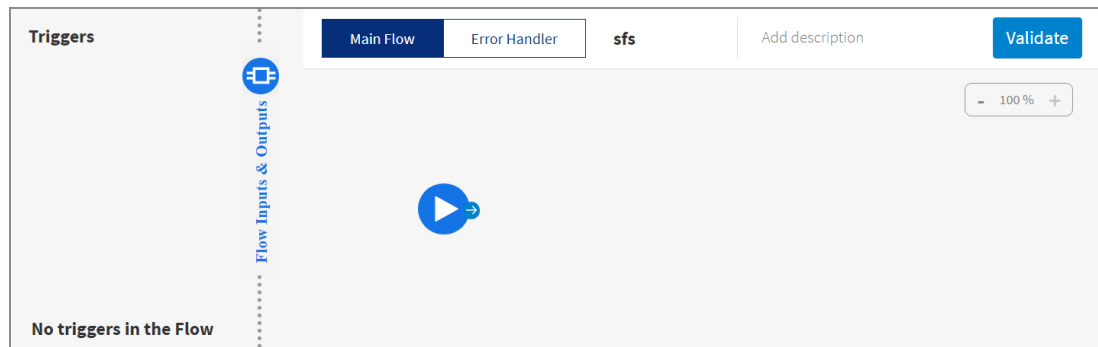
Follow these steps to create a flow:

1. Click the app in the **File Explorer** tab of Visual Studio Code to open app details page.
2. Under the **Flows** page, click **Create**. This opens the **Add triggers and flows** dialog.
3. Enter a name for the flow in the **Name** text box. Flow names within an app must be unique. An app cannot contain two flows with the same name.
4. Optionally, enter a brief description of what the flow does in the **Description** text box. The **Flow** option is selected by default. To create a flow from a specification, select the specification under **Start with** and refer to the appropriate section under [Building APIs](#).
5. Click **Create**. The **Flow** is created.

 **Note:** When you click a flow name under **Flows** in the Flogo App panel, it opens the flow. The current open flow also is reflected in the Flogo App panel.

6. After the **Flow** has been created, you can start with either of the following actions:
 - Start with a trigger - If you know the trigger with which you want to activate the flow, select this option. Select a trigger from the **Triggers** palette. For more details on the type of trigger that you want to create, see the relevant section in the [Starting with a Trigger](#) topic. If there are existing flows attached to triggers, you are prompted to either use an existing trigger or use a new trigger that has not been used in an existing flow within the app.
 - Configure flow inputs and outputs - Select this option if you know the

algorithm for the flow, but do not yet know the circumstances that cause the flow to run. It creates a blank flow that is not attached to any trigger. Flow inputs and outputs create a contract between the trigger and the flow. When you create a trigger, you must map the output of the trigger to the input of the flow. This contract serves as a bridge between the trigger and the flow. You have the option to attach your flow to one or more triggers at any later time after the flow has been created.



If you select **Start with a trigger**, the flow is attached to the trigger you selected. If you select **Configure flow inputs and outputs**, a blank flow without a trigger gets created.

Note: **StartActivity** is a special activity that is always added to the newly created flows.

Creating a Flow Starting with a Trigger

When creating a flow, if you know the circumstances in which you want the flow to activate, select the **Start with a trigger** option and select an available trigger that activates the flow.

Warning: If an app has multiple triggers that require a port to be specified, make sure that the port number is unique for each trigger. For example, REST or GraphQL trigger. Two triggers in the same app cannot run on the same port.

If you are unsure of the circumstances under which the flow should be activated, or if you want the flow to be activated under more than one situation, use the **Configure flow inputs and outputs** option and attach the flow to one or more triggers later as needed. For more information, see [Creating a Flow without a Trigger](#).

Creating a Flow Attached to a REST Trigger

When creating a flow with a REST (ReceiveHTTPMessage) trigger, you can enter the schema in the **Configure trigger** dialog during trigger creation.

Note: An app cannot have two REST flows implementing the same operation on the same resource. For example, you cannot create two REST flows implementing a GET operation with /data as the resource path. So, the combination of the operation and its resource path must be unique within an app.

Note: If you want to have two flows using the same operation on the same resource, while creating the flows on the **Settings** tab of the **ReceiveHTTPMessage** trigger, ensure that you configure different ports for each flow.

You can create a REST flow by entering a JSON schema.

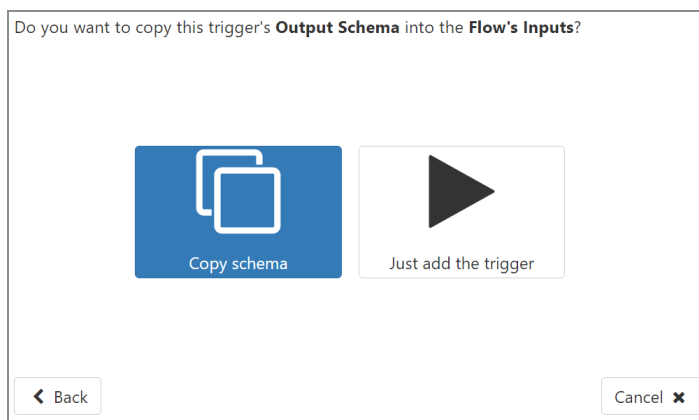
Warning: If you modify the **Reply Settings** tab of a **ReceiveHTTPMessage** trigger, the corresponding **ConfigureHTTPResponse** activities within that flow do not change appropriately. This happens when you remove fields from the **Reply Settings** tab. Redo the mappings for the **ConfigureHTTPResponse** Activity.

To create a REST flow by entering the schema:

1. Click the app in the **File Explorer** tab of Visual Studio Code to open app details page.
2. Click **Create** to create an empty flow.
3. Edit the name for the flow. Flow names within an app must be unique.
4. Optionally, enter a brief description of the flow in the **Add description** text box.
5. Now, click the **Triggers** palette. The triggers palette opens with all the available triggers listed.
6. Drag **Receive HTTP Message** to the **Triggers** area on the left. The trigger configuration dialog opens.
7. Select the REST operation under the **Method** that you want to implement by clicking it.

Note: Two REST triggers cannot have an identical port, path, and method combination. Each REST trigger needs to differ from the other for the same flow with either a unique port, path, or operation.

8. Enter a resource path in the **Resource Path** text box.
9. Enter the JSON schema or JSON sample data for the operation in the **Response Schema** text box. This is the schema for both input and output.
10. Click **Continue**.
11. Select one of the following options:



If you select **Copy Schema**, the schema that you entered in [step 10](#) automatically gets copied or displayed in a tree format to the following locations when the trigger gets added:

- Trigger output, on the **Map to Flow Inputs** tab of the trigger.
- Flow input, on the **Input Settings** tab of the **Flow Inputs & Outputs** tab.
- Trigger reply (if the trigger has a reply), in **Reply Settings** of the trigger.

For details on configuration parameters, see the [REST Trigger](#) section.

If you select **Just add the trigger**, a REST trigger is added to the flow without any configuration. You can configure this REST trigger later by clicking the trigger from the app details page. Any changes made to the trigger must be saved by clicking **Save**.

The flow page opens.

12. Map the trigger output to the flow input.
 - a. Open the trigger configuration dialog by clicking the trigger.
 - b. Open the **Map to Flow Inputs** tab.
 - c. Map the elements under **Flow inputs** to their corresponding elements under **Available data** one at a time.
13. Map the flow output to the trigger reply as follows:
 - a. In the trigger configuration dialog, click the **Map from Flow Outputs** tab.
 - b. Map the elements under **Trigger reply** to their corresponding elements under **Available data**.
 - c. Close the dialog.
14. Click **Save**.

Creating a Flow Attached to Other Triggers

This section applies to triggers that are not REST, or GraphQL triggers.

To create a flow with such a trigger:

1. Click the app in the **File Explorer** tab of Visual Studio Code to open app details page.
2. Click **Create** to create an empty flow.
3. Edit the name for the flow. Flow names within an app must be unique.
4. Optionally, enter a brief description of the flow in the **Add description** text box.
5. From the **Triggers** palette, select the desired trigger and drag it to the trigger's area.
6. Click the trigger to display its properties.
7. Configure the properties for the trigger.

Creating a Blank Flow (Flow without a Trigger)

You can create a flow in Flogo app without attaching it to a trigger. This method of creating a blank flow is useful when the logic for the flow is available, but you do not know the condition under which the flow should activate. You can start by creating a flow with the logic and attach it to one or more triggers later.

Follow these steps to create a flow without a trigger:


1. Click the app in the **File Explorer** tab of Visual Studio Code to open app details page.
2. Click **Create** to create an empty flow.
3. Edit the name for the flow. Flow names within an app must be unique.
4. Optionally, enter a brief description of the flow in the **Add description** text box.
5. Click **Flow Inputs & Outputs** to configure the inputs and/or outputs to the flow on the **Input** or **Output** tab respectively. For more information, see [Flow Inputs & Outputs Tab](#).

Mapping trigger outputs to flow inputs and flow outputs to trigger reply creates a contract between the trigger and the flow. Hence, when you attach the flow to a trigger, you must map the output of the trigger to the flow input. You have the option to attach your flow to one or more triggers later after the flow has been created. For more details, see [Attaching a Flow to One or More Triggers](#).

6. Enter a JSON schema containing the input fields to the flow on the **Input Settings** tab and click **Save**.
7. Enter the JSON schema containing the flow output fields on the **Output Settings** tab and click **Save**.
8. When you are ready to add a trigger, refer to [Adding Triggers to a Flow](#) to add one or more triggers to the flow. For triggers that need to send back a response to the server, you must map the flow output to the reply of the trigger.

Flow Input and Output Tab

Use these tabs to configure the input to the flow and the flow output. These tabs are particularly useful when you create blank flows that are not attached to any triggers.

 **Note:** The schemas for input and output to a flow can be entered or modified only on this **Flow Inputs & Outputs** tab. You cannot coerce the flow input or output from outside this tab.

Both the tabs (**Input** tab and **Output** tab) have two views:

- **JSON schema view:**

You can enter either the JSON data or the JSON schema in this view. Click **Save** to

save your changes or **Discard** to revert the changes. If you have entered JSON data, the data is converted to a JSON schema automatically when you click **Save**.

- **List view:**

This view allows you to view the data that you entered in the JSON schema view in a list format. In this view, you can:

- Enter your data directly by adding parameters one at a time.
- Mark parameters as required by selecting its checkbox.
- When creating a parameter, if you select its data type like an array or an object, an ellipsis (...) appears to the right of the data type. Click the ellipsis to provide a schema for the object or array.
- Click **Save** to save the changes or **Discard** to discard your changes.

Attaching a Flow to One or More Triggers

If you have created a blank flow without attaching it to a trigger, you can now attach it to an existing trigger that is being used by another flow in the same app.

A flow that is created without being attached to a trigger has its input and output parameters defined on the **Flow Inputs & Outputs** tab. You can access it by clicking the blue bar with the same label. The output from the trigger is the input to the flow. So, you must map the input parameters defined on the **Input** tab of this dialog to the trigger output parameters. This mapping must be done in the trigger. The mapping creates a contract between the trigger and the flow and is mandatory for the flow and the trigger to interact with each other.

You can use one of these methods to attach a flow to a trigger:

1. From the flow details page:
 - a. Open the flow details page by clicking the flow name on the app details page.
 - b. From the **Triggers** palette, drag a desired trigger to the trigger's area.

For REST, you are prompted to enter additional handler setting details.

Click the trigger icon to configure the trigger as needed. For REST triggers, be sure to map the trigger outputs to flow inputs and the flow outputs to the trigger reply.

Catching Errors

You can configure a flow to catch errors at two levels:

- At the flow level, by configuring the **Error Handler** in the flow. For more information on configuring the error handler in the flow, see [Creating an Error Handler Flow](#).
- At the Activity level, by creating an error branch from an Activity. For details on how to create an error branch from an **Activity**, see [Types of Branch Conditions](#).

Creating an Error Handler Flow

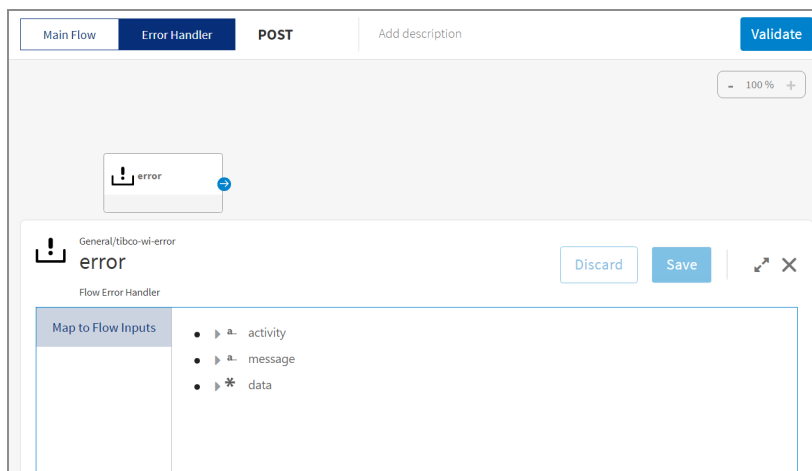
Use **Error Handler** to catch exceptions that occur while running a flow. The error handler is designed to catch exceptions in the activities within a flow. If there are multiple flows in an app, the error handler must be configured for each flow separately. Branching is supported for error handler flows, similar to the other flows.

To configure the error handler:

1. Click an existing activity in a flow.
2. Click the **Error handler** tab.

The error handler opens with the **error** Activity displayed.

Clicking the **error** activity exposes the fields that you can configure for an error that is generated by the activity.



The **Map to Flow Inputs** tab of the **error** Activity has three elements; **Activity**, **message**, and **data**. The **activity** element is used to output the name of the activity

that is generating the error, the **message** element is used to output the error message string, and the **data** element can be configured to output any data related to the error. The **message** element on the **Input** tab of any activity in the **Error Handler** flow can be configured to output one or all of these three elements.

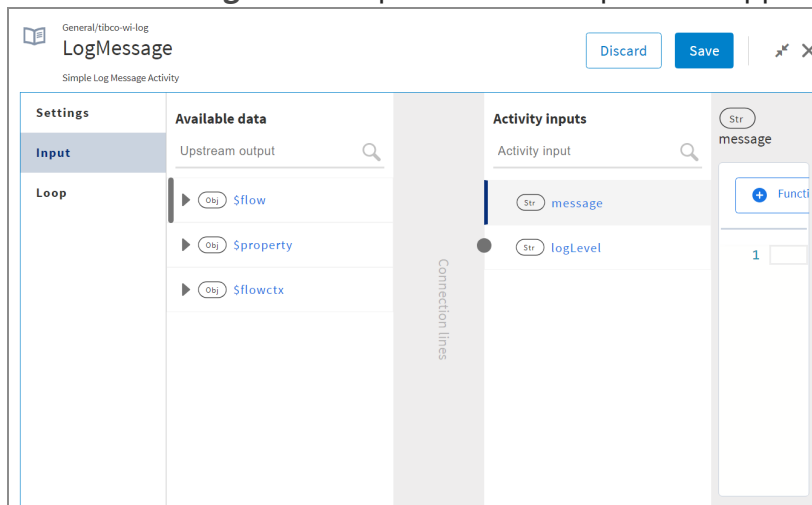
3. From the **Activities** palette, add an activity for which you want to configure the error message. Add a branch to connect the **error** with the activity that you have added.

The **Input** tab of that Activity displays a **message** in its input schema. This is a required element that you must map.



Note: A **Return** Activity is not added by default. Depending on your requirements, you must add the **Return** Activity manually.

4. Click the **message** in the input schema to open its mapper.



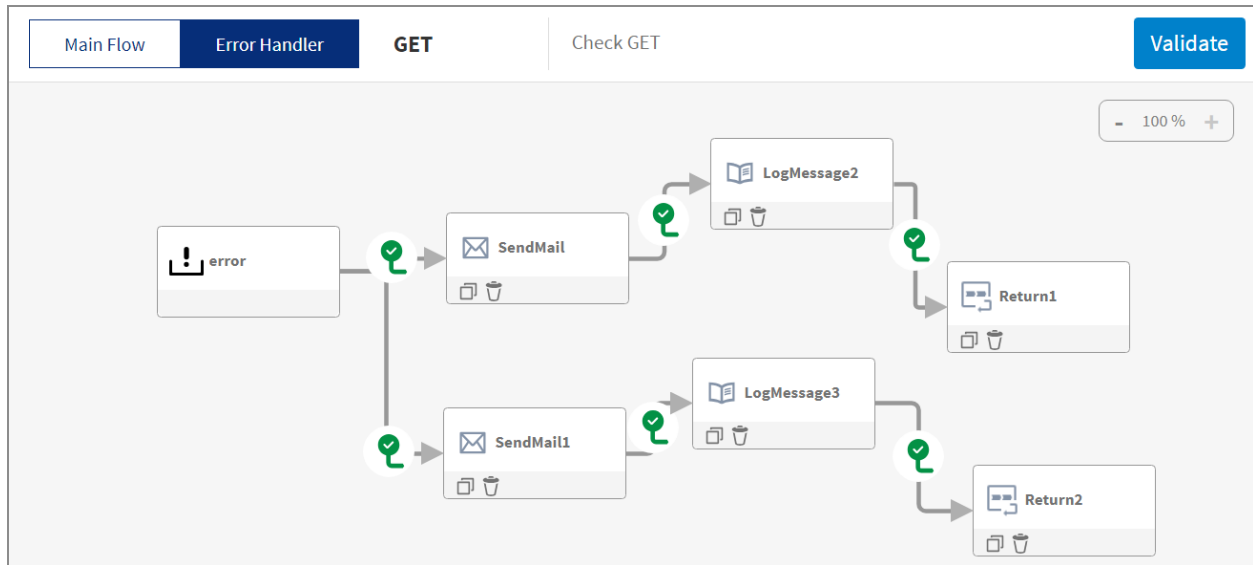
5. Expand **\$error** to expose the **Activity**, **message**, and **data** elements that you can configure for the error message.

To map the **message** element under **Activity inputs**, you can either manually type in the error string enclosed in double-quotes or use the **concat** function under **string** in the mapper to output the Activity name along with a message. For more information, see [Using Functions](#).

6. Continue configuring the error message for each activity in the flow.

If there is an error for the activity in any flow of the app, it shows as output in the log for the app when the app is pushed to the cloud.

Here is an example of how an error handler flow looks after it is configured:



Viewing Errors and Warnings

Flogo uses distinct icons to display errors and warnings within an app.

The following icons are used:

- ✖ - error icon. Resolve the errors before building the app. Errors should not be ignored.
- ⚠ - warning icon. Warnings are generated to alert you of something that might need to change in the entity where the warning icon is displayed. You have the option to ignore the warning and move on.
- 🔧 - missing extension icon. Check and correct missing extensions before building an app. Missing extensions must be fixed before proceeding.

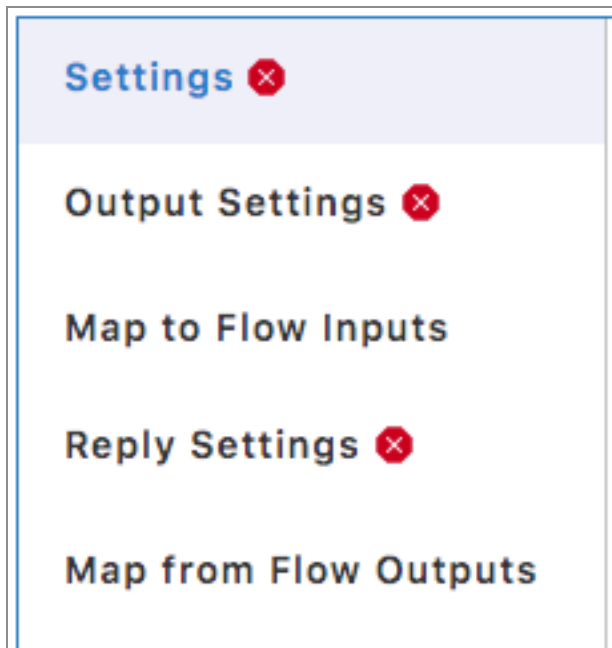
Here is the hierarchy of errors and warnings reported in Flogo:

App level reporting - When you open an app, the app details page displays the list of flows. If there are errors or warnings in a flow, appropriate icons are displayed next to the flow name along with a number, where the number indicates an aggregate number of errors or warnings in the flow. If there are no errors or warnings, these icons are not displayed.

Activity and Trigger level reporting - when you click a flow name, the flow details page opens displaying the implementation of the flow. This page displays errors if any at the

activity level. For instance, a **LogMessage** activity may display an error symbol within the activity configuration. Resolve the error before proceeding.

Activity and Trigger configuration tab level reporting - when you click an activity or a trigger in the flow, its configuration page opens, displaying the various tabs. Click a tab to see the errors or warnings in the configuration within that tab.



Using Subflows


Flogo provides the ability to call any flow from another flow in the same app. The flow being called becomes the subflow of the caller flow. This helps in separating the common app logic by extracting reusable components in the app and creating standalone flows for them within the app. Any flow in the app can become a subflow for another flow within the same app. Also, there are no restrictions on how many subflows a flow can have or how many times the same subflow can be called or iterated in another flow. Hence, subflows are useful when you want to iterate a piece of app logic more than once or have the same piece of logic repeated in multiple locations within the app.

Here are a few points to keep in mind when creating and using subflows:

- The subflow and its calling flow must both reside within the same app. You cannot call a flow from another app as a subflow in your app.
- You can now open a subflow from the main flow. In addition, you can navigate to the

main flow from an open subflow.

- Since you can call any flow from any other flow within the app, you must be careful not to create cyclical dependency where a flow calls a subflow and the subflow, in turn, calls its calling flow. This results in an infinite calling cycle and the "Cyclic dependency detected in the subflow" error is displayed.
- You can configure the iteration details on the **Loop** tab of the **Start a SubFlow** Activity. The **Start a SubFlow** Activity iterates multiple times, resulting in the subflow being called multiple times.


 **Important:** You can delete any flow in an app even though the flow might be in use as a subflow within another flow. You do not receive any error messages at the time of deletion, but when you run the app, its execution fails with an error.

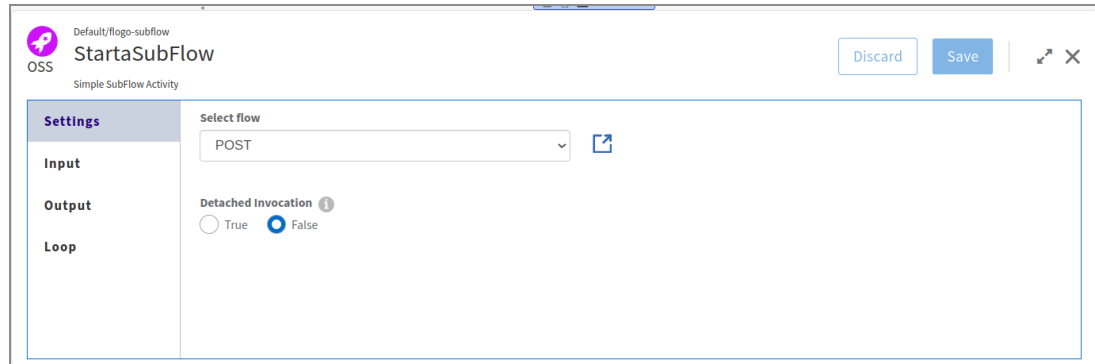
Creating Subflows

You can create a subflow exactly like you would create any other blank flow.



To create a subflow:

1. Identify the piece of logic in your app that you want to reuse elsewhere in the app or iterate multiple times.
2. Create a flow without a trigger for that logic. For more information on how to create such a flow, see [Creating a flow without a trigger](#).
3. To use this flow as a subflow within another flow, you must add a **StartaSubFlow** Activity at the location in the calling flow from where you want to call the subflow. For example, if you want to call a subflow after the third Activity in your calling flow, insert a **StartaSubFlow** Activity as the fourth Activity in the calling flow. To do so:
 - a. Open calling flow.
 - b. On the flow details page, click the **Activities** palette.
 - c. Under the **Default** category, select the **StartaSubFlow** activity and drag it to the activities area.
 - d. Add the branches to connect the **SubFlow** activity with the activity that you want to call a subflow from and to the activity where the subflow must end. Also set the branch conditions for each connection line wherever required.

- e. Click the **StartaSubFlow** activity to open the configuration dialog. To call the required subflow, select the subflow from the **Select flow** dropdown in the **Settings** tab and save the changes. If you want to see the flow in detail, use the **Open Subflow** option in the **Settings** tab or click the  icon on the **StartaSubflow** Activity. This appends the active flow in a breadcrumb trail and highlights it. A breadcrumb trail starts with the app name and includes the flows and subflows.



Note:

- If a subflow is already selected on the **StartaSubflow** activity, then you can directly open it in a breadcrumb trail by clicking the  icon on the activity tile.
- If the  icon is present on the **StartaSubflow** activity, it means that a subflow is selected in the **StartaSubflow** Activity.
- Several flows can open in a breadcrumb trail. If you click one flow, the child hierarchy disappears and only the parent hierarchy remains.

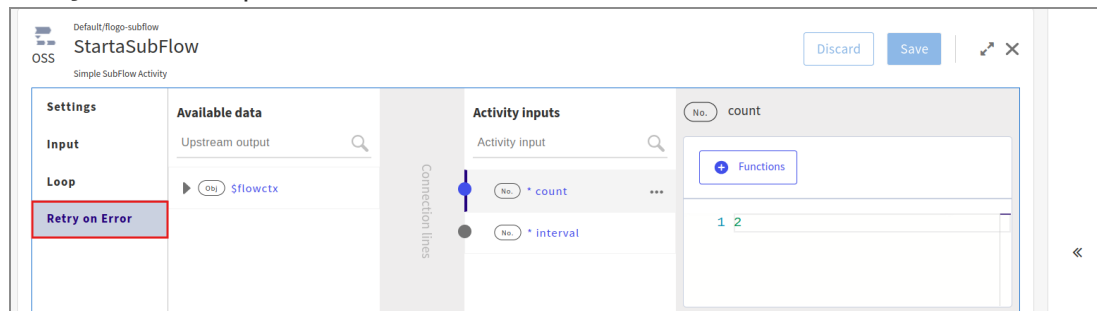
The schemas that you had configured in the **Input Settings** and **Output Settings** of the **Flow Inputs & Outputs** tab in the selected subflow appear on the **Input** and **Output** tabs of the **StartaSubFlow** Activity.

You can now configure the input and output for the subflow in the

StartaSubFlow Activity. If you add additional input and/or output parameters on the **Flow Inputs & Outputs** tab of your subflow, they become available to configure from the **Input** and/or **Output** tabs of the **StartaSubFlow** Activity. The output from the **StartaSubFlow** Activity is available for use as input in all activities that appear after it.

At app runtime, the **StartaSubFlow** Activity in the calling flow calls the selected subflow.

- f. If you want your subflow to iterate multiple times, configure the iteration details on the **Loop** tab of the **StartaSubFlow** activity. For more information on how to configure the **Loop** tab, see [Using the Loop](#).
- g. If you want to run the subflow again in case of an error, you can select the **Retry on Error** option.



- h. If you want to execute certain events in the main flow without waiting for the subflow to complete its execution, you can do this using the **Detached Invocation** toggle on the **Settings** tab of the **StartaSubFlow** activity. When you set this **Detached Invocation** toggle to true, the **Output** option is not available in the **StartaSubFlow** activity window, and without waiting for the subflow output, the main flow is executed.

Note: When a subflow activity is configured to run in detached mode, it runs as an independent flow instance and its output is not available to the calling flow. The **Retry on Error** feature is not applicable to detached mode.

Creating a Flow Execution Branch

Activities in a flow can have one or more branches. If you specify a condition for a branch, the branch runs only when the condition is met. You also have the option to create an error branch from an activity. The purpose of the error branch is to catch any errors that

might occur while running an activity. Branching is also supported for **Error Handler** flows, to catch all errors at the flow level.



Note:

- You cannot create a branch from a trigger or a **Return** Activity.
- All activities that come after a branch are run irrespective of how the branch condition is evaluated.

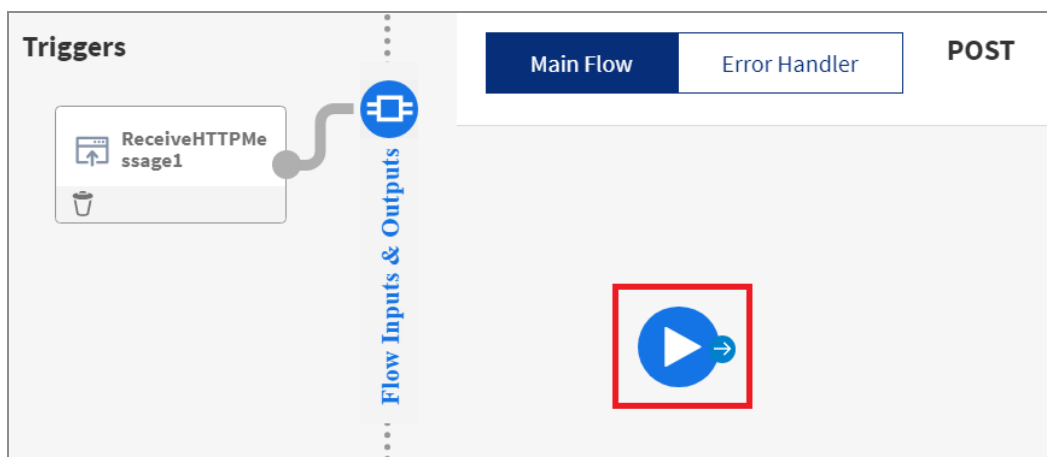
A **Return** activity ends the flow execution. Regardless of where the **Return** activity is placed in the flow, the flow execution exits the process as soon as it encounters a **Return** activity anywhere in the flow.



Note: A **Return** Activity is not added by default. Depending on your requirements, you must add the **Return** Activity manually. For example, if any trigger needs to send a response back to a server, its output must be mapped to the output of the **Return** Activity in the flow.

To create a flow execution branch:

1. Click the app in the **File Explorer** tab of Visual Studio Code to open app details page.
2. For a start branch, drag a connection line from the blue arrow on the **StartActivity** icon to the desired activity that you want to start the execution with.



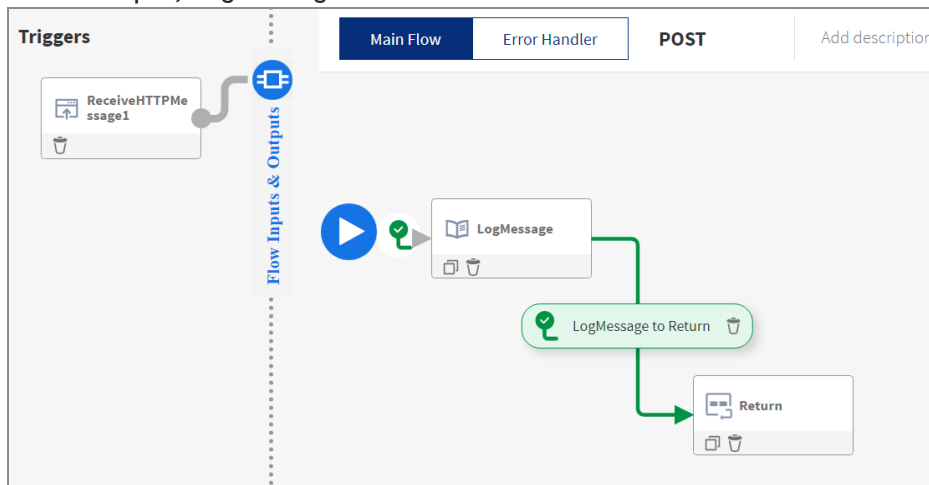
3. A branch gets created.

Each branch has a label associated with it. The label has the following format:

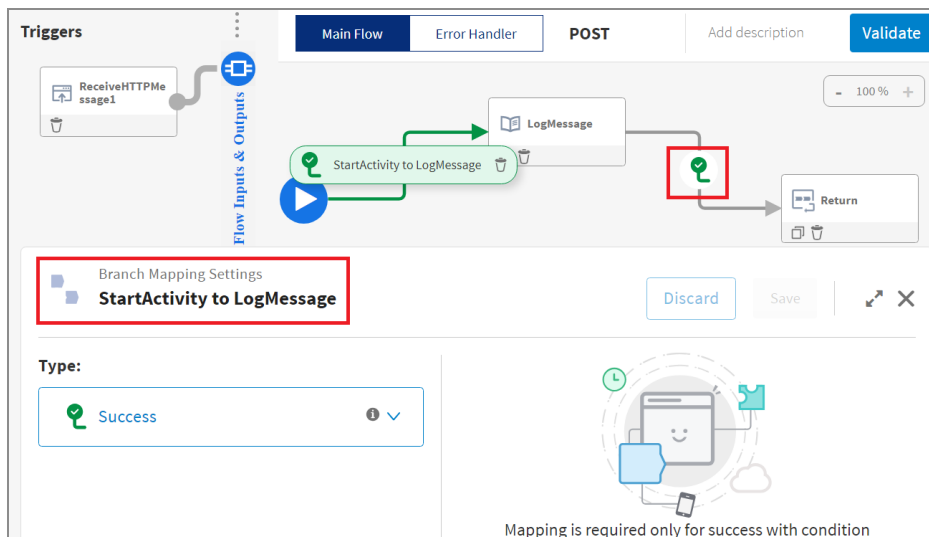
When branching to a specific activity:

<Name of activity in main flow>to<Name of activity in branch>

For example, LogMessage to Return.



4. You can add a branch between the two activities. Hover over the activity that you want to start with and drag a connection line to the activity you want to connect to.
5. Clicking the branch opens the **Branch Mapping Settings** dialog.

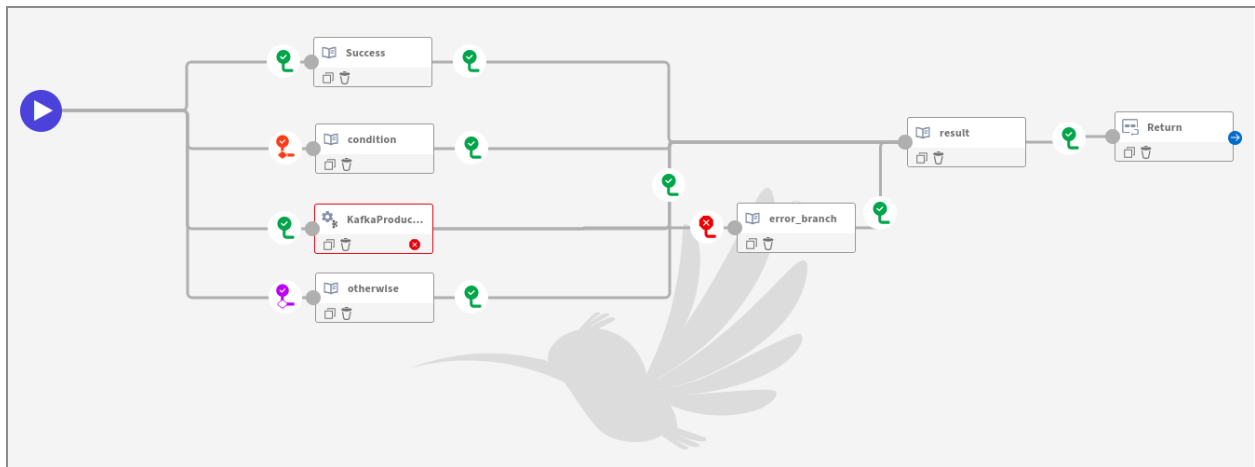


6. Select either of the branch conditions: **Success**, **Success with condition**, **Success with no matching condition**, or **Error**. For more information about the conditions, see [Types of Branch Conditions](#).
7. Click **Save**.

8. Add a condition to a branch as required. For more information, see [Setting Branch Conditions](#).
9. If you want the flow execution to end after this branch is run successfully, add and configure the **Return** activity at the end of the branch. If you do not want the flow execution to end, do not add a **Return** activity at the end of the branch.

Joining or merging branches

You can now connect multiple activities to a single activity. In this case, an activity is executed only after all connected activities are either executed or skipped due to conditional branch.



Types of Branch Conditions

TIBCO Flogo® Enterprise supports multiple types of branch conditions.

Select one of the following conditions during branch creation:

- **Success**

A success branch is run whenever an activity is run successfully. If there is an error in the activity completion, this branch does not run. The branch has no conditions set in it.

- **Success with condition**

Select this condition if you want a branch to run only when a particular condition is met. If you select this condition and do not provide the condition, the branch never

runs.

You can form an expression using anything available under upstream activity outputs and available functions, which should evaluate to a boolean result value.

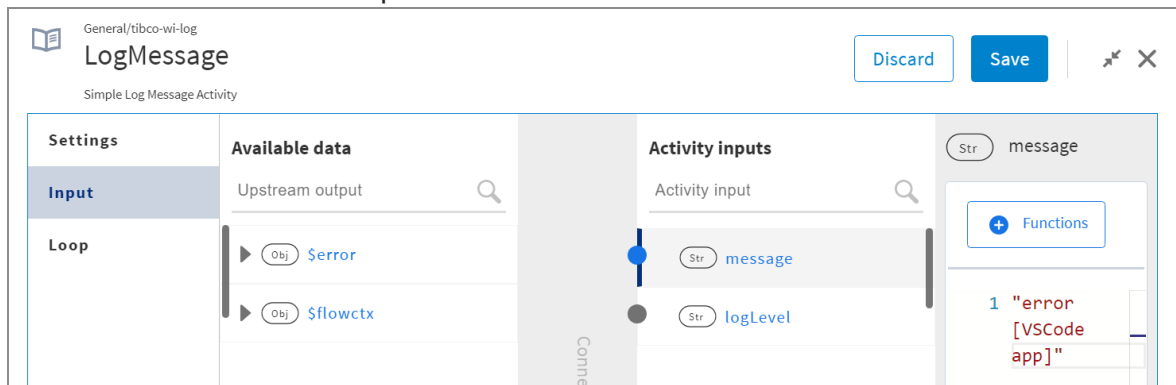
- **Success with no matching condition**

This branch condition is displayed only when you already have an existing **Success with condition** branch.

- **Error**

A branch with this condition runs if there are errors in completion of the activity. An activity can have only one **Error** branch.

Details of the error, such as the Activity and the type of the error message, are returned in `$error`. For example:



The **Error** branch flow differs from the error handler flow. In the error branch, the error branch is designed to catch exceptions at the activity level from which the error branch originates. Whereas the error handler flow is designed to catch exceptions that occur in any activity within the flow. So, if you handle the errors by creating an error branch at the activity level, the flow execution control never transfers to the error handler flow.

Order in which Branches are Run

When an Activity has multiple branches, regardless of the number of branches or the order in which the branches appear in the UI, the branch execution follows a pre-defined order.

i Note: The flow execution ends if it encounters a **Return** activity at any branch. In such situations, the activities that are placed after the return activity are not run.

The order in which the branches are run is as follows.

1. **Success with condition** branch

This branch runs only if its branch condition is met.

2. **Success with no matching condition** branch

This branch condition is displayed only when there is at least one existing **Success with condition** branch for the Activity. The **Success with no matching condition** branch is typically used when you want a specific outcome if none of the **Success with condition** branches meet their condition.

- This branch runs only if none of the **Success with condition** branches run. If the **Success with condition** branch runs and it does not have a **Return** Activity at the end of the branch, the flow execution control is passed to the success branch. If the **Success with condition** has a **Return** activity, the flow execution is ended after the **Success with condition** branch runs.
- If you delete all **Success with condition** branches without deleting the **Success with no matching condition** branch, you receive a warning informing you that the **Success with no matching condition** branch is orphaned.

3. **Success** branch

When an Activity has both **Success** and **Success with condition** branches, always the **Success with condition** branch runs first and if there are multiple success branches, the order of execution depends on the reverse order in which each branch was created, that is, the success branch that was created at last is executed first.

4. The **Error** branch is run as soon as the flow execution encounters an error.

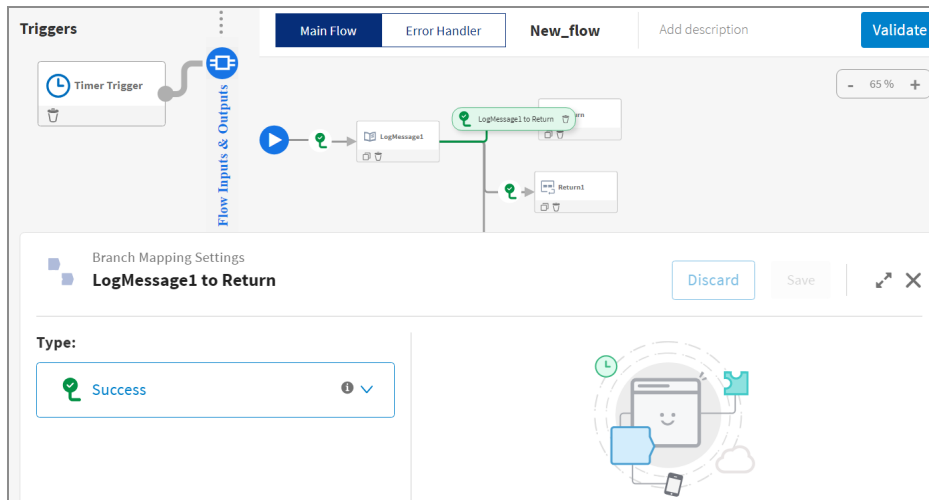
Setting Branch Conditions

You can set conditions on a branch such that only if the condition is met, the branch runs.

To set conditions on a branch:

1. Click the branch that you want to set the conditions for. The **Branch Mapping**

Settings dialog opens.



2. Select a branch condition: **Success**, **Success with condition**, or **Error**. If you already have a **Success with condition** branch present, you see **Success with no matching condition**.

For more information on the three conditions, see [Types of Branch Conditions](#).

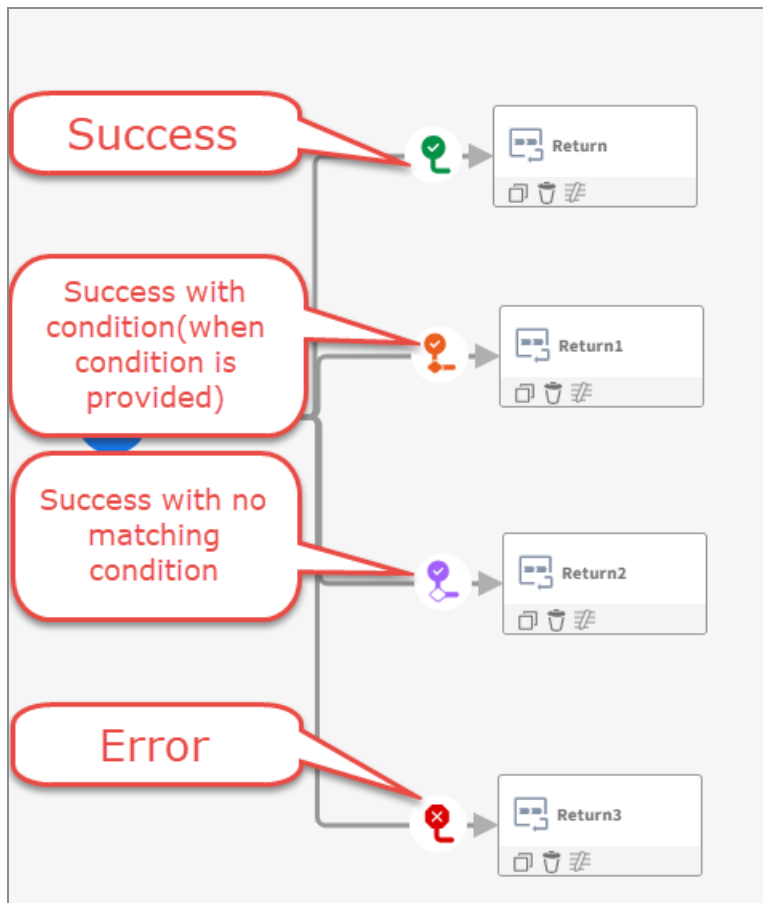
3. Click **Save**.
4. If you select **Success with condition**, the mapper opens for you to set the condition. Click **condition**.

The mapper is exposed to the right of the dialog. The functions that you can use to form the condition are shown under **Functions**.


5. Enter an expression with the condition or click a field from the output of a preceding Activity to use it. The output from preceding activities appears under the left **Upstream Output** in a tree format.

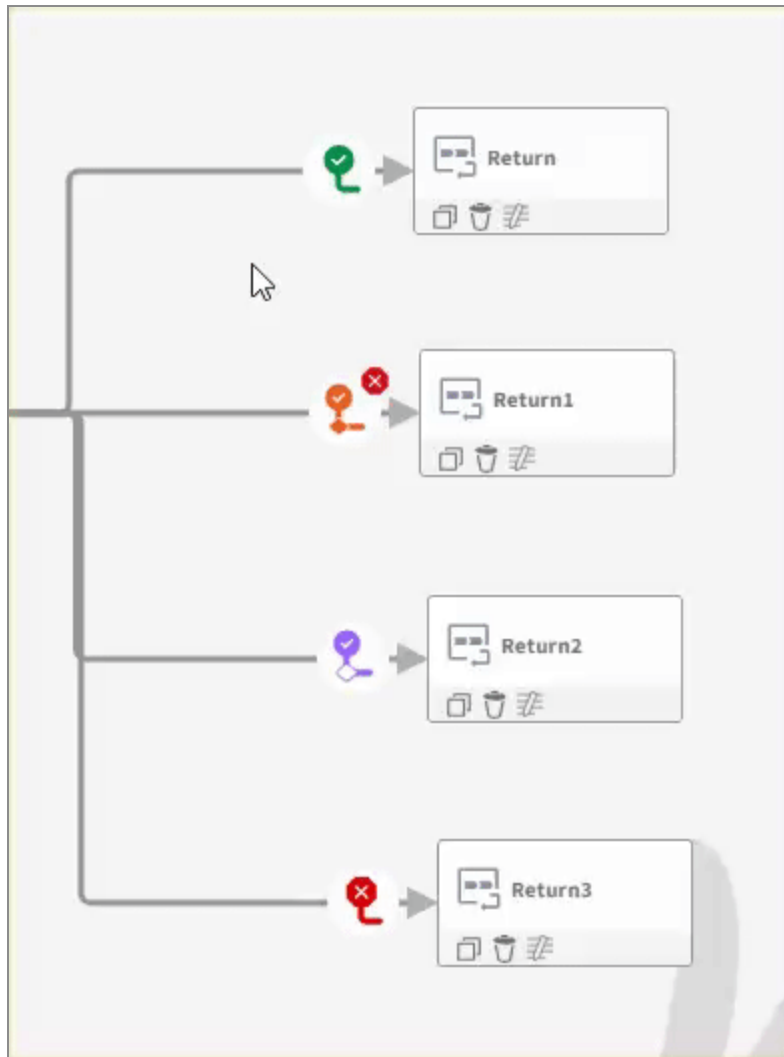
Key Considerations When Setting Branch Conditions

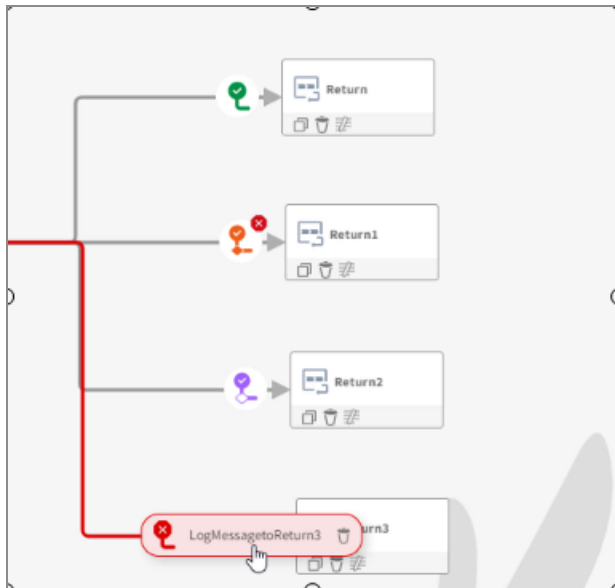
- The condition must resolve to a boolean type. The following image shows how the branches appear based on the branch condition:



- When you hover over the branch lines or the branch labels, they appear in different colors according to the condition that is set.
 - Green - Success
 - Orange - Success with condition
 - Purple - Success with no matching condition
 - Red - Error.

These lines indicate the exact start and end points of the connection between any two activities. This is helpful in large and complex flows where the exact flow seems unclear and jumbled. The branch labels indicate the names of the activities that are connected. You can rename the labels as per requirement. For the **success with condition** label, when it is empty or when there is a wrong condition, the  icon appears on it.




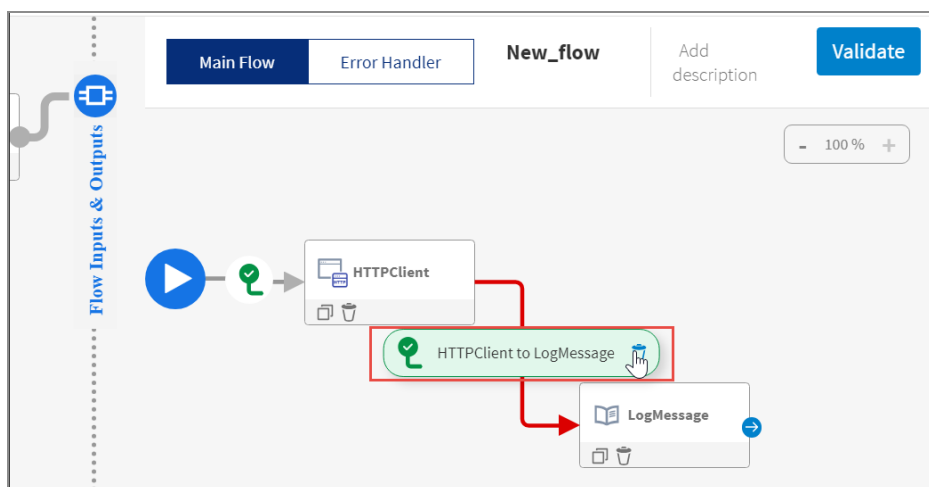


Deleting a Branch

You can delete a branch at any time after creating it.

To delete a branch:

1. Hover over the branch that you want to delete. A branch label appears.
2. On the label, click the  icon that appears.



3. On the confirmation dialog, click **Delete**. The selected flow is deleted.

Editing a Flow

You can edit the flow name or its description after creating the flow. You can also add more activities. Rearrange existing activities by dragging them to the desired location or delete activities from the flow.

To edit a flow:


1. Click the app in the **File Explorer** tab of Visual Studio Code to open app details page.
2. Click the flow name that opens the flow page. Revalidate the app after making the required changes.


To edit the flow name, click anywhere in the flow name and edit the name. To add an activity between two existing activities, you can make a space by dragging the activities to anywhere you want in the activities area.

Deleting a Flow

You can delete a flow from the app details page.

To delete a flow:

1. Click the app in the **File Explorer** tab of Visual Studio Code to open app details page.
2. Hover over to the extreme right of the flow name until the **Delete flow** icon  displays.
3. Click the **Delete flow** icon.
4. On the confirmation dialog, click **Delete**. The selected flow is deleted.

 **Note:** If multiple flows are attached to a trigger only the specific flow gets deleted. If there is only one flow attached to the trigger, the trigger also gets deleted.

Adding an Activity

After a flow is created, you must add activities to the flow.

1. Click the app in the **File Explorer** tab of Visual Studio Code to open app details page,

then click the flow name to open the flow details page.

2. Click the **Activities** palette available on the right side. The categories of the activities are displayed.
3. Click the category from which you want to add an activity. For example, to add a general activity such as **Log Message**, click the **General** category.
4. Drag the required activity to the activities area.
5. To change the order in which the activities appear in the flow, you can drag the activity anywhere in the activities area.
6. Click the activity to open its configuration dialog and configure it.

✔ **Tip:** If you want to add an activity in between two activities, you can directly drop the activity on the branch label in between the two activities. You need not delete the incoming and outgoing connections and reconnect them. Adding an activity between two activities is only possible when an activity is dragged and dropped from the **Activities** panel, not for the activities already present on the flow canvas.

Copying or Pasting Multiple Activities

This feature allows users to copy and paste multiple activities within the same app and across different applications. It is only applicable for activities, not for triggers.

❗ **Note:** App properties, app schemas, app specs, and connections cannot be copied to a different app.

✔ **Tip:** Before copying and pasting activities from other apps that are bound with app schemas and app specs, use the **Import App** option to import the app schema and app specs.

Procedure

1. Select the activities that you want to copy. You can select multiple activities by pressing Ctrl and click. Or, select and hold to create a selection box around the

activities.

2. Right-click to **Copy** or **Paste** the selected activities.
 - a. To copy activities, select the **Copy activities** option.
 - b. To paste activities, select the **Paste activities** option.
3. You can use the keyboard shortcuts in Visual Studio Code. To add custom key bindings, see [Adding Custom Key Bindings](#).

Validating After Pasting Activities

After you paste the activities, a validation process is triggered automatically. If the validation detects any issues, you must resolve them before proceeding. To resolve the issues, perform the following steps:

Procedure


1. If the error relates to disconnected activities, ensure that all pasted activities are properly linked to the rest of the workflow.
2. Make the necessary adjustments and run the validation process again.
3. If any other errors are reported, address them as required.


After successful validation checks, the pasted activities are ready for use.

Adding Custom Key Bindings

To add custom key bindings, perform the following steps:

Procedure

1. Go to **File > Preferences > Keyboard Shortcuts**. Or, press Ctrl+K and Ctrl+S to open **Keyboard Shortcuts**.
2. To add a keybinding for copying, search flogo copy.
3. Click the  icon next to the command (flogo.copy activities). Press the keys that you want to use and **Enter**.
4. To add a keybinding for pasting, search flogo paste.

5. Click the  icon next to the command (flogo.paste activities). Press your preferred keys and **Enter**.

Searching for a Category or Activity

You can search an activity or category by entering the activity or category name in the **Search** box of the **Activity** palette.

You can enter either the full or partial name (a string of characters appearing in the name) of the activity or category in the **Search** box.

- All categories whose names either wholly match the search string or contain the partial search string in their name get displayed.
- Only those activities in the category whose names contain the search string are displayed in the search results. The activities in the category whose names do not match or contain the search string are not displayed in the search results.
- For any activity whose name wholly or partially matches the search string, the category that contains that activity is displayed. For example, if you enter "delete" in the search box, since there are activities whose name contains the string "delete" in Marketo, Salesforce, Zoho-CRM, all these categories are displayed, even though the category names themselves do not contain the string "delete".

Configuring an Activity

After adding an activity, you must configure it with the required input data. Also configure the output schema for activities that generate an output.

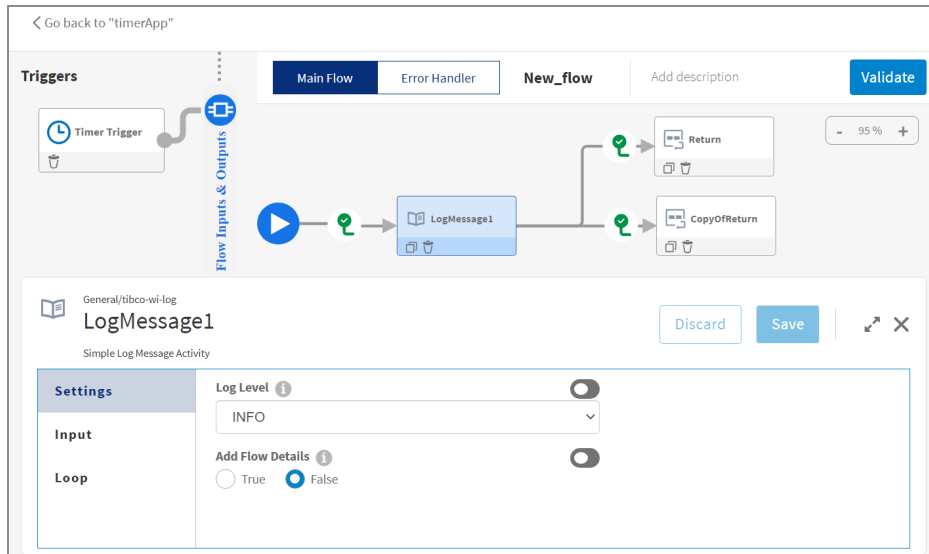
There are three ways to configure data for an activity:

- Configuring static data where you manually type the data in the mapper for the field. For example, type in a string that you want to output. Strings must be enclosed in double quotes. Numbers must be typed in without quotes.
- Mapping an **Activity** input to the output from one of the activities preceding it in the flow, provided that the previous activities have some output.
- Using functions. For example, the concat function to concatenate two strings.

After configuring or modifying the configurations in any activity, you must explicitly **Save** or **Discard** the changes.

To configure an activity:

1. On the flow details page, click an activity. The configuration box opens under the activity.



2. Click each tab in the configuration box under the activity name and either manually enter the required value, use a function, or on the **Input** tab, map the output from the trigger or a preceding activity using the mapper. For more information on details of mapping, see [Mapper](#).


If one or more activities are not configured properly in a flow, the error or warning icon is displayed in its upper-right corner. Click the activity whose tab contains the error or warning. For more information, see [Errors and Warnings](#).

Duplicating an Activity

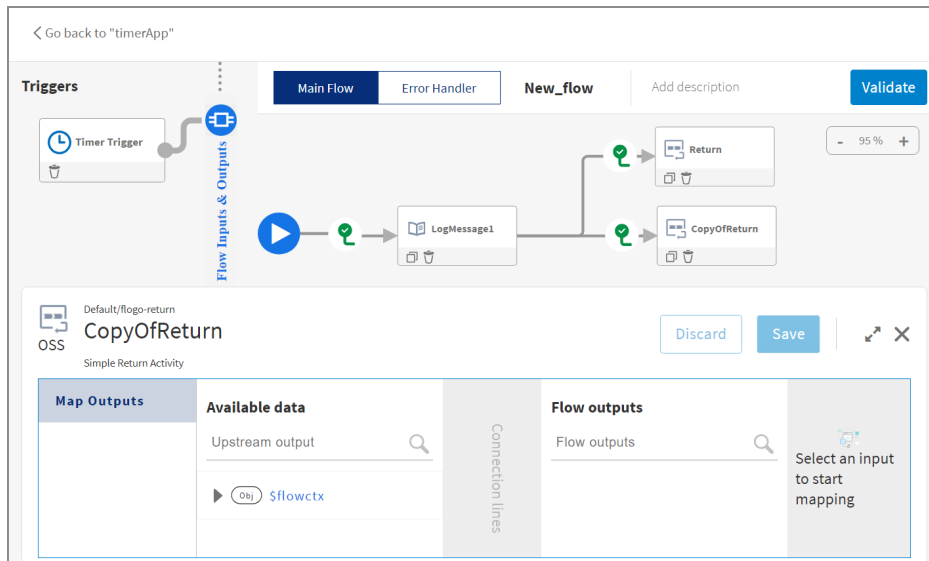
You can duplicate an activity within the same flow. The activity along with the existing configuration is duplicated into a new activity. The duplicate of the original activity is created with a default name beginning with CopyOf. You can rename the activity by clicking the activity name. Duplicating an activity saves you time and effort in situations when you want to create an activity with similar or the same configurations as an existing activity in the flow. After you duplicate the activity, you can change the configuration, move it around in the flow by dragging it to the required location or delete it from the flow.

Note: A trigger within a flow cannot be duplicated.

To duplicate an activity:

1. From the **Apps** page, click the app name and then click the flow name to open the flow details page.
2. Hover over the activity that you want to copy and click the  icon.

For example, in the following screenshot, the **Return** activity is duplicated and added to the flow. The duplicate activity is **CopyofReturn**.



3. Configure the duplicated activity as required.

Using the Loop Feature in an Activity

When creating a flow, you may want to iterate a certain piece of logic multiple times. For example, you want to send an email about an output of a certain activity *activity1* in your flow to multiple recipients. To do so, you can add a **SendMail** activity following *activity1* in your flow. Then configuring the **SendMail** activity to iterate multiple times when *activity1* outputs the desired result. Each iteration of the **SendMail** activity is used to send an email to one recipient.

Keep the following in mind when using the Loop feature:

- Iteration is supported for an activity only. You configure the iteration details on the **Loop** tab of the activity.
- The **Loop** tab is unavailable for certain activities that do not require iteration. For

example, the **Return** activity. Its purpose is to exit the flow execution and return data to the trigger.

- You cannot iterate through a trigger.

To configure multiple iterations of an Activity:

1. Click the Activity in the flow to expose its configuration tabs.
2. Click the **Loop** tab.
3. Select a type of iteration from the **Type** menu.

The default type is **None**, which means the Activity does not iterate.

Iterate

This type allows you to enter a number that represents the number of times you would like the Activity to iterate without considering any condition for iterating.

Click **iterator** to open the mapper to its right. You can either enter a number (integer) to specify the number of times the activity must iterate or you can set an expression for the loop by either entering the expression manually or mapping the output from the preceding activities or triggers. You can also use the available functions along with the output from previous activities and/or manually entered values to form the loop expression. The loop expression determines the number of times the activity iterates.



Warning: The loop expression must either return a number or an array. The array can be of any data type. If your loop expression returns a number, for example 3, your activity iterates three times. If your loop expression returns an array, the activity iterates as many times as the length of the array. You can hover over the expression after entering the expression to make sure that the expression is valid. If the expression is not valid, a validation error is displayed.

If you select this type, the **Input** tab of the Activity displays the \$iteration scope in the output area of the mapper. \$iteration contains three properties, **key**, **index**, and **value**. **index** is used to hold the index of the current iteration. The **value** holds the value that exists at the index location of the current iteration if the loop expression evaluates to an array. If the loop expression evaluates an array of objects, **value** also displays the schema of the object. If the loop expression evaluates to a number, the **value** contains the same integer as the **index** for each iteration. To examine the

results of each iteration of the Activity, you can map the **index** and **value** to the **message** input property in the **LogMessage** Activity and print them. The **key** is used to hold the element name when configuring a condition if the value evaluates to an object. However, you can map only to the output of the last iteration if you did not set the **Accumulate Output** checkbox to **Yes**. For more information, see [Accumulating the Activity Output for All Iterations](#).

Repeat while true

For more information, see [sample](#) for an example of how to use this feature.

Select this type if you want to set up a condition for the iteration. This acts like the do-while loop where the first iteration is run without checking the condition and the subsequent iterations exit the loop or continue after checking the condition. You set the condition under which you want the activity to iterate by setting the **condition** element. The condition gets evaluated before the next iteration of the activity. The activity iterates only if the condition evaluates to true. It stops iterating once the condition evaluates to false. Click **condition**, and manually enter an expression for the condition. For example, `$iteration[index] < 5`.

Keep in mind that the index for the **Repeat while true** iteration begins at zero and iterates $n+1$ times. If you enter 4 as the iterator value, it runs as the following iterations: 0,1,2,3,4.

By default, the results of only the final iteration are saved and available. All previous iteration results are ignored. If you would like the results of all iterations to be stored and available, set **Accumulate** to **Yes**.

You have the option to set a time interval (in ms) between each iteration, which can help you manage the throughput of your machine. To spread the iterations out, set the **Delay** element. The default delay time is 0 ms, which results in no delay.

Result

After you enter the loop expression, the loop icon appears on the upper-right corner of the activity.

Accumulating the Activity Output for All Iterations

When using the Loop tab to iterate over an Activity, you have the option to specify if you want the Loop to output the cumulative data from all iterations. You can do so by setting the **Accumulate** checkbox to **Yes**.

When the **Accumulate** checkbox is set to **Yes**, the activity accumulates the data from each iteration and outputs that collective data as an array of objects. Here, each object contains the output from the corresponding iteration. The accumulated results are displayed as an array in the downstream activities in the mapper and are available for mapping.

When mapping to an element within an object in the output array of the activity, you must provide the index of the element to which you want to map. For instance, when you click a property within the object under **responseBody**, the expression displayed in the mapper is `$activity [<activity-name>] [<<index>>].responseBody.<property-name>`. Replace `<<index>>` with the actual index of the object to whose property that you want to map.

When the **Accumulate** checkbox is not selected, the output of the Loop displays an object that contains only the data from the last iteration. Data from all previous iterations is ignored. When mapping to an element in the output object of the activity, when you click a property within the object under **responseBody**, the expression displayed in the mapper is `$activity [<activity-name>].responseBody.<property-name>`.

The **Output** tab of the activity changes based on your selection of the **Accumulate** checkbox. The parent element (the name of the activity and the data type of the iteration output) is displayed regardless of your selection. If you set the **Accumulate** checkbox to **Yes**, the data type of the parent element is an array of objects. If you did not select the checkbox, the data type of the parent element is an object. The **Output** tab contents are also available in the mapper allowing for the downstream activities to map to them.

Accessing the Activity Outputs in Repeat While True Loop

This feature is useful when an activity needs to use the loop feature to do batch processing or fetch multiple records by running the activity multiple times. With each iteration of the activity, the output is available for mapping to the activity input.

This feature is available in all activities that generate an output (have an **Output** tab).

To use this feature:

1. On the **Loop** tab, set the **Type** to **Repeat while true**.
2. Set the **Access output in input mappings** to **Yes**.

This makes the output of the activity iteration available in the **Upstream Output** for mapping. Now you can map your output as a next input parameter.

3. Enter a **condition** in its text box. The activity evaluates this condition before each run. If the condition evaluates a true value, the activity runs.

i Note: The output is only available in subsequent iterations after the first iteration. Since the activity output is not available for the first iteration, your condition must perform a check to see if it is the first iteration of the activity.

For example, use `$iteration[index]> 0 && isdefined($activity[SFQuery].output.locator)` to begin your condition. The `$iteration[index]> 0` checks to make sure that it is not the first run of the activity. The `isdefined($activity[SFQuery].output.locator)` function checks whether the output field exists.

Using the Retry On Error Feature in an Activity

Using the **Retry on Error** tab, you can set the number of times the flow tries to run the activity on encountering an error that can be fixed on retrieval. The errors such as waiting for a server to start, intermittent connection failures, or connection timeout can be fixed on retrieval.

You can set the count and the interval in one of the following ways:

- Manually type the value in the mapper
- Map the value from the previous Activity
- Select a function from the list of functions
- Map app property to override the values


Field	Description
Count	The number of times the flow should attempt to run the activity. This value must be an integer.
Interval (in millisecond)	The time to wait in between each attempt to run the activity. This value must be an integer.

i Note: The **Count** and **Interval** fields are mandatory. By default, the values are set to 0.

Deleting an Activity

You can delete an activity in a flow from the flow details page.

To delete an activity:

1. On the **Apps** page, click the app name then click the flow name to open the flow details page.
2. Hover over the activity that you want to delete and click the  icon.

Triggers

Triggers are used to activate flows. This section contains information on creating and managing triggers in your app.

Deleting a Trigger

You can delete a trigger from the app details page by hovering over the trigger and clicking **Delete**.

Playing a Test Case Once


After you design a flow, you can test it by playing it once.

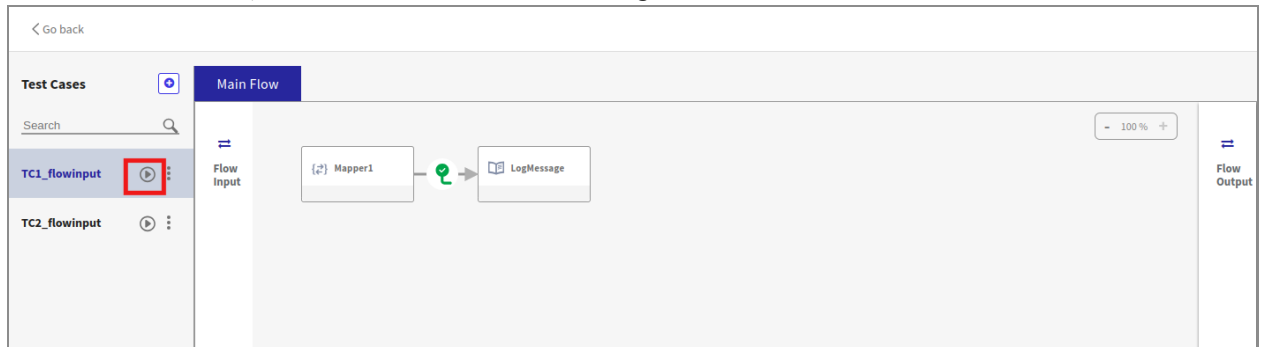
When designing a flow, runtime errors can go undetected until you build the app to execute the flow. It can be cumbersome to test flows that start with a trigger as the triggers activate based on an external event. So, before you can test the flow, you must configure the external app to send a message to the trigger to activate it and consequently execute the flow. The play a test case once feature eliminates the need to activate the trigger to execute the flow.

You provide the input to the flow in playing a test case once. It executes the flow on demand without using a trigger. Each activity executes independently and displays its logs. It can help detect errors in the flow upfront without actually building the app. You can use several modes, such as **Mock Output** and **Skip Execution**. For more information on modes, see [Creating and Running a Test Case](#).

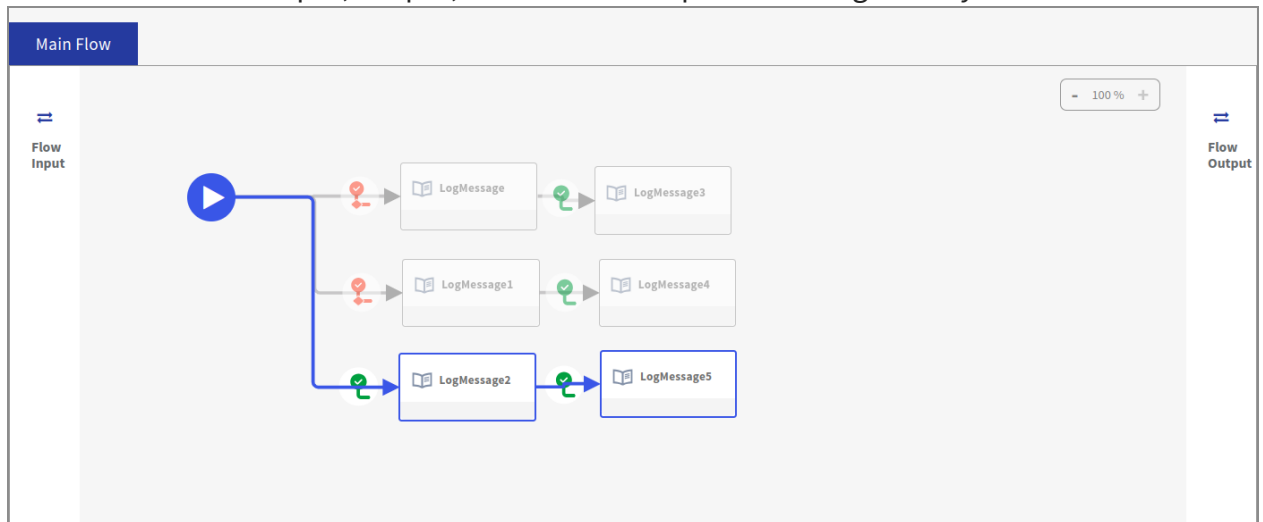
After creating a test case, it is ready to run in a local runtime environment. You can play a test case once to test against a particular set of flow inputs.

Procedure

1. Create a test case. For more information, see [Creating and Running a Test Case](#).
2. To run a test case, click the  icon on the .flogotest file.



3. View the execution logs of the flow in the terminal. The execution path is highlighted in blue and other activities appear grayed out. You see a new or updated test results file with additional input, output, or error data captured during activity execution.





4. Click the executed tasks to see the inputs, outputs, or errors of the activity in read-only mode. If an activity is executed and does not have any configuration or output, you cannot click it after execution.



When a test case is running, the **Main Flow** tab, the **Error Handler** tab **Stop Testcase**, **Go Back**, and **Zoom in/out** buttons are enabled. All inactive test cases, search and addition of new test cases, Flow input, and Flow output are unavailable.



Tip:

- You must stop the testing mode to configure the test case.
- When a test is running, the  icon changes to the  icon. You must stop an ongoing test case to run a different one.

Handling Errors

If an activity encounters errors, it is highlighted with a red border.



You can perform the following steps:

1. Click the activity to see a detailed error message.
2. Go to **Error Handler** if the main flow activity has an error handler flow.

Using App Executable

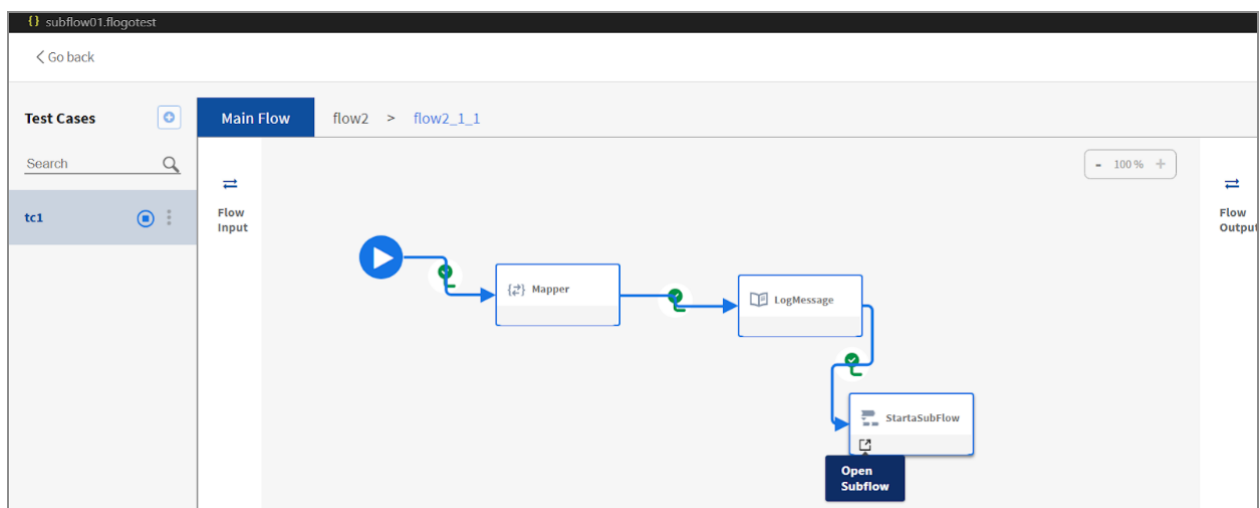
Using this feature, you can test a case with the app executable.

1. To enable this feature, use the app executable.
2. Use the `--test-preserve-io` flag in test commands for running a unit test. For more information, see [Unit Testing for the CI/CD](#).
Or, set the environment variable `FLOGO_UT_PRESERVE_IO` to true.

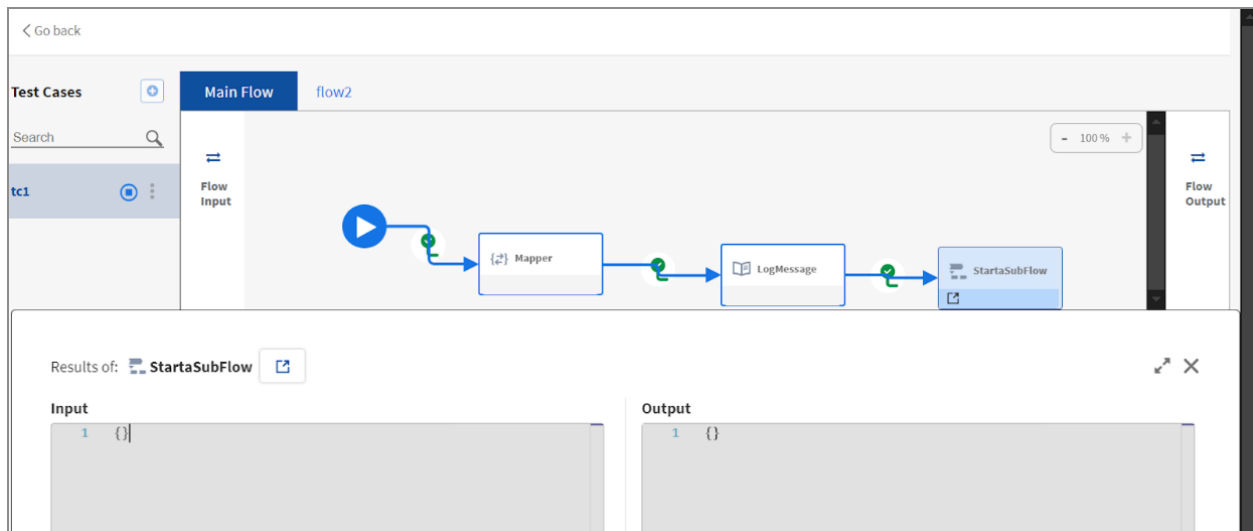
If Step 2 is not done, the system defaults to the previous behavior. After the execution is complete, you can see the executed activity logs in the terminal and the test results file is updated.

Drilldown Support for Subflow Activities in Test Play Mode

You can visualize the execution of all activities within subflows for enhanced visibility into end-to-end execution, including nested subflows.



You can open the invoked subflow either from the subflow tile or directly from the subflow configuration.



Note: If the .flogo file is changed during test play execution, the process stops. To ensure uninterrupted execution, avoid updating the .flogo file during the test play. It is a best practice to stop the .flogotest file, make the necessary changes, and re-run the unit test file in play mode.

Synchronizing a Schema Between Trigger and Flow

If you change the schema that you entered when creating the trigger, you must explicitly save any changes you make, then propagate the changes to the flow input and flow output. This is done by synchronizing the schemas.

To synchronize the schema between the trigger and the flow:

Procedure

1. Click the trigger to open its configuration details.
2. Make your changes and click **Save**. If you do not click **Save**, a warning message is displayed asking you to first save your changes before the schema can be synchronized.
3. Click **Sync** on the upper-right corner.

The trigger output schema is copied to flow inputs and the trigger reply schema is copied to flow outputs.

Data Mappings

Use the graphical data mapper to map data between:

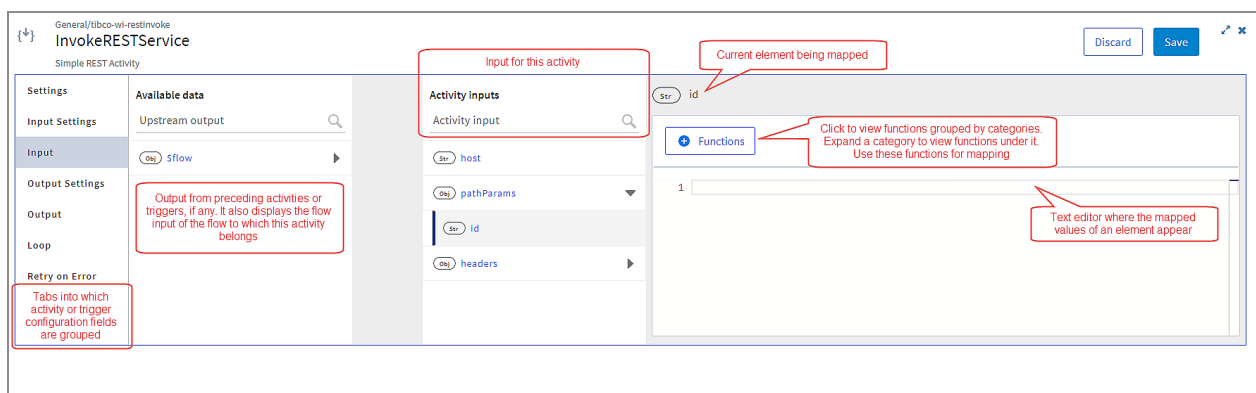
- Activities within a flow
- Trigger and the flows attached to the trigger

Enter the flow or activity input values manually or map the input schema elements to output data of the same data type from preceding activities, triggers, or the flow itself.

Data Mappings Interface

An activity has access to the output data from the trigger to which the flow is attached. It also has access to the output from any of the activities that precede it in the same flow provided that the trigger or activity has an output. This data is displayed in a tree structure under **Available data** in the mapper. The input schema for the activity is displayed in the **Activity inputs** pane to the right of the **Available data** pane. You can map data coming from the upstream output to the input fields of the activity. Also, each activity has access to the input fields of the flow to which the activity belongs. You can enter the flow input schema on the **Input Settings** tab of the **Flow Inputs and Outputs** tab.

When you click an activity or trigger on the flow details page, the configuration page for that activity or trigger opens. The following image is an example of the configuration page that opens when you click the **InvokeRESTService** activity.



The left-most pane displays the tabs for the configuration fields for that activity or trigger. Each activity or trigger has one or more of the following tabs:

- **Settings**

For triggers, this tab is displayed as **Trigger Settings**. This tab shows the activity settings, trigger settings, or handler settings.

- Activity settings are specific to the activity.
- Trigger settings are specific to the particular trigger.
- Handler settings apply to a specific flow attached to the trigger. Each flow attached to the trigger can have its own handler settings.

- **Input Settings**

On this tab, you can enter the schema for the flow or activity input.

- **Input**

This tab is displayed for activities and shows the schema that you entered on the **Input Settings** tab in a tree format. You can manually enter values for any elements in the input schema or map any input element to the output from previous activities or triggers on this tab.

- **Output Settings**

On this tab, you can enter the schema for the flow or activity output.

- **Output**

This tab displays the schema that you entered on the **Output Settings** tab in a tree format. The schema displayed on this tab is set to read-only mode as it is for informational purposes only.

- **Map to Flow Inputs**

The settings on this tab must be configured only if your trigger has an output, for example, in the REST or GraphQL triggers. You can manually enter or map the elements from the trigger output (schema set on the **Output Settings** tab) to the flow input elements (schema entered on the **Input Settings** tab of the **Flow Inputs & Outputs** tab). This allows the output from the trigger to become the input to the flow.

- **Reply Settings**

This tab is applicable only to triggers that send replies to the caller, such as the REST or GraphQL triggers. You enter the trigger reply schema on this tab.

- **Map from Flow Outputs**



This tab is specific to triggers that need to send a reply to the caller, such as the REST or GraphQL triggers. You manually enter or map the elements from the output of the flow (schema set on **Reply Settings** tab) to the flow output elements (schema entered on the **Output Settings** tab of the **Flow Inputs & Outputs**). This allows the output of the flow to become the reply that the trigger sends back to the request that it receives.

- **Loop**

On this tab, enter the iteration details for activities that you want to iterate.

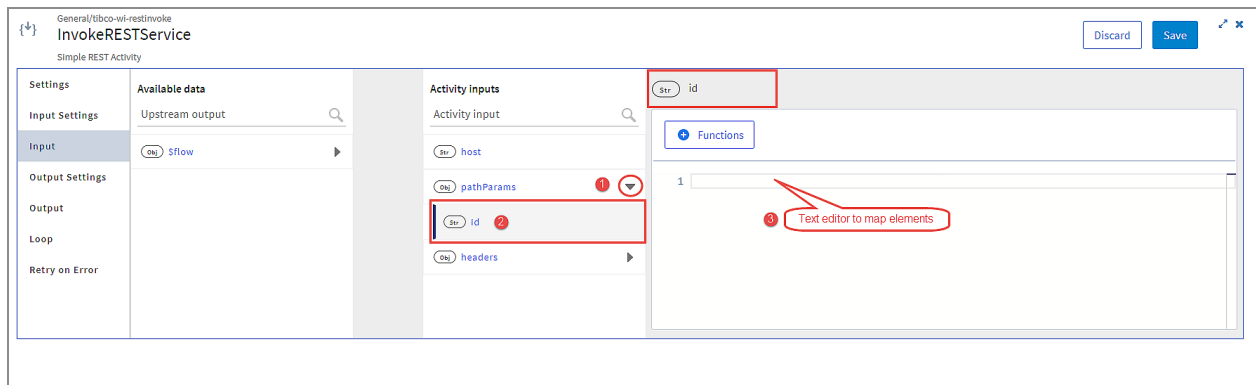
When mapping, you can use data from the following sources:

- **Literal values** - literal values can be strings or numeric values. These values can be either manually typed in or mapped to a value from the output of the trigger or a preceding activity in the same flow. To specify a string, enclose the string in double quotes. To specify a number, type the number in the text box for the field. Constants and literal values can also be used as inputs to functions and expressions.
- **An input element that is directly mapped to an element of the same type in Available data.**
- **Mapping using functions** - the mapper provides commonly used functions that you can use with the data to be mapped. The functions are categorized into groups. Click a function to use its output in your input data. When you use a function, placeholders are displayed for the function arguments. Click a placeholder argument within the function and drag an element from **Available data** to replace the placeholder. Functions are grouped into logical categories. For more details, see [Using Functions](#).
- **Expressions** - You can enter an expression whose evaluated value is mapped to the input field. For more details, see [Using Expressions](#).

The error and warning icons are displayed on the **Activity inputs** pane, on the configuration fields in the left-most pane, and on the activity tile. In case of errors in mapping (such as empty mandatory fields and incorrect mapping at activity or trigger level), an error icon  is displayed. A warning icon  is displayed if your changes are not saved or discarded, input and output are not mapped in triggers, or mappings are removed for mandatory fields.

Mapping Data from the Data Mappings Interface

In the following example, in the **Activity inputs** pane, clicking the arrow expands the object **pathParams**. You can select the input (in this case, **id**) that you want to map. A section with a text editor opens on the right side in the mapper.

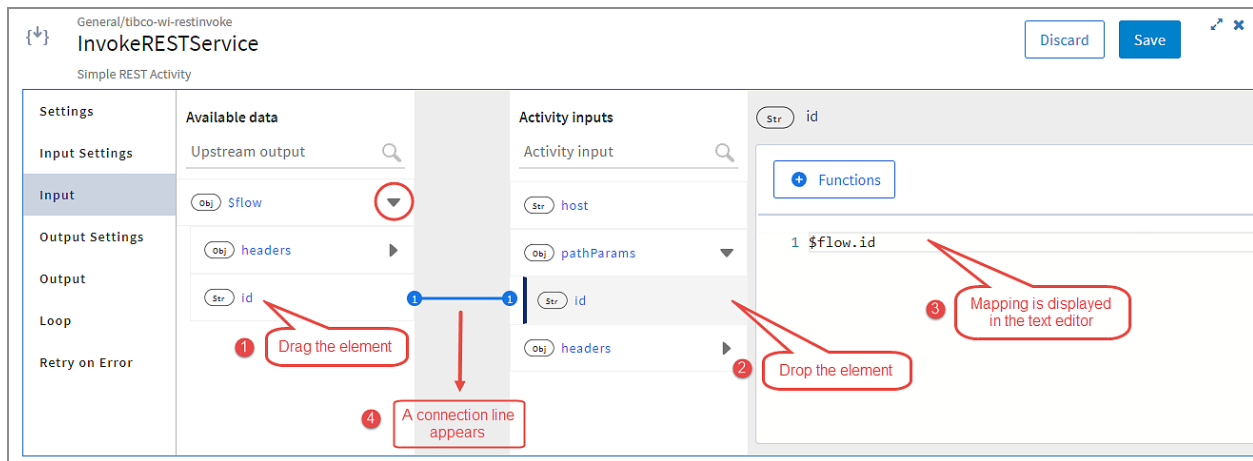


To map data coming from the upstream output to the input fields of the Activity:

In the **Available data** pane, click the arrow to view the fields. You can map an element from the **Activity inputs** pane to an element in the **Available data** pane using one of the following methods:

- Drag the element from **Available data** and drop it on the input in the **Activity inputs** pane. The mapping is displayed in the text editor.
- Click the element from the **Activity inputs** pane. The text editor opens on the right side of the mapper. Drag the element from the **Available data** pane and drop it in the text editor.
- Click the element from the **Activity inputs** pane and double-click the element in the **Available data** pane to map it to the input.

A connection line appears to show the mapping between the **Available data** and the **Activity inputs**.

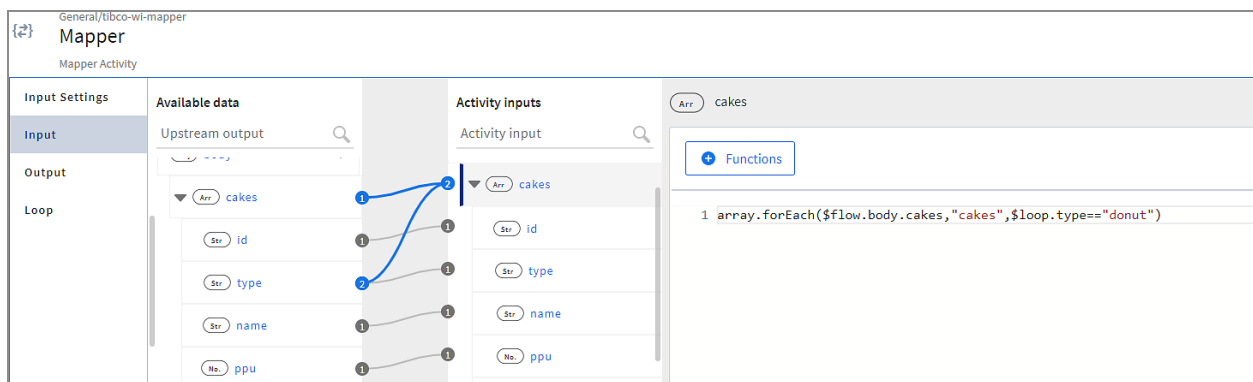


To add functions in the mapper, see [Using Functions](#).

Connection Lines

Connection lines show the mapping between the data and the input. These lines appear when you map an element from the **Available data** with an element from the **Activity inputs**. The lines also appear for mapped arguments. When the mapped element is selected in the **Activity inputs** pane, the connection line is blue. Otherwise, it is gray. The numbers at the ends of a connection line indicate the total number of mapped elements for a particular element.

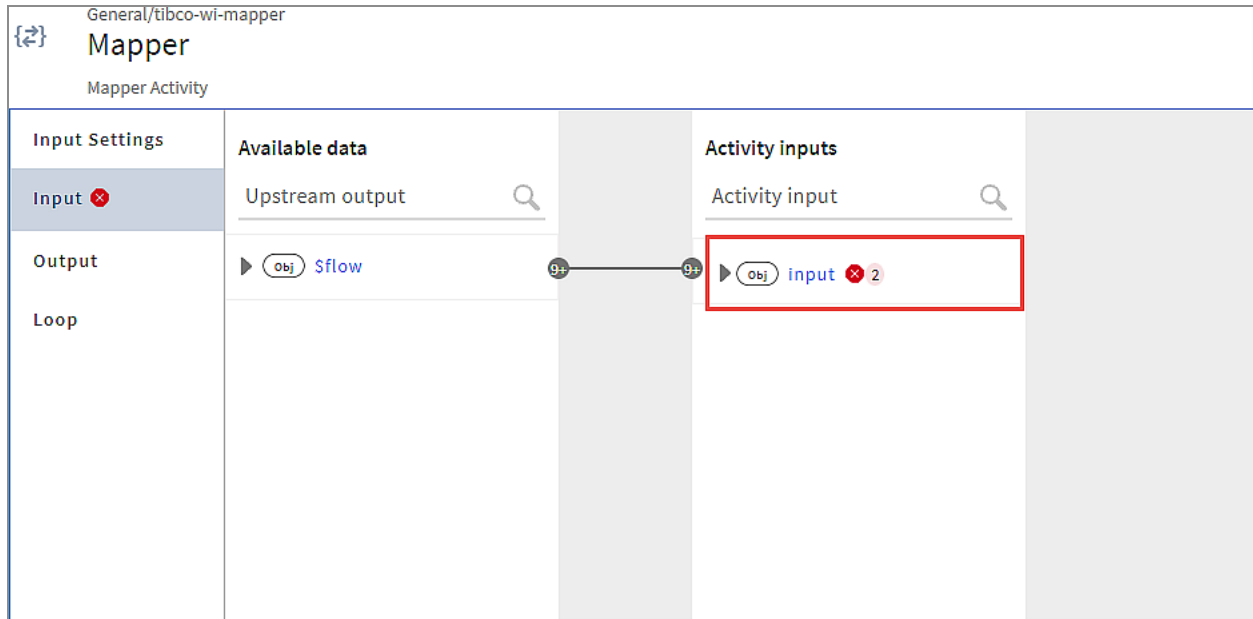
The following screenshot shows the connection lines and the total count of mappings for each element.



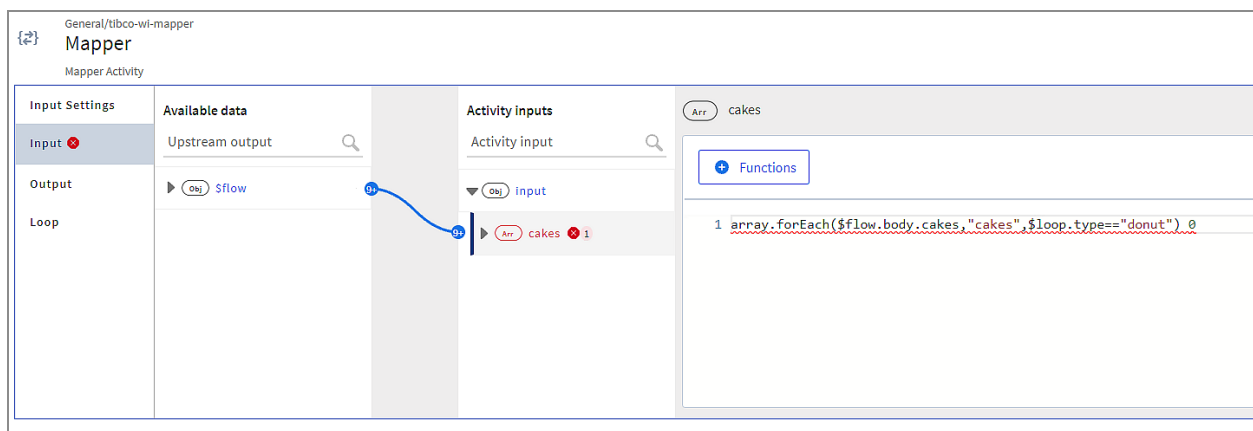
Errors in Mapping

In the mapper, you can see the total count of errors and warnings each in the mapping next to the parent object in the **Activity inputs** pane.

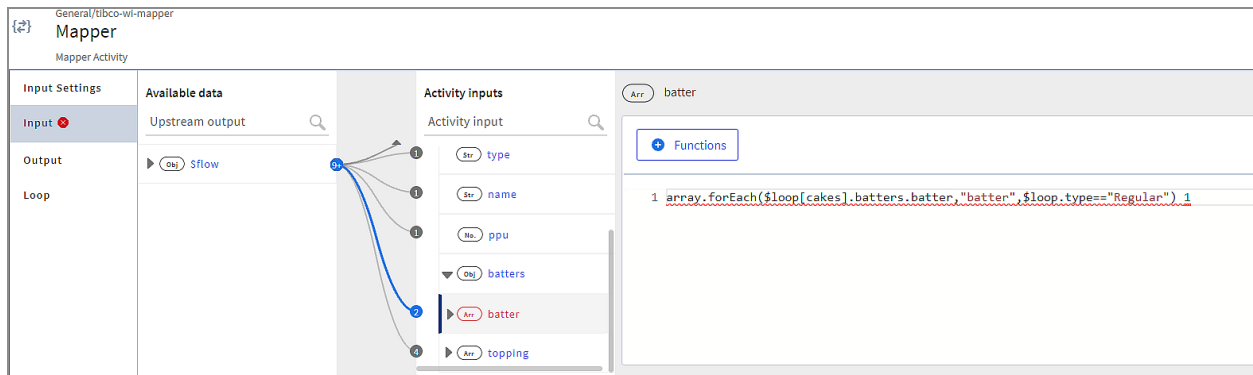
In the following example, the parent object **input** has a total of two errors in mapping.



Expanding the **input** object shows that the array **cakes** is mapped incorrectly. Notice that **cakes** contains one element with incorrect mapping.



Expanding the array **cakes** shows that the array **batter** under the object **batters** has an error in mapping.



Note: The errors in mappings are also observed when the property in the app properties dialog is edited, moved from a group to another or from a group to top level as a standalone property. A warning message about the same, pops up on the screen when you edit any properties.

Scopes in Data Mappings

The **Available data** pane in the mapper displays the output data from preceding activities, triggers, and flow inputs. This area groups the output elements based on a scope. A scope represents a boundary in the **Available data** within which an input element can be mapped. For example, when mapping an input element to an element from the output of a trigger, the scope of the input element is represented in **Available data** as **\$trigger**. The following scopes are currently supported by the mapper.

Scope Name	Used to...	Available in...
\$trigger	Map flow input to trigger output.	Trigger (Map to Flow Inputs tab) to map flow inputs to trigger outputs.
\$flow	Map flow output to trigger reply.	<ul style="list-style-type: none"> Trigger (Map to Flow Outputs tab) to map flow output to trigger reply. Activities (Input tab) to map Activity input to flow input. Return Activity (Map Output tab) to map flow output to flow input.

Scope Name	Used to...	Available in...
\$Activity. [Activity-name]	Map input elements of the Activity to elements from the output of previous activities.	\$activity represents the scope of an activity. [activity-name] indicates the activity whose scope that you are defining. Each preceding activity has its own scope in the mapper.
\$iteration	This keeps a record of the current iteration and is available only when the iterator is enabled for an activity on the Loop tab.	<p>Input tab of an Activity that has Loop enabled. This tab is displayed only when the Loop for the Activity is enabled. The following elements are displayed under \$iteration:</p> <ul style="list-style-type: none"> • key - This element represents the iteration index. Thus, it is always of type number. For example, if the Loop expression is set to an array, the key element represents the array index of the current iteration. • value - The value can be of any type depending on what is being iterated. For example, if you are iterating through an array of strings, the value is of type string.
\$property [property-name]	Map to app properties that are defined in the app.	<p>For any app that has app properties defined, this scope is available for mapping from any activity that allows mapping. Even the app properties from the connection are available for mapping under this scope.</p> <p>All the mapped configurations can be pre-checked using a flow tester or by creating a pre-check flow.</p>
\$loop [LoopName]	Map elements within an array.	\$loop[LoopName] variable provides direct access to the current element during array iteration. When mapping elements within an array, use \$loop [LoopName], where LoopName is the name assigned to the loop. The scope of this variable is the current array being iterated.

Scope Name	Used to...	Available in...
		<p>Note: As of Flogo Extension for Visual Studio Code version 1.3.2, it is a best practice to use the \$loop [LoopName] syntax. The previous usage of \$loop (for example, \$loop.elementName) remains compatible with existing flows.</p>
\$flowctx	Map the flow context details to the current flow.	<p>Input tab of every activity. The scope provides flow context details that can be mapped to any activity that allows mapping. Using this scope, the unique parameters, such as AppName, AppVersion, FlowId, Flowname, ParentFlowId, ParentFlowName, SpanId, TraceId, can be accessed in the flow and subflow.</p> <p>Here:</p> <ul style="list-style-type: none"> • The ParentFlowId and ParentFlowName is the ID and name of the flow that is invoking the current flow. • The TraceId is the unique ID of a single request, job, or an action initiated by the user. • The SpanId is the unique ID of the activity <p>Note: This scope is only available for the flow configuration and not for the trigger configuration.</p>

Data Types

Supported data types

The following data types are supported:

- BIT
- CHAR

- DECIMAL
- INTEGER
- TEXT
- NUMERIC
- REAL
- SMALLINT
- DATE
- TIMESTAMP
- MONEY
- ENUM
- JSON
- XML
- TINYINT
- VARCHAR
- SMALL MONEY

Unsupported data types

The following data types are not supported:

- BIGINT
- BINARY

Reserved Keywords to be Avoided in Schemas

Flogo uses some words as keywords or reserved names. Do not use such words in your schema. When you copy an app to the workspace, if the schema entered on the **Input** or **Output** tab of an Activity or trigger contains reserved keywords, such attributes are now treated as special characters and might cause runtime errors.

Avoid using the keywords listed below in your schema:

- break
- case
- catch
- class
- const
- continue
- debugger
- default
- delete
- do
- else
- enum
- export
- extends
- false
- finally
- for
- function
- get
- if
- import
- in
- index
- instanceof
- new

- null
- return
- set
- super
- switch
- this
- Generate
- true
- try
- typeof
- var
- void
- while
- with

Mapping Different Types of Data

The mapper opens when you click any element in the input schema tree on an activity configuration tab.

You can map the following elements:

- A single element from the input to another single element in the output.

i Note: If the single element comes from an array in the output, then you must manually add the array index to use. For example, `$flow.body.Account.Address[0]` city.

- A standalone object (an object that is not in an array).
- An array of primitive data type to another array of primitive data type.
- An array of non-primitive data types (object data type or a nested array) to another array of the same non-primitive data type.

Keep the following in mind when using the mapper:

- Make sure that you map all elements that are marked as required (have a red asterisk against them), whether they are standalone primitive types, within an object, or within an array. When mapping identical objects or arrays, such elements get automatically mapped, but if you are mapping non-identical objects or arrays, be sure to map the elements marked as required individually.
- The `in` and `new` attributes are treated as special characters if you use them in the schema that you enter in the REST Activity or trigger. For example, mappings such as `$flow.body["in"]` and `$flow.body["new"]` are not supported. If an app copied to workspace contains these attributes, it results in runtime errors.
- Use of the anonymous array is not supported on the **Flow Input & Output** tab and the **Return** Activity configurations. To map to an anonymous array, you must create a top-level object or a root element and render that.
- You cannot use a scope (identified with a beginning \$ sign) in an expression, for example, `renderJSON($flow, true)`. You can use an object or element under it, for example `renderJSON($flow.input, true)`.
- You can only map one element at a time.

Note: If the output element names contain special characters other than an underscore (_), they appear in bracket notation in the mapping text box.

In the following example, **name** under **Available data** does not contain any special characters. Hence, it is displayed in dot notation.

The screenshot shows the TIBCO Flogo extension interface. On the left, under 'Available data', there is a list of upstream outputs: \$flow (Obj), name (Str), strArray (Str[]), numArray (No[]), boolArray (Bool[]), and objArray (Arr). A blue line with a '1' at each end connects the 'name' output to the 'name1' output in the 'Flow outputs' list. On the right, the 'Functions' panel shows a mapping for 'name1' (Str) with the expression '\$flow.name' highlighted in a red box.

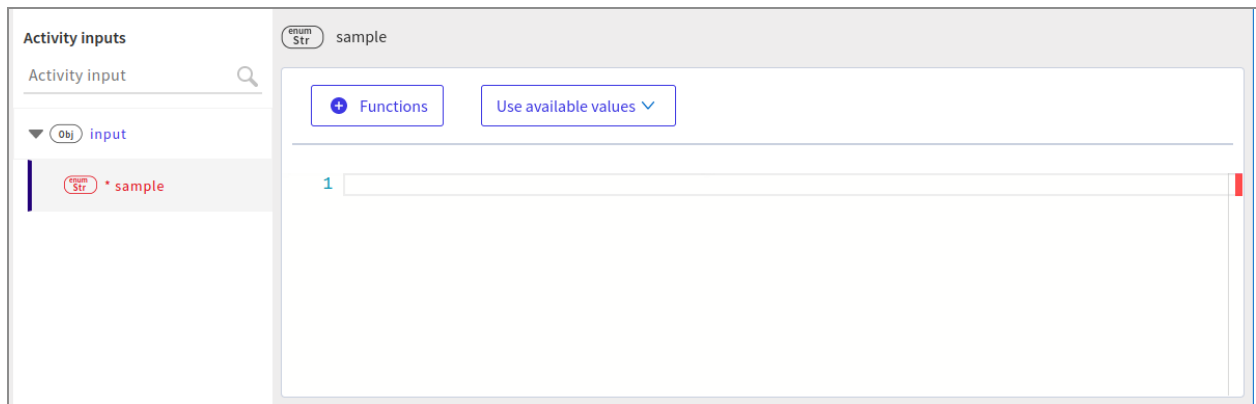
In the following example, **name 1** contains a space. Hence it appears in the bracket notation.

The screenshot shows the TIBCO Flogo extension interface. On the left, under 'Available data', there is a list of upstream outputs: \$flow (Obj), name 1 (Str), strArray (Str[]), numArray (No[]), boolArray (Bool[]), and objArray (Arr). A blue line with a '1' at each end connects the 'name 1' output to the 'name1' output in the 'Flow outputs' list. On the right, the 'Functions' panel shows a mapping for 'name1' (Str) with the expression '\$flow["name 1"]' highlighted in a red box.

Mapping an Enum value

You can map values of the Enum data type to the **Activity Inputs** element directly by selecting the values from the **Use available values** dropdown.

This feature is available for all activities and triggers that have a schema option.



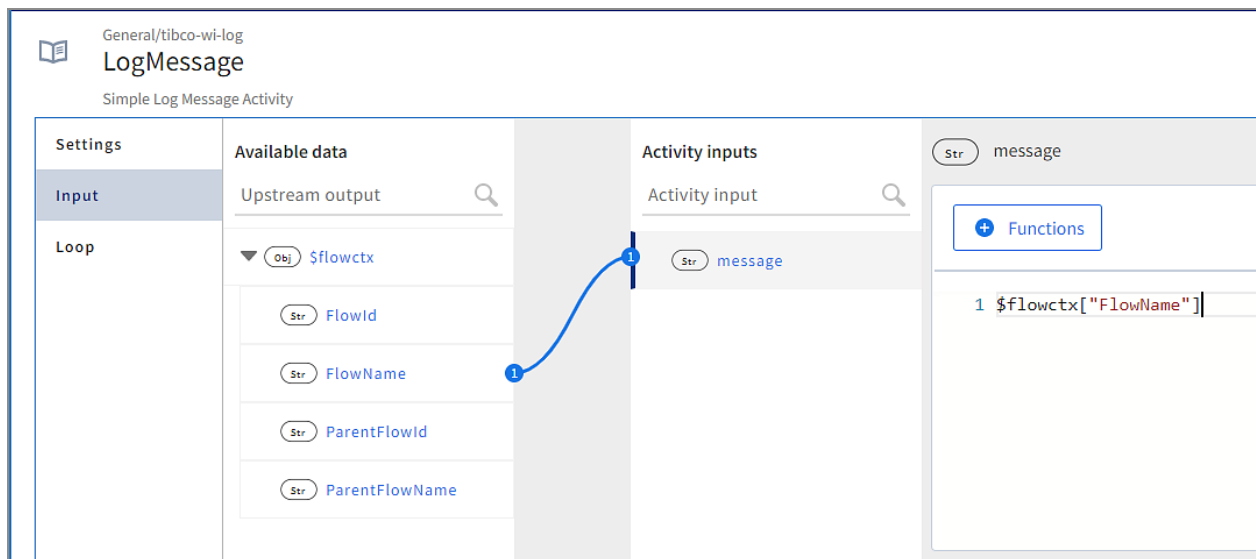
✓ **Tip:** Always use the **enum** keyword to identify the constant values.

Mapping a Single Element of Primitive Data Type

You can map a single element of a primitive data type to a single element of the same type in the output schema under **Available data**.

Drag the element from **Available data** and drop it on the destination element that you want to map in the **Activity inputs** pane.

In the following example, drag and drop **FlowName** (source) on **message** (destination) to map it. Alternatively, click **message**. Drag and drop **FlowName** in the text editor, or double-click **FlowName**.

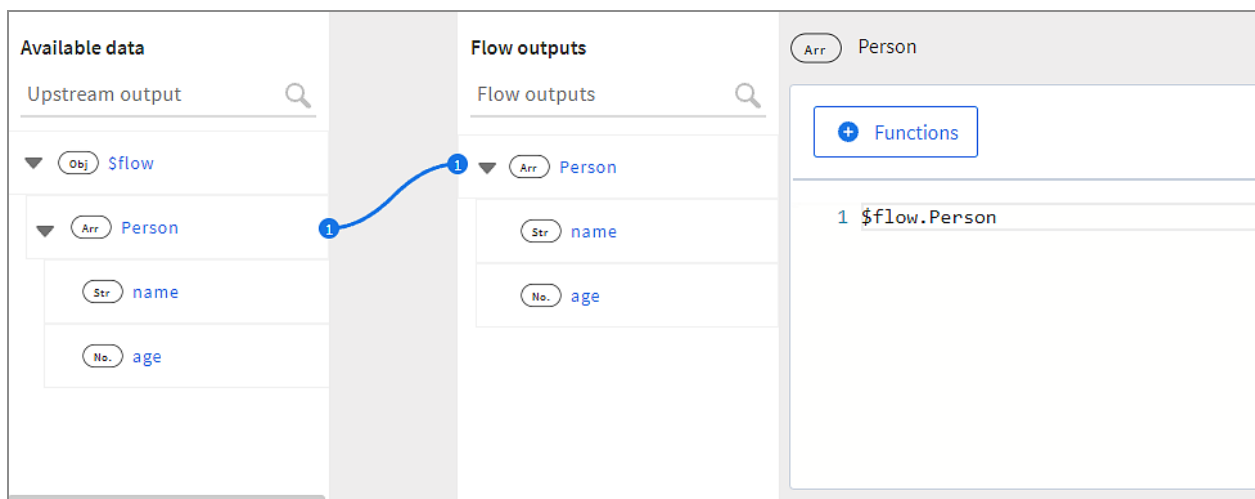


i Note: When user drags a non-string field to a string type of input in the data mapper, then the mapped variable is wrapped with `coerce to string` function implicitly.

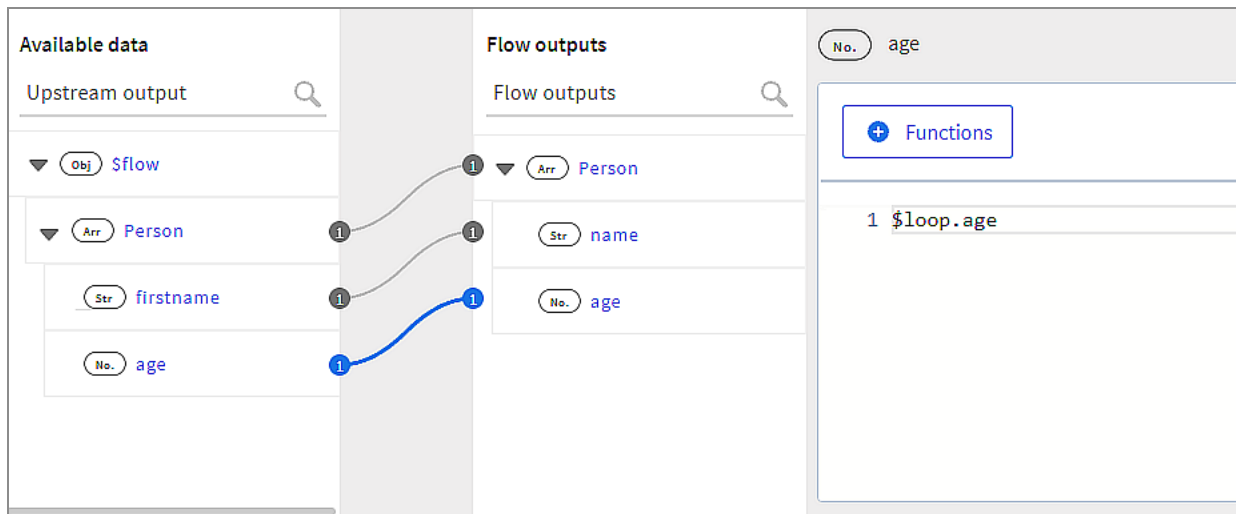
Mapping an Object

Standalone objects (objects not within an array) whose property data types match can be mapped at the root level. If the destination object is identical to the source object under **Available data** (both the names of the properties as well as their data types match exactly), you need not match the elements in the object individually. If the property names are not identical, then you must map each property individually within the object.

For example, in the image below the **Person** objects are identical. So, you can map **Person** to **Person**. You do not need to map the name and age individually.



In the following image, the data types match but the property names do not match. In such a case, you must map each property individually in addition to mapping the object root.



Mapping Arrays

When mapping arrays, you must first map their array root before you can map their child elements.

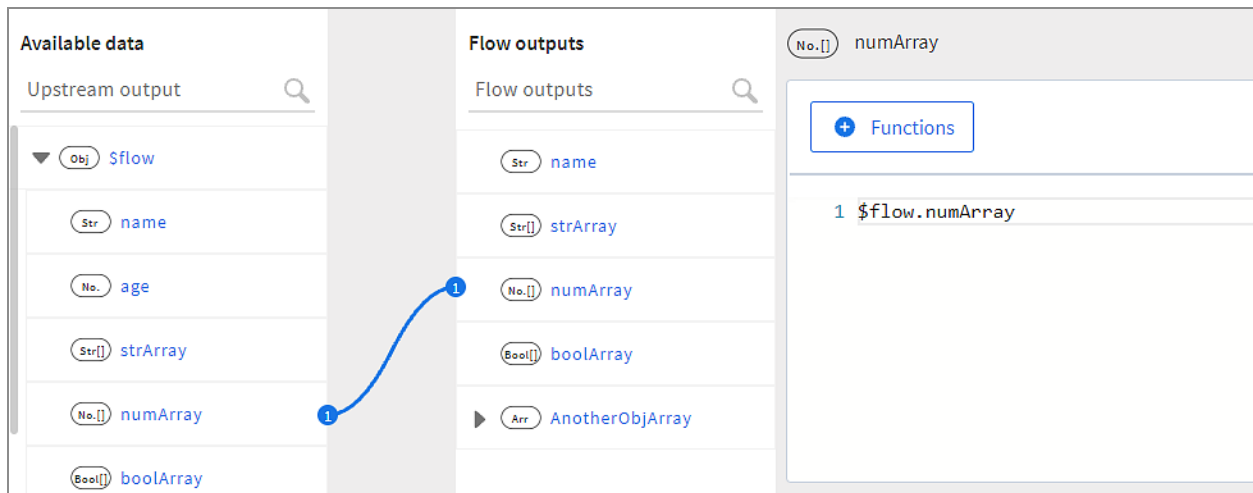
The following mappings are supported when mapping arrays.

- Mapping arrays of primitive data types
- Mapping an array of objects
- Mapping nested arrays

Mapping an Array of Primitive Data Types

To map arrays of the same primitive data type, you only need to map the array root. You need not map the array elements.

Here is an example of mapping arrays of primitive data types:



The array names need not to match, but their data types must match. In **Available data**, \$flow points to **numArray** which is the scope for **numArray** in the input.

When you do not have a matching data type array in your output

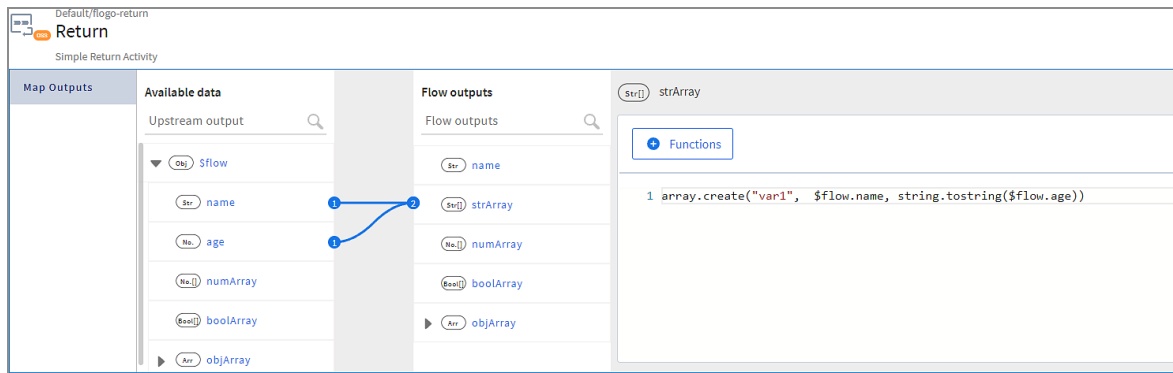
If you want to map an array of primitive data types, but you do not have an array of the same data type in **Available data**, you can create an array using the `array.create(item)` function.

Note: `array.create(item)` can only be used to create an array of primitive data types. You cannot use it to create an array of objects.

To do so:

1. Click the array for which you want to do the mapping in the input schema. The mapper opens to its right.
2. Click **Functions** and click **array** to expand it.
3. Click `create(item)`. It appears in the text editor.
4. Replace `item` with the output element to create the array.

In the following image, to map **strArray**, you would need to create an array since there is no array of strings under **Available data**. So, you map **strArray** by creating an array. The `array.create()` function accepts any of the following: a hardcoded string, an element from **Available data**, an expression, or a function as shown below as long as they all evaluate to the appropriate data type.



Mapping Complex Arrays

Complex arrays are arrays of objects that can optionally contain nested arrays. You can map these arrays using the three available options - **Configure with Items**, **Configure with Source** and **Configure with JSON**.

For examples, see <https://github.com/TIBCOSoftware/tci-flogo/tree/master/samples/app-dev/Mapping-Arrays/array.forEach.sample>.

When you use the **Configure with Items** option, you define an implicit scope consisting of everything available in **Available data**. It is equivalent to creating an implicit array with a single object element consisting of everything in **Available data**. Hence, the resulting length of the array is always one element.

To create a confined scope within **Available data**, use the **Configure with Source** option. When using this option, you must map three fields - **Source**, **Loop name** and the **Filter by**. Here, **Loop name** gets auto populated. When mapping identical arrays, the source name gets inserted in the **Select Source** field by default.

The **Source** defines the scope within **Available data**. Simply put, the input object or array can only be mapped to elements in **Available data** that fall within the boundary indicated by its scope.

The **Loop name** is a scoping variable given to the scope that you have defined in the first argument. By default, the scoping variable name is the same as the input element name for which you are defining the scope. By doing so, the mapper associates the input object to its scope by the scoping variable. Once there is a scoping variable for the scope, the mapper uses that scoping variable to refer to the scope in future mappings. You can edit the scoping variable to any string that might be more meaningful to you. The scoping variable is particularly useful when mapping the child elements in nested arrays.

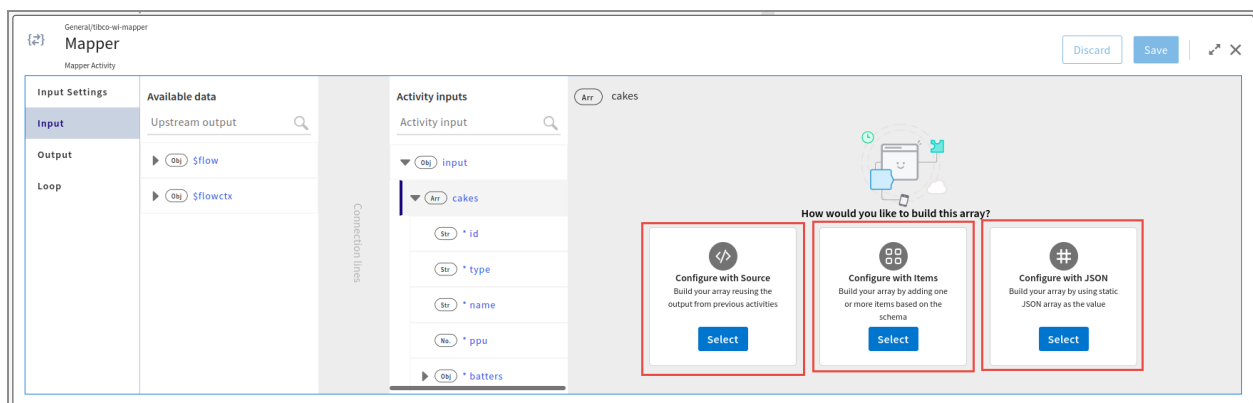
The **Filter by** field is optional. When iterating through an upstream output array, you can enter a filter to specify a particular condition for mapping as the **Filter by** field. When using this field, you must enter the scoping variable in the loop name field. Only array elements that match the filter get mapped. For instance, if you are iterating through an array (array1) in the upstream output with a filter `$loop.name=="Jane"` mapped in the **Filter by** field, if array1 has 10 elements and only four out of them match the condition of the filter, only those four elements are mapped to the input array and the remaining six are skipped. This results in the size of the input array being only four elements, even though array1 has 10 elements. See [Filtering Array Elements to Map Based On a Condition](#) for more details.

Note: If you have used the `array.forEach()` in a legacy app, to update your app with the current changes, delete the old mapping and remap the elements. A scoping variable is now included in the Loop name field. For example, if the old mapping is: `array.forEach($flow.body.Book)`, after remapping, `$flow.body.Book` is added to the Source field, where "Book" is added in the Loop name field, which is also the scoping variable.

Note: If you use a function as a source array in the source field, the array element schema cannot be determined and a design-time validation error is returned. It is recommended that you use the mapper to define the function output schema and then use it in the source field.

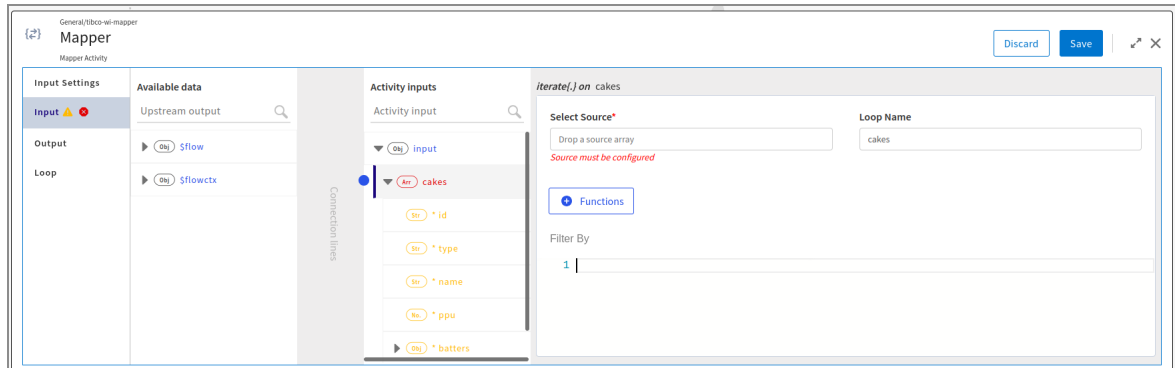
Mapping of Unmapped Arrays

With the support of first class `for.each()` in the Mapper activity, you can map elements to an unmapped array in three different ways.



- Using the **Configure with Source** option

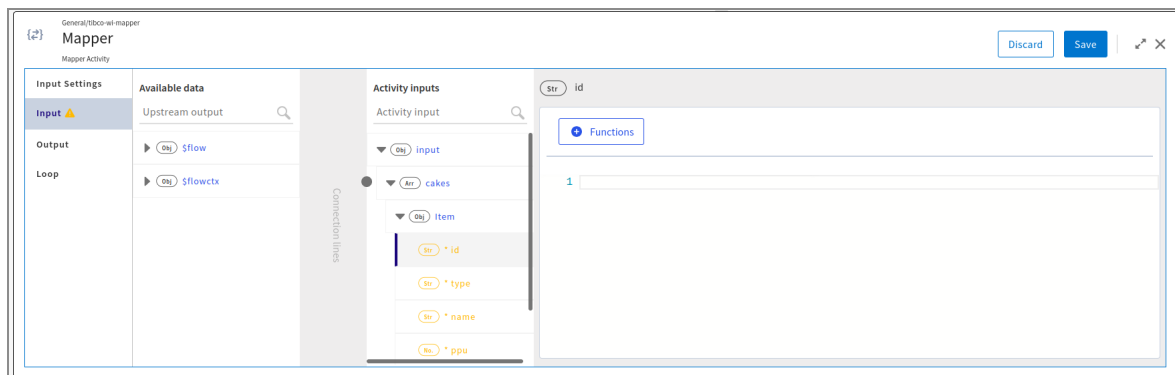
For mapping, double-click the element from the **Available data** array. You can also drag the element of the **Available data** array to the element of the **Activity inputs** array.



Note: To change the element that is already mapped, either drag another element or select the element from the source array.

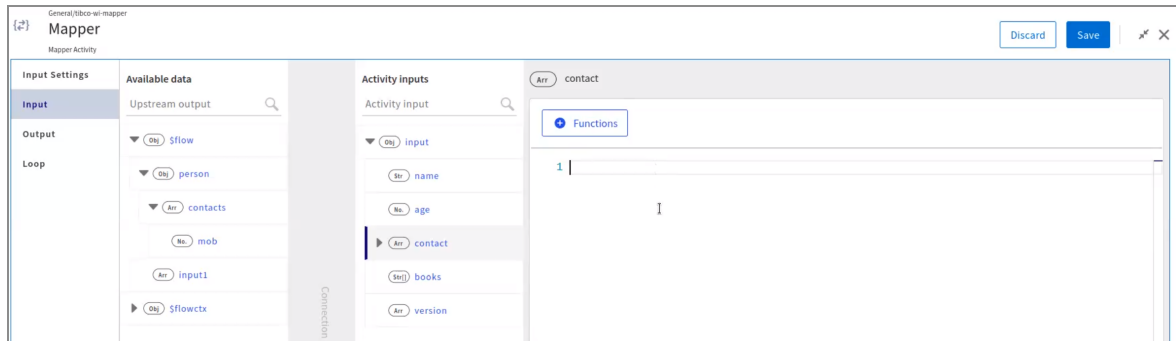
- Using the **Configure with Items** option

You can add the elements to your array manually.



- Using the **Configure with JSON** option

You can map the empty array by literal value mapping or type in the required expression.

**Note:**

- **Reset** option allows you to delete all the items from the array and set the array to the default form.
- **Clear mappings** can be used to remove all the mappings on the item level.
- For an empty `for.each()` array, you can clear the mappings for child items only.

Add Items to Array

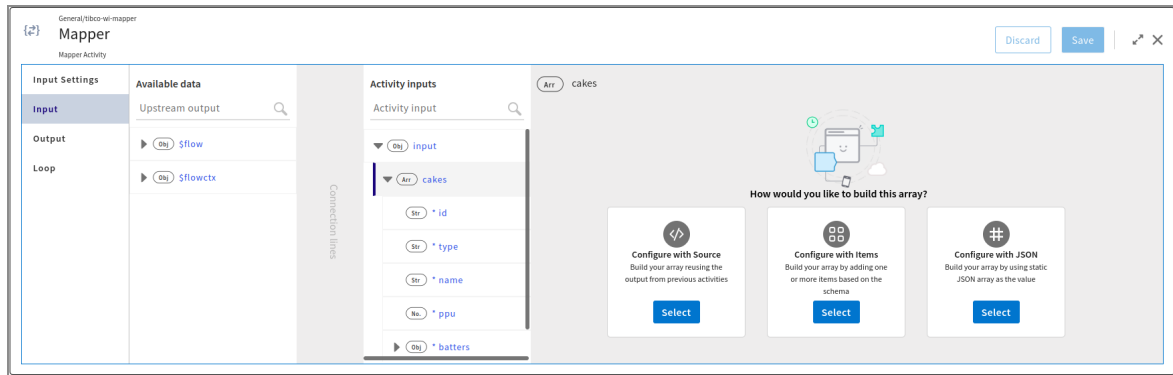
Now, when there is a need to map more than one array object in the same array, you can add items to the array. Each item can be mapped with different values.

For example, if one item is mapped with the flow input, the other can be mapped with literal values.

You can add an item to the array:

1. For an unmapped array

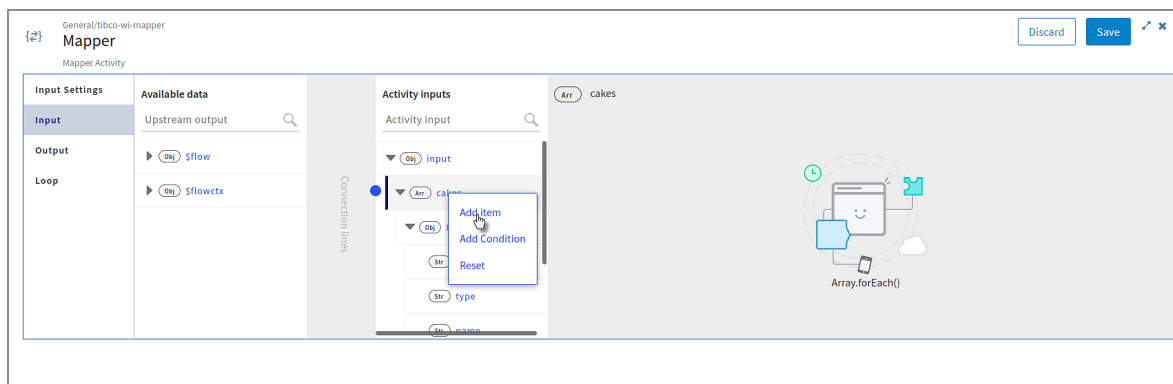
You can add an item in an unmapped array.



Note: Use the **Configure with Items** option for adding a single item.

2. For empty for.each() array

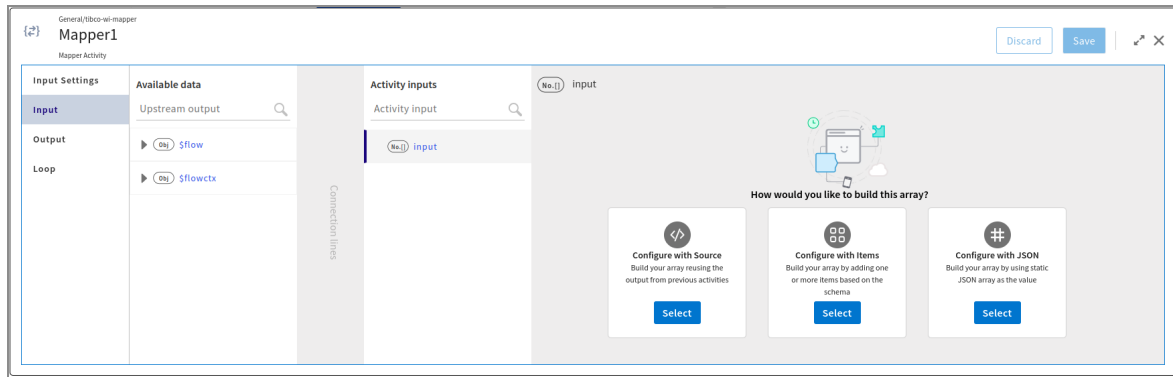
You can also add an item to an empty For.each() array.



- Note:**
- For all pre-existing array mappings with empty array.foreach() the properties are displayed as an array item and the array level mapping is not editable.
 - On adding empty array.foreach(), the input mapper at array level turns to non-editable.
 - Elements under an existing array mapping that has array.foreach() without source are wrapped in an item object.

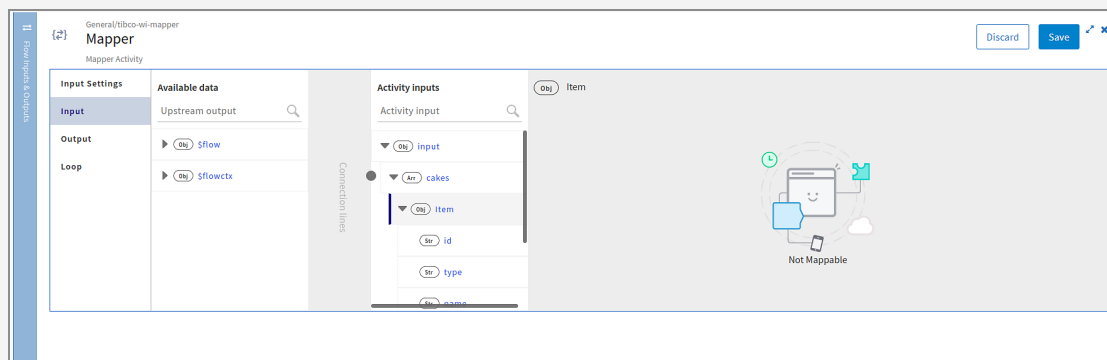
3. Primitive data type array

An item can even be added to an empty primitive data type array.



Note:

- **Add item** option is not available for an array of type 'any'.
- On copying an app to a workspace, which has inline array mapping, array elements are wrapped in an item object.
- On adding items under an array, mapping cannot be done at the item-level



Loop Variables

Loop variable is an automatically generated variable that represents the current item being processed in each iteration of a loop configured on an activity. When you set up an activity to iterate over an array of data, the Loop variable provides you with direct access to that data element for the current loop cycle.

Loop variables are designed to simplify and clarify data mapping within iterative processes. They work as temporary placeholders that focus on one item at a time, ensuring you can work with the item properties.

Loop variables have the following primary functions:

- **Mechanism:** When you configure an Iterate loop on an activity for Array Objects, such as in a Mapper activity, and select a source array, a default, editable Loop Name, such as `product`, is generated.
This Loop Name links to the corresponding Loop variable in the Available data pane. For example, if your loop is named `product`, the variable is `$loop[product]`.
 - You can drag a property from `$loop[product]` (for example, `$loop[product].name`) to a target field in your output schema. Alternatively, you can directly access these properties using dot notation, such as `$loop[product].name`, `$loop[product].price`, `$loop[product].quantity`
- **Expression Builder Integration:** You can use loop variables to construct complex expressions for calculations, concatenations, conditional logic, and function calls that operate on the data of the current item.
 - For example, to calculate `totalPrice` for each item, you can construct the following expression: `$loop[product].price * $loop[product].quantity`

Important Considerations

- **Primitive arrays:** Loop variables are primarily designed for arrays of objects. For arrays of primitive types, such as strings, numbers, and booleans, the Loop variable might not work, as there are no sub-properties to expose.
- **Loop Name updates:** If you rename the Loop Name after the configuration, existing mappings using the old Loop Name break and require manual re-mapping.
- **Nested loops:** Loop Variables are generated for nested loops, such as `$loop[parentLoopName]` and `$loop[childLoopName]`, allowing access to both the outer and inner contexts.
- **Clearing mappings:** If you clear the mappings involving loop variables, the associated mapping lines disappear and the loop variable is removed from the Available data pane.

Mapping Identical Arrays of Objects

When mapping an array of objects in the input to an identical array of objects (matching property names and data types) in **Available data**, keep the following in mind:

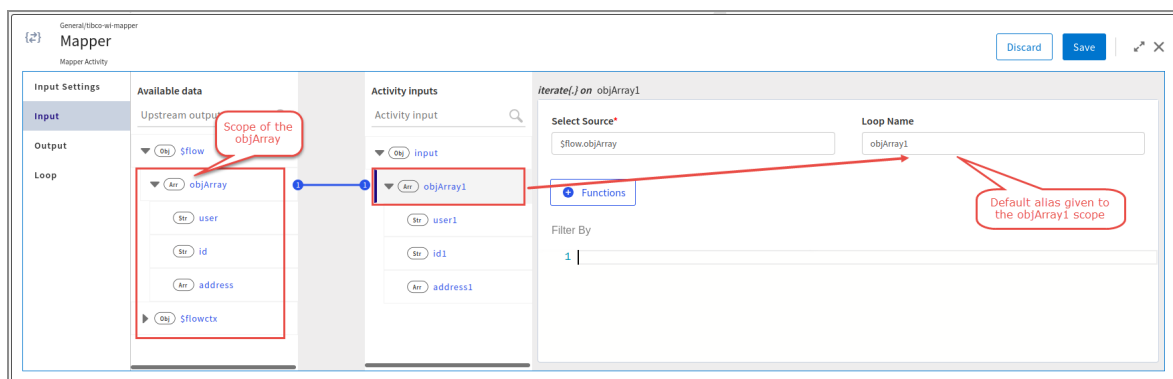
- Map the array at the root level by either dragging or double-clicking the **Available Source** array. The **Configure with Source** screen displays the array scope and the

scoping variable. You need not map the array object properties individually if you want all properties to be mapped and if the object property names are identical. The properties are automatically mapped.

- If you do not want all the properties within the object to be mapped or if the names of object properties do not match, you must map the object properties individually too after mapping the root. If you do not do the child mapping individually, the mismatched properties in the objects remain unmapped if the properties are not marked as required (marked with a red asterisk). If such a property is marked as required, then you see a warning.
- The size of the input array is determined by the size of the array in **Available data** to which you are mapping.

To map identical arrays of objects:

- Drag the array that you want to map from **Available data** (**objArray** in the image below) and drop it on the array in the **Flow outputs** pane (**objArray1** in the image below). The **Configure with Source** screen appears in the text box. If the names of all the child elements match, the child elements get mapped automatically. You need not match each child element individually. In this example, none of the child names match, so you would need to do the individual mapping otherwise none of the elements get mapped.



The "**objArray1**" in the **Loop name** is the scoping variable that constitutes the scope of the current input array. Basically, this means that you can map any element in **objArray1** with an element of the same data type in **flow.objArray** in **Available data**. So, you are defining the scope of **objArray1** to be all the elements within **objArray**.

Mapping Array Child Elements to Non-Array Elements or to an Element in a Non-Matching Array

There may be situations when you want to map an element within an array of objects to an output element that is not in an array or belongs to a non-matching array in the **Available data** pane. In such a situation, you must create an array with a single element. You do this by using the **Configure with JSON** option. When you use this option, it creates an array with an item having a single object element. The single object element treats everything in **Available data** as the children of the newly created array object element. This allows you to map to any of the **Available data** elements as they are now treated as if they were within an array.

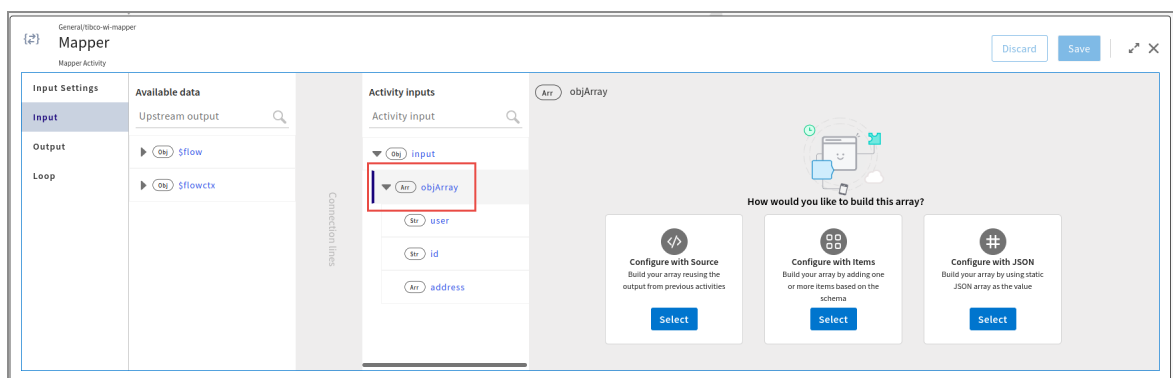
Important: When using the **Configure with Items** option be sure to map the child elements individually. Otherwise, no child elements get mapped. Only elements that you have specifically mapped acquire the mapped values.

Note: Keep in mind that in this scenario, the resulting length of the array is always one element.

Mapping an array child element to a non-array element is a two-step process:

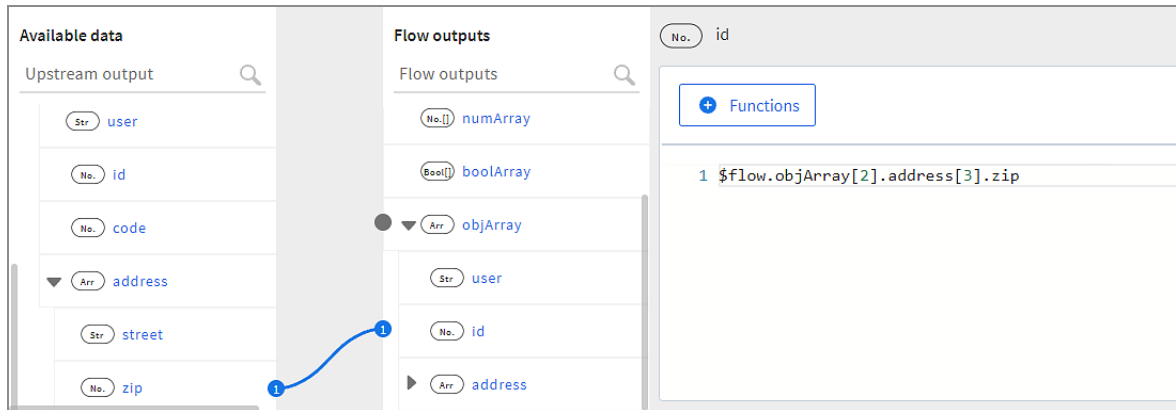
1. Click the input array root (**objArray** in the example below) and select the **Configure with Items** option.

This creates an array of objects with a single element in it. The element contains everything under **Available data**, hence allowing you to map to any element in the **Available data** pane. The element you are mapping to can be a non-array element or reside within a nested array.



2. Map each element in the input array individually to any element of the same data

type under **Available data**.

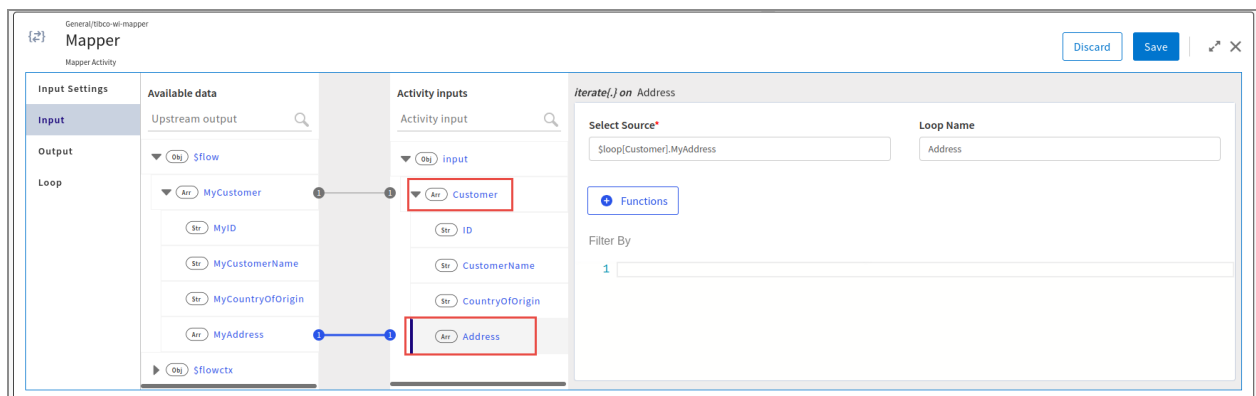


To map an element inside an array, provide the index of the array. To map an element in a nested array, provide the index for both the parent and the nested array as shown.

Mapping Nested Arrays

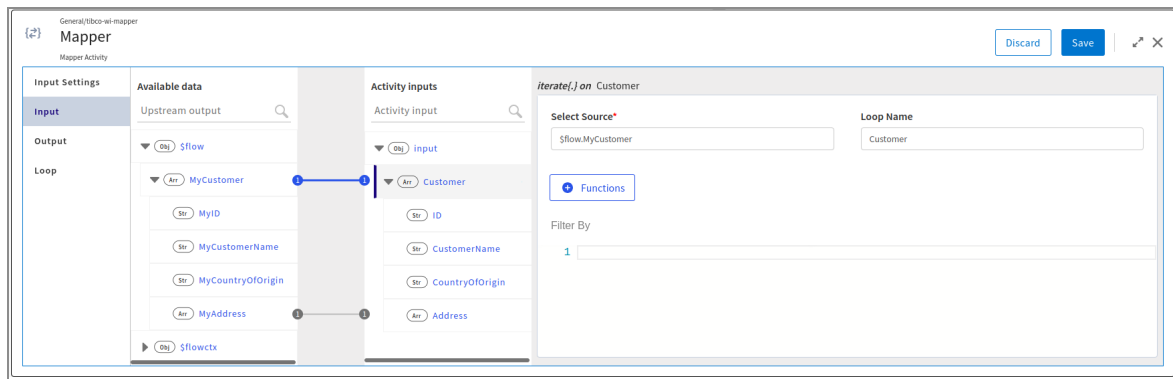
Before you map a nested array, you must map its parent root. The scoping variable is particularly useful when mapping the child elements in nested arrays.

The example below is that of a nested array, where **Address** is a nested array whose parent is **Customer**:



To map **Address**:

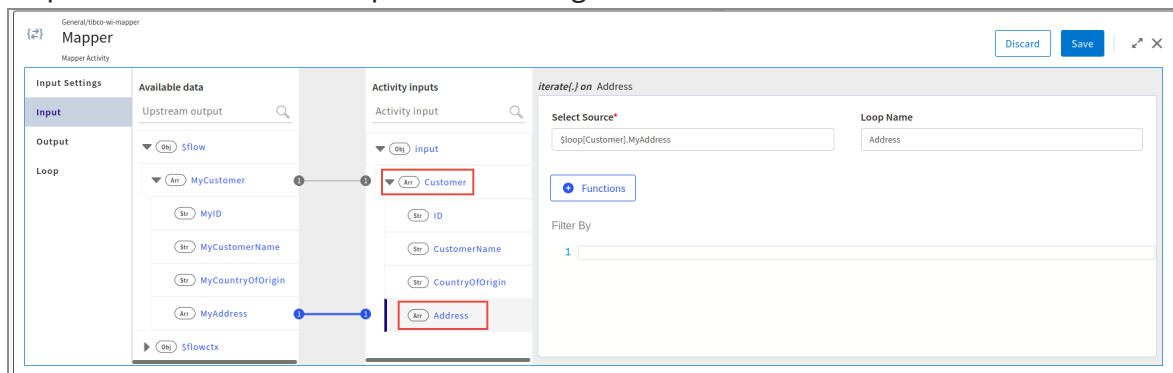
1. Map its parent, **Customer**. When you map **Customer**, you automatically set the scope of **Customer**.



In the image, **Customer** is mapped to **MyCustomer**. In the **Select Source** field, the `$flow.MyCustomer` is the source array (from which **Customer** gets the data) that you are mapping to. This defines the scope (boundary) in **Available data** within which you can map **Customer**. So, this is the scope of **Customer**.

The **Loop name** field, "Customer", is the scoping variable given to this scope - the loop here refers to the iteration of **Customer**. By default, the scoping variable has the same name as the loop for which the scope is being defined (in this case **Customer**). You can edit the scoping variable to any string that might be more meaningful to you. This is equivalent to saying that mapping of a child element of **Customer** can happen only to children of **MyCustomer** in **Available data**.

2. Map **Address**. Now the scope of **Address** gets defined.



Notice the mapping for **Address**:

- contains the parent scope as well. The parent scope is referred to by its scoping variable, "Customer". Remember that the scope of **Customer** is already set when you are mapping **Customer** to **MyCustomer** in the first step, so we can now simply refer to the parent scope by its scoping variable, "Customer".

- `$loop[Customer]` refers to the iteration of the **MyCustomer** array. `$loop` represents the memory address of the **MyCustomer** (the scope for **Customer**) in **Available data**.
- `$loop[Customer].MyAddress1` is the scope of **Address**. This scope is denoted by the scoping variable "Address", which is the second variable in this mapping. Since **Address** is a nested array of **Customer**, when you map to **Address** or its child elements, its mapping includes the scope of **Customer** as well.

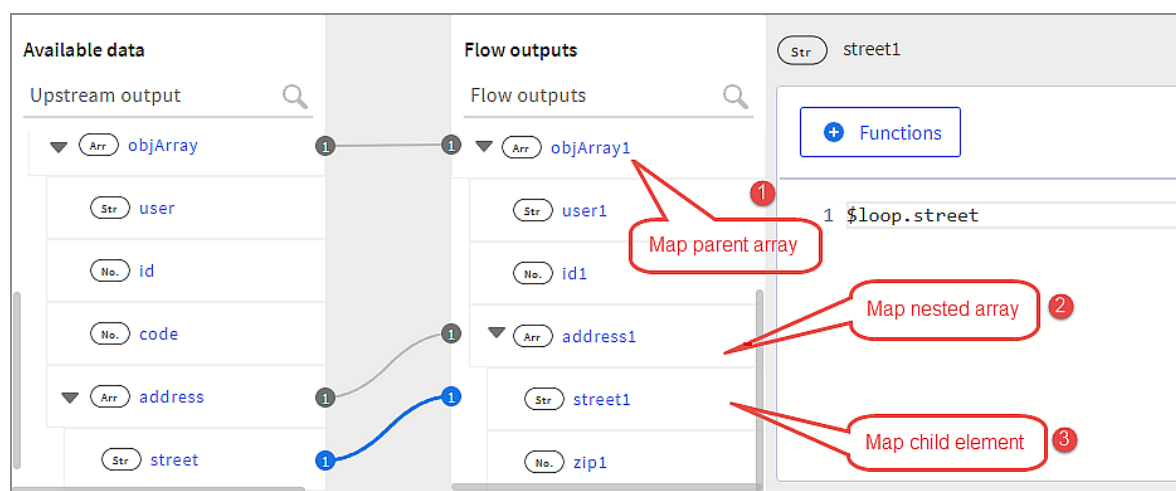
Mapping Child Elements within a Nested Array Scope

A child element in the input array can be directly mapped to a child element of the same data type within the array scope. As mapping is done within the nested array scope, you need not explicitly state the scoping variable for the nested array scope. The mapping appears as `$loop.<element>`.

To map a nested array child element:

1. Map the parent of the nested array.
2. Map the nested array itself.
3. Map the nested array child elements if the names are not identical or if you do not want to map all elements in the nested array.

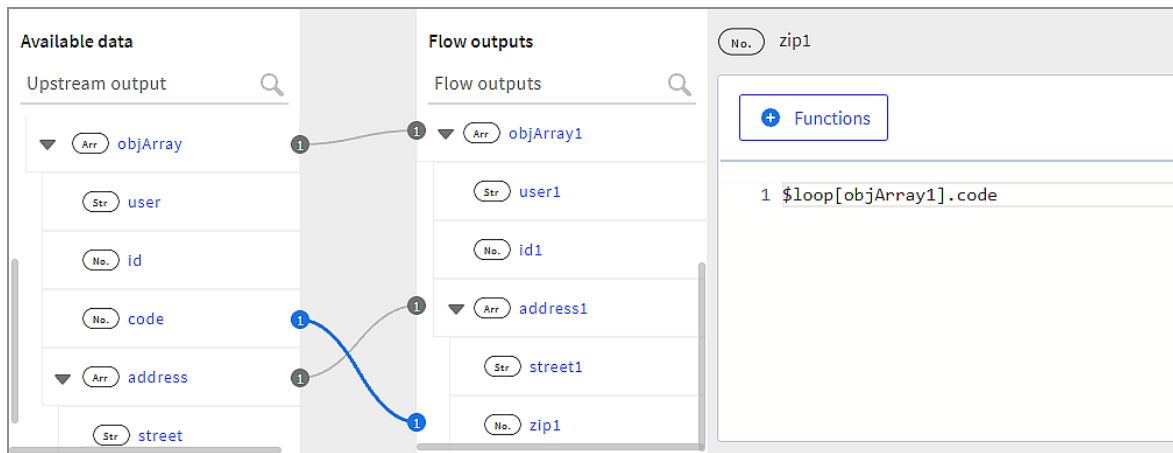
In the following example, since **street** is within the scope of **address1**, **street1** is directly mapped to **street**. `$loop` implicitly points to **address** which is the scope for **address1** in the input schema.



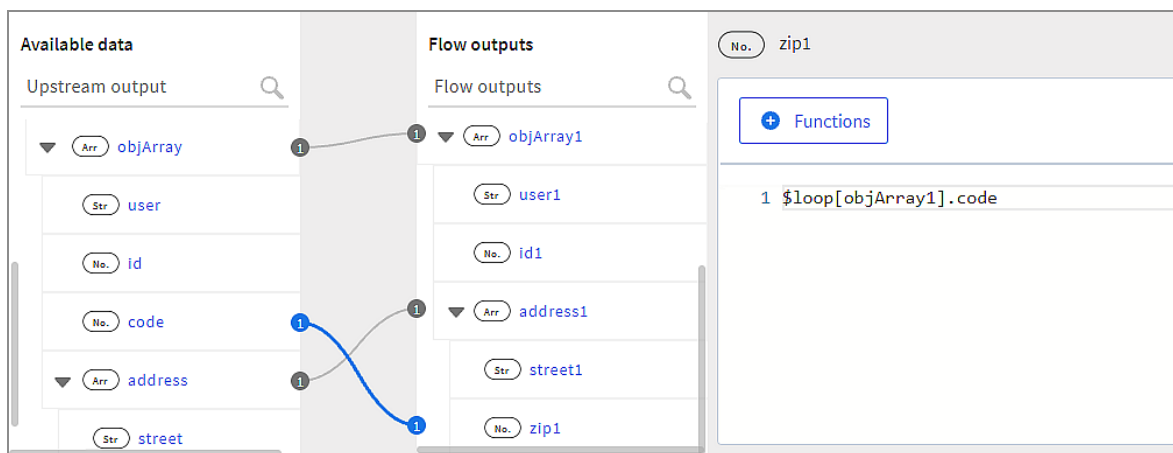
Mapping a Nested Array Child Element Outside the Nested Array Scope

To map a nested array child element outside the nested array scope but within its parent array, you must use the scoping variable of the parent array.

1. Map the parent array root.
2. Map the nested array root.
3. Map the nested array child element.



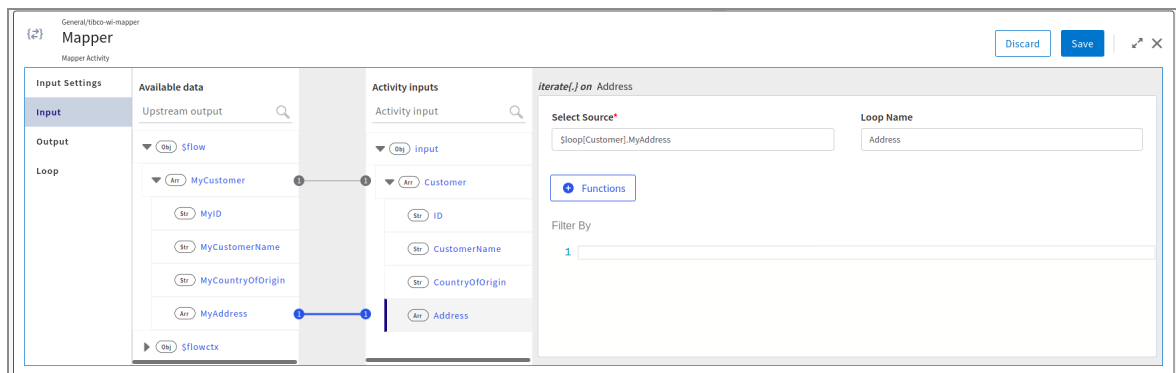
In the image below, `$loop` implicitly points to **address**. In addition, the mapping also explicitly specifies the scope of the parent, "objArray1. This is because **zip1** is mapped to **code** which is outside the scope of **address1**, but within the scope of its parent array (**objArray1**).



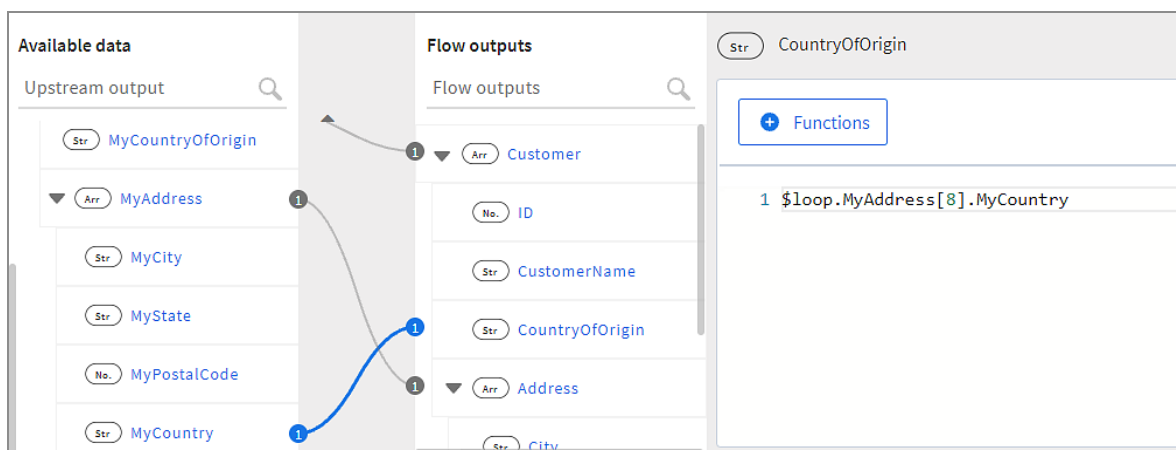
Mapping an Element from a Parent Array to a Child Element in a Nested Array within the Parent

When mapping a primitive data type child element of the parent array to a child element of its nested array, the scope in the mapping is implicitly set to the scope of the parent array. In addition, you must provide the index of the nested array element whose variable you want to map to.

1. Map the parent array root.
2. Map the nested array root.
3. Map the parent array element.



In this example, `$loop` is implicitly set to the scope of **Customer**, which is **MyCustomer**. Notice that you must provide the index of the object in the **MyAddress** array whose **MyCountry** element you want to map to.



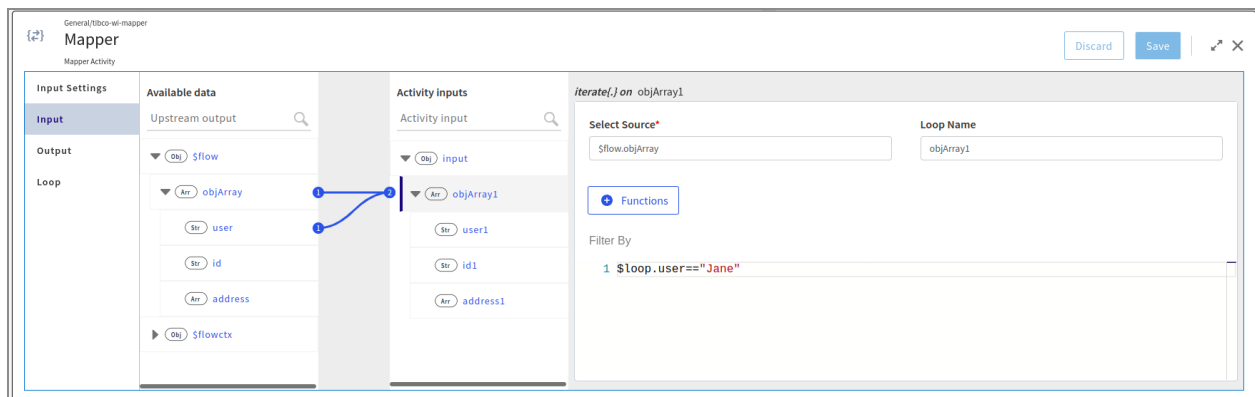
Filtering Array Elements to Map Based On a Condition

When mapping arrays of objects, you can filter the objects that are mapped by specifying a filter in the **Filter by** field when the **Configure with Source** option is selected.

Specify the filter in the **Filter by** field. The **Select Source** value is the scope of the element that is mapped and the **Loop name** is the scoping variable.

To add the filter in the **Filter by** field, the **Source** and the **Loop name** must be specified.

Here is an example that contains a filter in the **Filter by** field:



The above example indicates the following:

- **objArray1** is being mapped to **objArray** in **Available data**
- When iterating through **objArray** in **Available data**, only the array elements in **objArray** whose child element **user** is "Jane" get mapped. If **user** is not equal to "Jane" the iteration for that object is skipped and **objArray1** does not acquire that object.
- **\$loop** here specifies the scope of the current loop that is being iterated, in this case **objArray**, whose scope is **objArray1** in **Available data**.

Mapping JSON Data with the json.path() Function

Use the `json.path()` function to query an element within JSON data. The JSON data being queried can come from the output of an Activity or trigger. In the mapper, you can use the `json.path()` function by itself when providing a value to an input parameter or use it within expressions to refer to data within a JSON structure.

This function takes two arguments:

- the search path to the element within the JSON data
- the JSON object that contains the JSON data you are searching

You can specify a filter to be used by the `json.path()` function to narrow down the results returned by the `json.path()` function.

To reach the desired node or a specific field in the node in the JSON data, you must follow a specific notation defined in the JsonPath specification. For more information on the notation to be used and specific examples of using the notation, see <https://github.com/oliveagle/jsonpath>.

Consider the example below which is available for you to experiment with at <https://github.com/TIBCOSoftware/tci-flogo/tree/master/samples/app-dev/json.path.sample>.

Examples

The following is an example of how to use the function:

```
json.path("$.store.book[?(@.price > 10)].title", $flow.body)
```

In this example, `$.store.book[?(@.price > 10)].title` is the query path. `[?(@.price > 10)]` is a filter used to narrow down the query results. `$flow.body` is the JSON object against which the query is run (in this case the JSON object comes from the flow input, hence `$flow`). So, this query searches the books array within the `$flow.body` JSON object and returns the title of the books whose price is more than \$10.

Consider the following sample JSON data:



Caution: Code snippets in the PDF could have undesired line breaks due to space constraints and should be verified before directly copying and running it in your program.

```
{
  "store": {
    "book": [
      {
        "category": "reference",
        "author": "Nigel Rees",
        "title": "Sayings of the Century",
```



```
"Availability": [
  {
    "Country": "India",
    "Quantity": 4000,
    "Address": [
      {
        "city": "houston"
      }
    ]
  }
],
"price": 8.95
},
{
  "category": "fiction2",
  "author": "Evelyn Waugh",
  "title": "Sword of Honour",
  "Availability": [
    {
      "Country": "USA",
      "Quantity": 5000,
      "Address": [
        {
          "city": "sugarland"
        }
      ]
    }
  ]
},
"price": 12.99
},
{
  "category": "fiction3",
  "author": "Herman Melville",
  "title": "Moby Dick",
  "isbn": "0-553-21311-3",
  "Availability": [
    {
      "Country": "UK",
      "Quantity": 7000,
      "Address": [
        {
          "city": "stafford"
        }
      ]
    }
  ]
},
],
```

```

    "price": 8.99
  },
  {
    "category": "fiction4",
    "author": "J. R. R. Tolkien",
    "title": "The Lord of the Rings",
    "isbn": "0-395-19395-8",
    "Availability": [
      {
        "Country": "Australia",
        "Quantity": 2000,
        "Address": [
          {
            "city": "aaaaa"
          }
        ]
      }
    ]
  },
  {
    "price": 22.99
  }
],
"bicycle": {
  "color": "red",
  "price": 19.95
},
"expensive": 10
}

```

The following are examples of some JSON query paths that search the JSON data above and return the category of the book. In the examples below, the second input parameter for this function (data) is the name of the file that contains the above JSON code.

- `json.path("$.store.book[?(@.Availability[?(@.Quantity >= 6000)])].category", $flow.data)`
In the example above, the query scope is the entire book array. The filter used to query this array is the condition - `[(@.Availability[?(@.Quantity >= 6000)])]`. Only the category values for the book elements that have Quantity >= 6000 is returned. So, this query returns fiction3.
- `json.path("$.store.book[?(@.author == 'Nigel Rees')].category", $flow.data)`
returns reference since it uses the filter `[(@.author == 'Nigel Rees')]` and the only book authored by Nigel Rees in this array of books has its category as reference.
- `json.path("$.store.book[?(@.Availability[?(@.Address[?(@.city == 'sugarland')])]).category", $flow.data)`

This query is an example of a nested filter where `[?(@.Availability[?(@.Address[?(@.city == 'sugarland')])])]` is the outer filter and the nested filter within it is `[?(@.city == 'sugarland')]`. It returns reference.

- `json.path("$.store.book[0].category", $flow.data)`

This query does not use a filter. It returns reference, since your query scope is limited to the `book[0]` element only within the store object and your request is to return the value of category.

Constructing any, param, or object Data Type in Mapper

When mapping values for data type any or object, you must manually enter the values in the mapper text box.

Below are some examples of how to construct the data type any:

Assigning a literal value to a data type any

To assign literal values to the any data type, you click the element of type any, then simply enter the values you want to assign to it in the mapper text box. For example, to assign the string Hello! enter:

```
"Hello!"
```

Assigning an object value to an object or element of a data type and

Here is an example of how to assign literal values to an object:

```
{
  "Author": "Martin Fowler",
  "ISBN": "0-321-12742-0",
  "Price": "$45"
}
```

Here, "Author", "ISBN", and "Price" are the object properties. You can use a function instead of a literal value when assigning values for each element. See the "Using a function" section for details on how to use a function.

Assigning an array value to an object or data type any

Here is an example of how to assign an array value to an array of objects or to an element of the data type any:

```
[
  {
    "Author": "Martin Fowler",
    "ISBN": "0-321-12742-0",
    "Price": "$45"
  }
]
```

You can use a function instead of a literal value when assigning values for each element. See the "Using a function" section for details on how to use a function.

Assigning a value from the upstream output

When mapping to an element from the upstream output, the data type of the source element whose value you are assigning determines the data type of the destination element. For example, if you assign the value of an array, then the target element (the element of the data type any) is treated as an array, likewise for a string, number, boolean, or object. For example, if you are mapping `$flow.Author` which is an array, then the `Author` object in the input (destination object) would also be an array. That is, there is a direct assignment from the source to the destination.

- **Single Element of Primitive Data Type:** To assign the value of a single element of a primitive data type that belongs to the output of the trigger, a preceding Activity, or the flow input, you must enter the expression for it. For example, to assign the value of `isbn` which comes from the flow input, enter the expression:

```
"=$flow.isbn"
```

Here `$flow` is the scope within which `isbn` falls.

- **An object:** When assigning an object, you must create a mapping node within the object. The mapping node is used to define how the object should be constructed and the various fields within the object mapped. For example, to assign the `bookDetails` object, enter:

```
{
```

```
"mapping": {
  "Author": "=$flow.author",
  "ISBN": "=$flow.name",
  "Price": 20,
  "BestSeller": true
}
```

You can use a function instead of a literal value when assigning values for each element. See the "Using a function" section for details on how to use a function.

- **An array of objects:** The following two examples show you how to assign values to arrays:

- **Building a new array**

To provide values for an array that has a fixed size (where the number of elements is declared), you must provide the values for each array element. For example, if the array has two elements, you must provide the values for each property of the object for both objects. Here is an example of how to do that:

```
{
  "mapping": {
    "books": [
      {
        "author": "=$loop.author",
        "title": "=$loop.title",
        "price": "=$loop.price"
      },
      {
        "author": "Author2",
        "title": "BookTitle",
        "price": 19.8
      }
    ]
  }
}
```

In the example above books is an array of two elements. The values for each property for both elements are provided.

You can use a function instead of a literal value when assigning values for each element. For details, see [Using Functions](#).

- **Building an Array from an upstream output array**

In the following example, books is an array of books coming from the upstream output. To iterate over the array, \$flow.store.books in upstream output, and assign its values to the input array, you would enter the following in the mapper text box:

```
{
  "mapping": {
    "@foreach($flow.store.books)": {
      "author": "=$loop.author",
      "title": "=$loop.title",
      "price": "=$loop.price"
    }
  }
}
```

The "@foreach(\$flow.store.books)" indicates that you are iterating an array of objects where the \$flow.store.books is the array. \$flow is the scope within which store.books falls and \$loop represents the scope for each property within the object.

- **Using a function:** The following example uses the output of a REST Invoke Activity to get a pet from the public petstore service. The mapper uses the string.concat() function and assigns the function return value to the description field in the data structure:

```
{
  "mapping": {
    "data.description": "string.concat(\"The pet category name is: \", $Activity[rest_3].result.category.name)"
  }
}
```

Assigning Values to the param Data Type

When you copy an app to a workspace that was originally created in TIBCO Flogo® Enterprise or TIBCO Cloud™ Integration, the app could contain elements that are of data type param. The param data type is similar to the object data type in that it consists of key-value pairs. The difference between an object and a param is that the object can contain values of any data type whereas the values for elements in the param data type *must* be of data type string only.

Here is an example of assigning values to a param data type element:

```
{
  "mapping": {
    "Author": "=$flow.author",
    "ISBN": "=$flow.name",
    "Price": "$20"
  }
}
```

Coercing of Activity Input, Output, and Trigger Reply Fields

In the OSS marked Activity input, output, or trigger reply configuration, if you have defined a parameter, but have not defined or cannot define a schema for the parameter, you can coerce the parameter to take the value from a schema that you dynamically define during design time. This feature is particularly useful for apps, which have activities for which input parameters or output are not defined with a schema.

Currently, coercion of parameters is supported only for the following data types:

- array
- object
- param
- any

After you enter the schema, it is displayed in a tree format under **Activity inputs**, **Output** tab, or **Trigger reply** in the mapper. All subsequent activities also display the elements of the schema under the Activity in the Upstream Output. The schema elements are now available for you to map.

Important Considerations

- Coercion is supported only in the **Default** category activities that are the activities marked as OSS, except for the **Return** and **Start a SubFlow** activities. These two activities display flow-level data. The flow-level inputs and outputs can be entered or modified only on the **Flow Inputs & Outputs** tab, hence they cannot be coerced from the **Input** tab of the activity itself.

- Currently, coercion is supported only for top-level parameters. Nested coercion (for example, an object within an object) is not supported.
- Currently, coercing a schema for trigger input is not supported. The coercing option is not available on the **Map to Flow Inputs** tab in the trigger configuration. This is because the parameters you see on this tab are flow input parameters and are not related to the trigger. You have the option to coerce these parameters on the **Input** tab of the **Flow Inputs & Outputs** tab.
- After you have mapped a child element within a parameter, if you change the name of the parent or the child, your mapping is lost. However, if you change the data type of the element, the mapping is preserved, but you see an error related to the mismatch in data type.
- The schema you enter is preserved when you export and import the app.
- If you edit the schema later, as long as you click **Apply** after editing, your edits are displayed in the mapper. You must then click **Save** in the mapper to persist your schema changes.
- You cannot coerce a parameter or edit its schema in any activity appearing in a subflow. For example, if the **OracleDatabaseQuery** activity appears in both the main flow and the subflow, you cannot edit the schema of any of its parameters in the subflow. But you can edit the schema of the **OracleDatabaseQuery** activity in the main flow. This is because the subflow activity input and output schemas are inherited from the main flow. There is a possibility that the same subflow could be used in multiple main flows, so if you edit an activity in the subflow it could break another main flow that uses the subflow.


To provide the schema for coercion:

Procedure

1. On the flow details page, click the activity or trigger to open its configuration.
2. Click any of the following tabs that you want to configure:
 - **Input:** To configure a parameter in the activity input
 - **Output:** To configure the schema for the activity output
 - **Map from Flow Outputs:** To configure the trigger reply
3. To configure a schema:
 - For a parameter in activity input, hover your mouse cursor over the parameter

name for which you want to configure the schema under **Activity inputs**.

- For the Activity output, hover your mouse cursor over the parameter name for which you want to configure the schema.
- For a parameter in the trigger reply, hover your mouse cursor over the parameter name.

Click the ellipsis icon () that appears next to it. **Clear mappings** and **Coerce with schema** options are displayed.

4. Click the **Coerce with schema** option.

i Note: The **Coerce with schema** icon appears against the parameter name for only those parameters that do not have a schema defined on the **Input Settings** tab (or a schema cannot be defined because the Activity does not have an **Input Settings** tab, for example, the OSS-marked activities) and whose data type is one of the following: array, param, object, or any.

5. Enter the schema for the parameter or activity output and click **Apply**. The mapper validates that the data type of the schema you entered matches the data type of the parameter being coerced. If the data types do not match, **Apply** remains disabled and you see an error. For activity input and trigger reply, the schema you enter displays in a tree format under the parameter name in the mapper.
 - For the activity output, the schema is displayed in a tree format on the **Output** tab of the activity. **Available data** displays the output of the preceding activities.
6. Click **Save** to persist the schema into the database or **Discard** to discard the schema. Now you can map the child elements within the parameter. In the case of the activity **Output** tab, the output tree does not display in the current activity but is displayed in the mapper for subsequent activities only. Once persisted in the database, these schema trees get displayed in the **Available data** area of the mapper for subsequent activities. This allows you to map to them in subsequent activities.

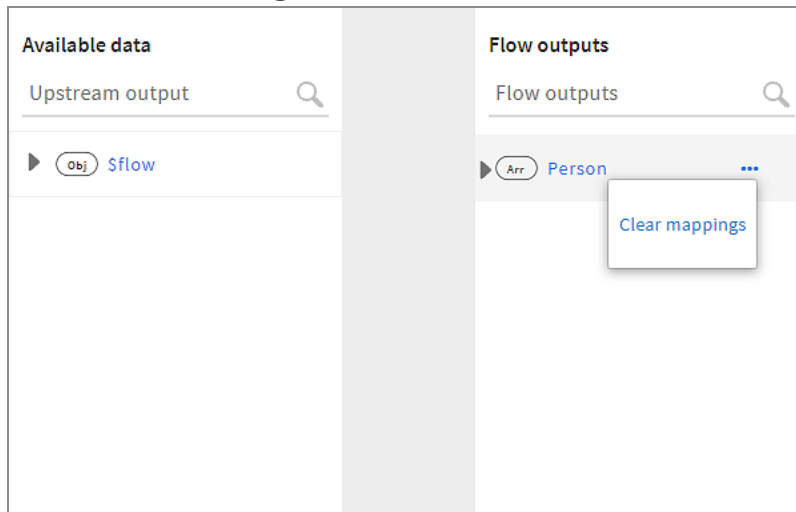
Clear Mapping of Child Elements in Objects and Arrays

After mapping an array or an object, you can clear the mapping of all the child elements within that array or object with one click. The mapping is cleared at the root level and the mapping for everything under that root gets cleared, even the nested arrays and objects,

should there be any. To clear the mapping for individual elements in an array or object selectively, click that element and delete the mapping for it.

To clear the mappings for all child elements of an array or object:

1. In the mapper, hover your mouse cursor to the right end of the root name until the ellipsis icon (...) appears and click it.
2. Click **Clear mappings**.



Ignoring Missing Object Properties when Mapping Objects

There may be instances when you map objects where one or more object properties might be missing in the source or target object. The mapper can be set to ignore such cases.

If you want the mapper to ignore such cases, you must set the `FLOGO_MAPPING_SKIP_MISSING` engine variable under the **Environment Controls** tab to true. The mapper ignores the missing mapping as long as the element is optional (not marked as mandatory with a red asterisk against it). Elements marked as mandatory must be mapped.

Mapping Data by Using if/else Conditions

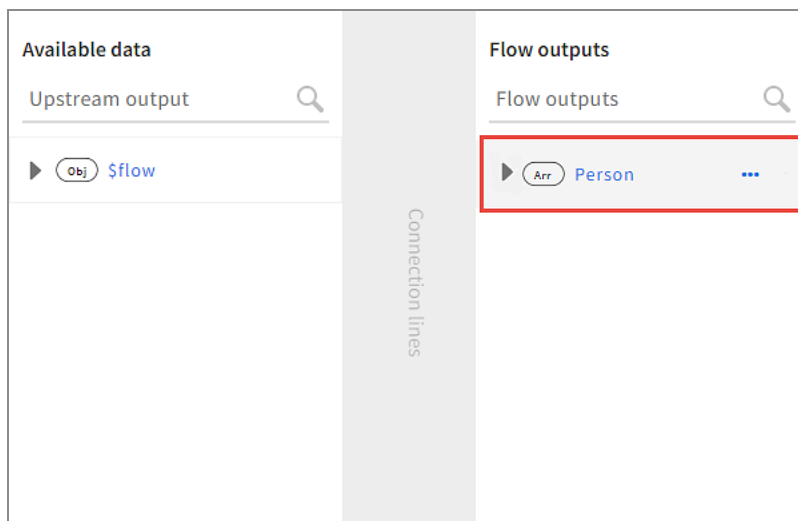
The if/else statements are used to execute blocks of code based on the specified conditions.

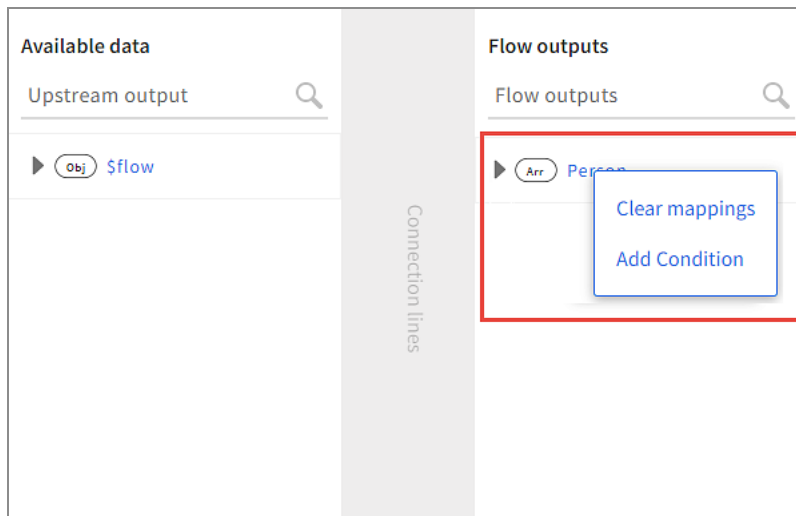
```
if (condition1)
{
  // execute this block of code
}
else if (condition2)
{
  // execute this block of code if the previous condition fails
}
else
{
  // execute this block of code if all conditions fail
}
```

You can add conditions in your data mappings to get outputs based on those conditions. You can add conditions to primitive objects, nested arrays, nested objects, and any other type of input. `if/else` conditions are available in activities and triggers in the main flow and error handler.

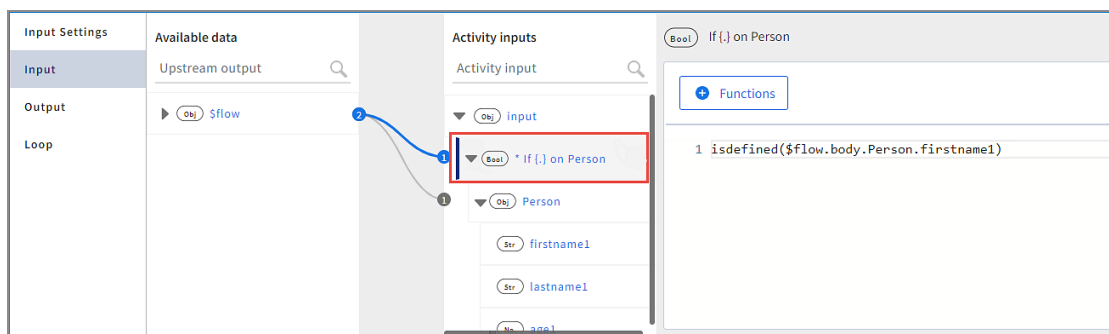
To Map Data Using Conditions

1. Click the ellipsis icon `...` to open the menu of the element to which you want to add the conditions. Select **Add Condition**. An If condition is added to the element.

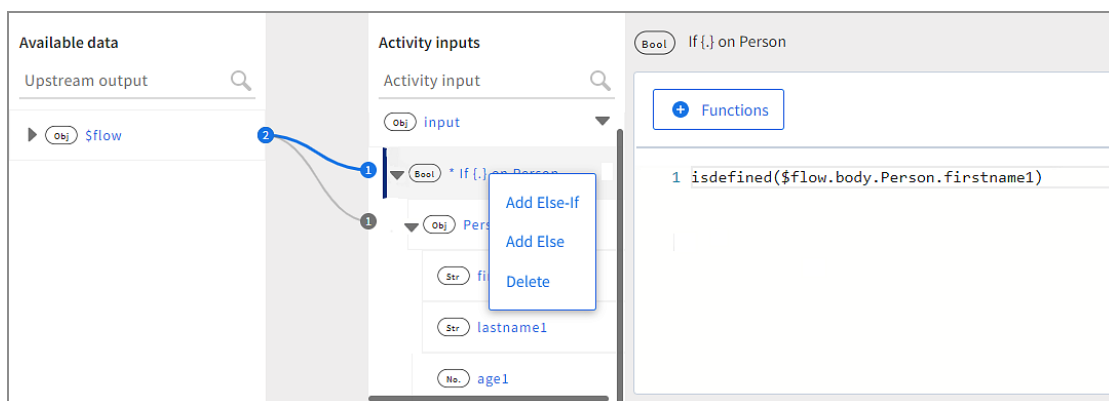




2. In the text editor of the If condition, enter an expression whose result evaluates to a Boolean value. You can enter the expression manually or map data from the **Available data** pane. If children elements exist, you can enter values for them.



3. To add an Else-if or an Else condition, click the ellipsis icon **...** to open the menu of the element with the If condition. Click **Add Else-If** or **Add Else**.



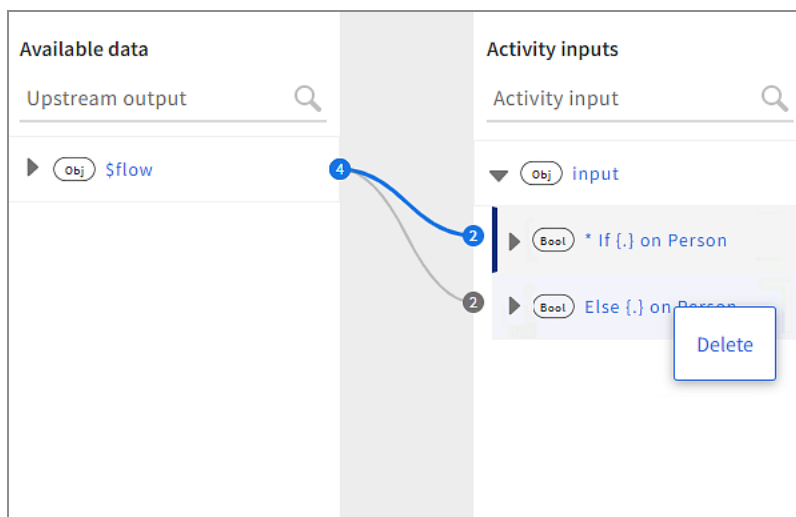
Considerations when using conditions:

- For one If condition, you can add multiple Else-if conditions and one Else condition.
- You can add an Else condition only from an element with an If condition.
- You can add an Else-if condition from an If condition and from an Else-if condition.

Note: In case the option to add conditions is not visible for the last element in the **Activity inputs** pane, scroll further down to view the options.

Deleting a Condition

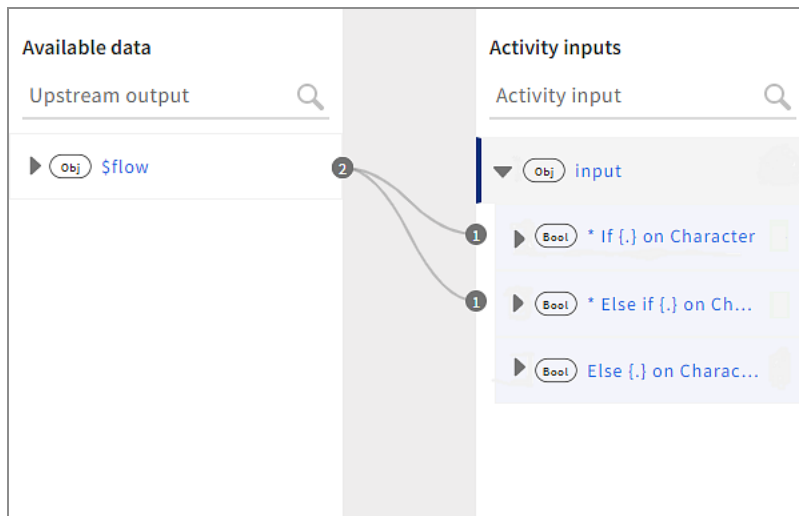
Click the ellipsis icon **⋮** to open the menu of the conditions that you want to delete. Click **Delete**.



To delete an If condition that has Else-if and Else conditions:

You cannot directly delete the If condition that has Else-if and Else conditions. You must first delete the Else-if and Else conditions to delete the parent If condition.

In the following example, to delete the If condition on **Character**, you must delete the Else-if and Else conditions.



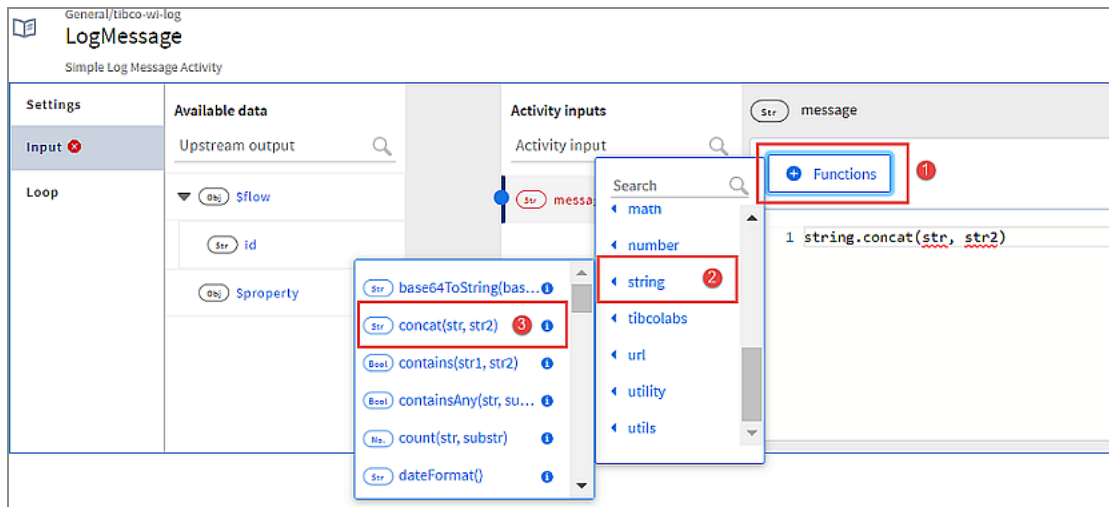
Note: For OSS activities having the **Coerce with schema** option, you can maintain only one schema for the input that you coerce. If you add conditions to the coerced inputs, you cannot change the schema specific to a condition. When you update the schema, it is updated for all the blocks.

Using Functions

You can use a function from the list of functions available under **Functions** in the mapper. Input parameters to the function can either be mapped from an element under **Available data**, a literal value, or an expression that evaluates to the appropriate data type or any combination of them.

The procedure below illustrates an example that concatenates two strings and assigns the concatenated value to the **message**. We manually enter a value for the first string (**str1**) and map the second string to **id** under **\$flow**. The value for **id** comes from the flow input.

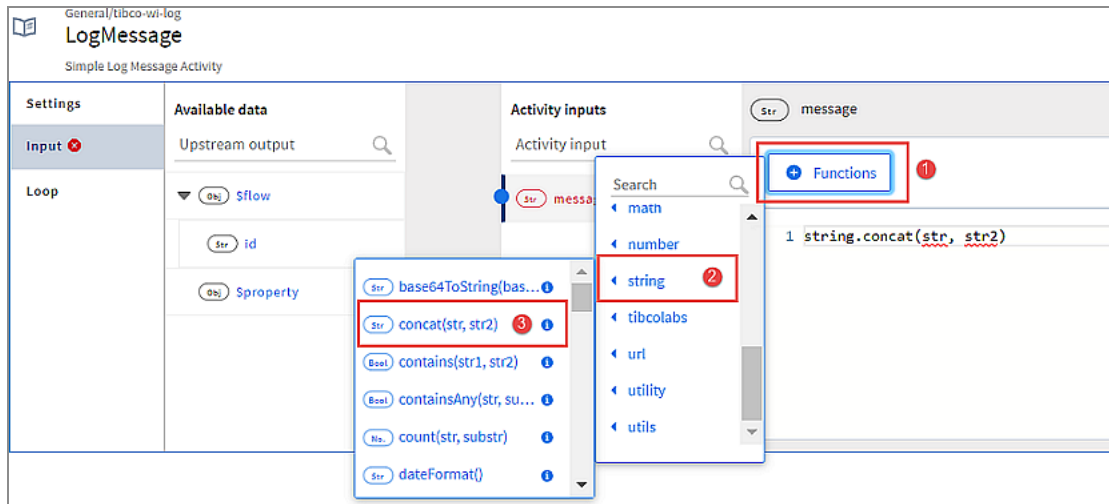
1. Click the **message** to open the text editor to the right.
2. Click **Functions**. Expand the **string** function group and click **concat(str1, str2)**.



3. Select **str1** in the function and type "Received: " (be sure to include the double quotes as shown below) to replace **str1** with it.



4. Drag **id** from **\$flow** and drop it in place of **str2**.



At run time, the output from the concat function is mapped to the **message**.

Using Expressions

You can use two categories of data-mapping expressions in Flogo.

Basic Expression

Basic expressions can be written using any combination of the following by using operators:

- literal values
- functions
- previous Activity or trigger output

See [Supported Operators](#) for details on the operators that can be used within a basic expression.

Here are some examples of basic expressions:

```
string.concat("Rest Invoke response status code:",$activity[InvokeRESTService].statusCode)
```

The above example combines the string and the statusCode from the InvokeRestService activity.

```
string.length($activity[InvokeRESTService].responseBody.data) >=7
```

The above example checks whether the length of data of the responseBody is greater than or equal to 7.

```
$activity[InvokeRESTService].statusCode == 200 && $activity  
[InvokeRESTService].responseBody.data == "Success"
```

The above example checks whether the statusCode is 200 and the data of responseBody has the value as "Success".

Ternary Expression

Ternary expressions are assembled as follows:

```
condition ? statement1 : statement2
```

The condition is to be evaluated first. If it evaluates to true, then statement1 is executed. If the condition evaluates to false, then statement2 is executed.

Here is an example of a basic ternary expression:

```
$Activity[InvokeRESTService].statusCode == 200 ? "Response successfully":"Response failed, status code not 200"
```

In the above example `$Activity[InvokeRESTService].statusCode == 200` is the condition to be evaluated.

- If the condition evaluates to true (meaning `statusCode` equals 200), it returns `Response successfully`.
- If the condition evaluates to false (meaning `statusCode` does not equal 200), it returns `Response failed, status code not 200`.

Here is an example of a nested ternary expression:

```
$Activity[InvokeRESTService].statusCode == 200 ? $Activity[InvokeRESTService].responseBody.data == "Success" ? "Response with correct data" : "Status ok but data unexpected" : "Response failed, status code not 200"
```

The example above checks first to see if `statusCode` equals 200.

- If the `statusCode` does not equal 200, it returns `Response failed, status code not 200`.
- If the `statusCode` equals 200, only then it checks to see if the `responseBody.data` is equal to "Success".
 - If the `responseBody.data` is equal to "Success", it returns `Response with correct data`.
 - If the `responseBody.data` is not equal to "Success", it returns `Status ok but data unexpected`.


Combining Schemas Using Keywords

You can use the `oneOf`, `allOf`, and `anyOf` keywords to combine schemas.

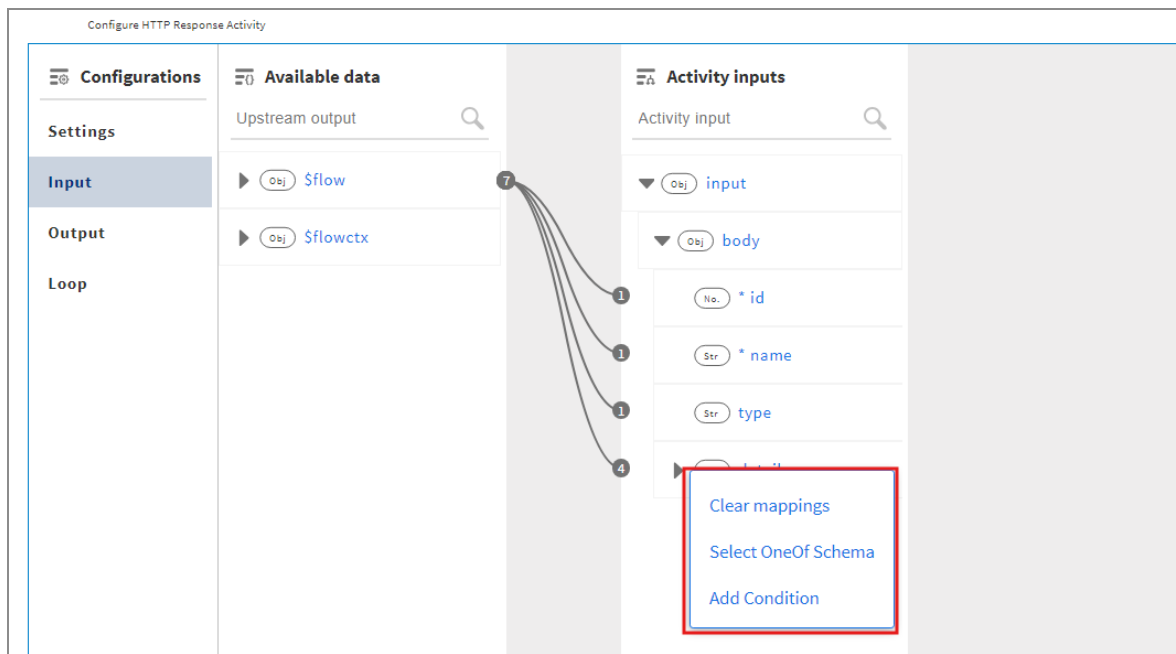
- `oneOf` keyword: This keyword can be used to validate the given data against one of the specified schemas.
- `allOf` keyword: This keyword helps the user ensure that the given data is valid against all the specific schemas.
- `anyOf` keyword: This keyword is used to ensure that the given data is valid against any

specific schema.

Using the oneOf Keyword

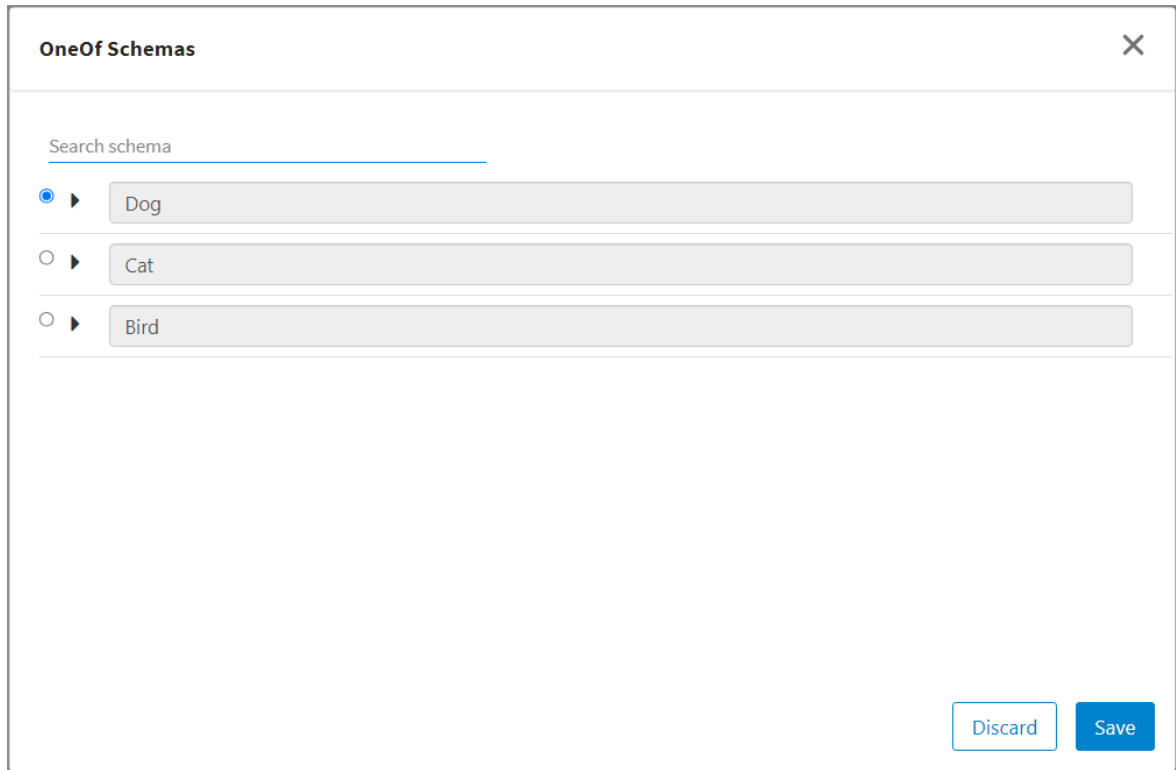
1. On the schema object in the activity/trigger input, click .

For an object with a oneOf keyword, the **Select OneOf Schema** option is displayed.



2. Click **Select OneOf Schema**.


The schema selector dialog displays all available schemas.



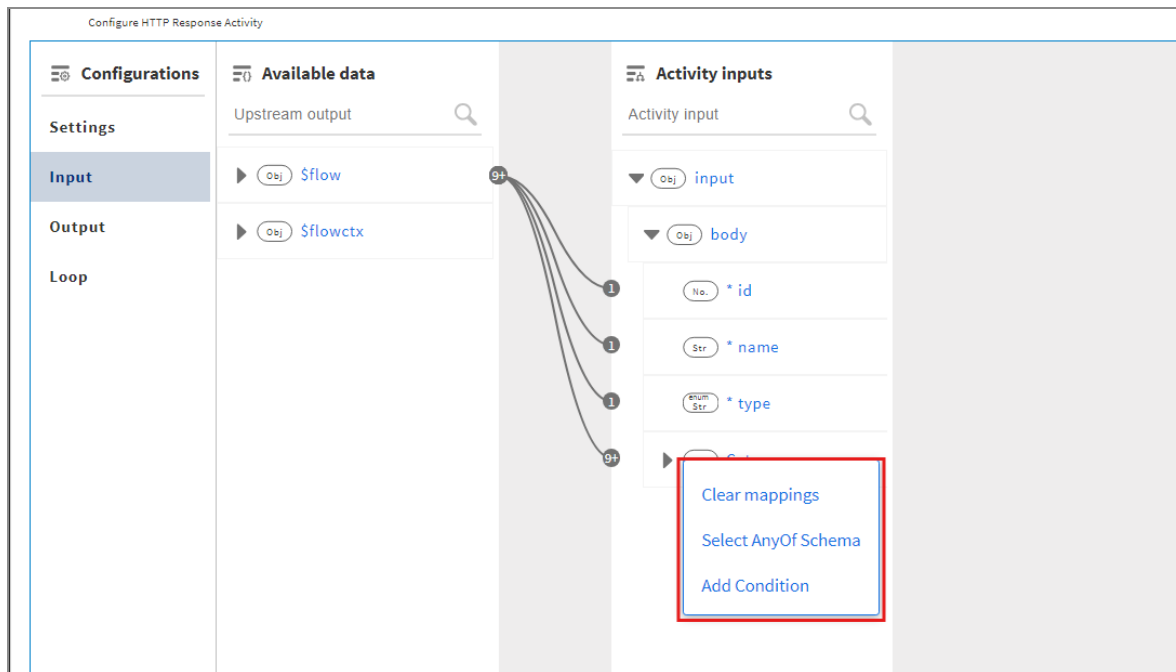
3. Select one schema from the schema selector dialog.

The mapping tree is rendered with the selected oneOf schema nodes.

Using the anyOf Keyword

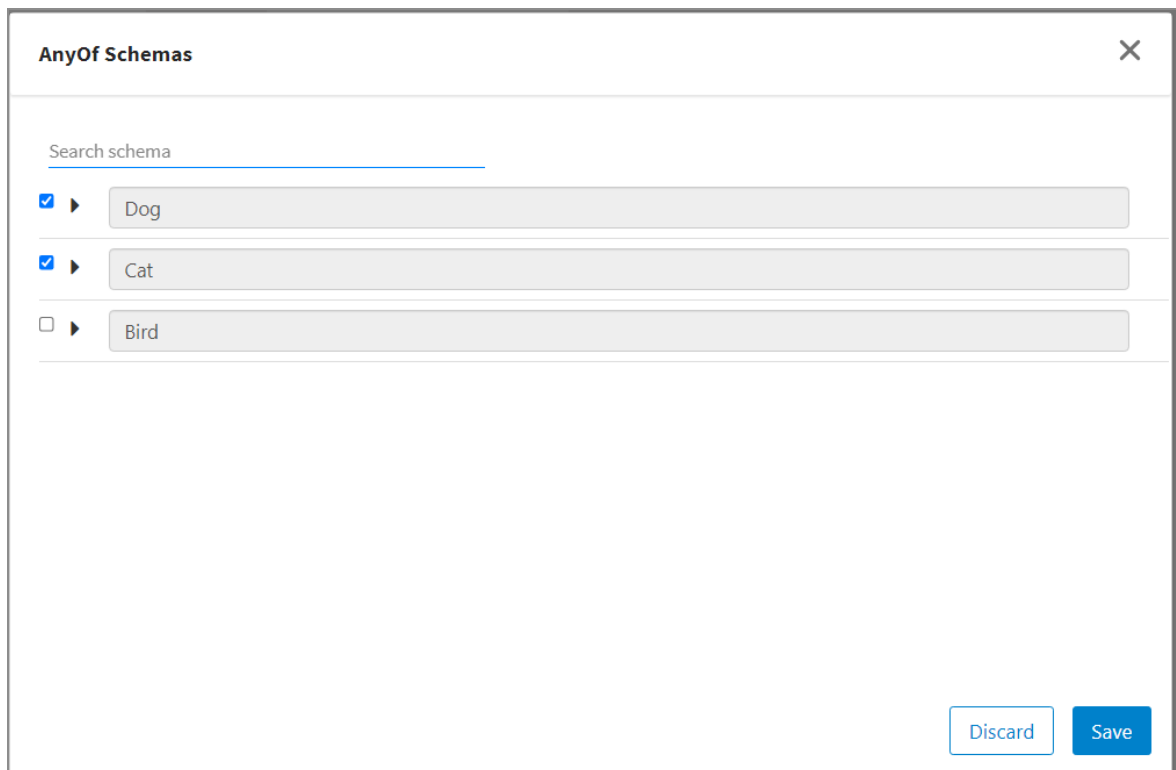
1. On the schema object in the activity/trigger input, click .

For an object with a anyOf keyword, the **Select AnyOf Schema** option is displayed.



2. Click **Select AnyOf Schema**.

The schema selector dialog displays all available schemas.



3. Choose any number of schemas from the schema selector dialog.

The mapping tree is rendered with the selected anyOf schema nodes.

Supported Operators

Flogo supports the operators that are listed below.

- ==
- ||
- &&
- !=
- >
- <
- >=
- <=
- +
- -
- /
- %
- Ternary operators - nested ternary operators are supported.

For example, `$activity[InvokeRESTService].statusCode==200?($activity[InvokeRESTService].statusCode==200?true:false):false`

General Category Triggers, Activities, and Connections

The **General** category is available by default in all flows. It consists of activities, triggers, and connections that may be commonly used by any flow in the app. A trigger initiates the flow in which it appears. Activity is used to perform a task. A connection is used to connect an app to various services.

Triggers

In addition to the triggers available for general use, triggers that were originally created in Project Flogo® are supported. Such triggers are marked with an OSS tag on them.

It is preferable to use the general-purpose triggers (the triggers that do not have an OSS tag on them) as they have richer functionality.

For more information on the triggers that are marked with an OSS tag, see <https://github.com/project-flogo/contrib>.

Trigger configuration fields are grouped into **Trigger Settings** and **Handler Settings**. A single trigger can be associated with multiple handlers.

- **Trigger Settings** - these settings are common to the trigger across all flows that use that trigger. When **Trigger Settings** are changed, the change applies to all flows that are attached to the trigger. A warning message is displayed asking you to confirm the changes before they are committed.
- **Handler Settings** - these settings apply to a specific flow attached to the trigger. Hence, each flow can set its values for the **Handler Settings** fields in the trigger. To do so, open the flow and click the trigger to open its configuration dialog. Click the **Settings** tab and edit the fields in the **Handler Settings** section.

**Note:**

- You cannot create a flow branch from a trigger.
- You can create the trigger at the time of flow creation or create a blank flow to begin with and attach the flow to one or more triggers later after the flow has been created. If you anticipate that you might need to attach the flow to multiple triggers, be sure to create a blank flow and attach it to the triggers as needed.

For triggers that have an output, the output from the trigger becomes the input to the flow. Likewise, the output from the flow becomes the reply from the trigger.

When using the Lambda, S3, or Gateway triggers, keep the following in mind:

- You can only have one trigger. The Lambda trigger supports only one handler per trigger, it can have only one flow attached to it. The S3 and Gateway triggers support multiple handlers (flows), so you can have multiple flows in the app that are attached to the same S3 or Gateway trigger.

- An app that has one of these triggers cannot contain any other trigger.
- You can also have blank flows in the app, which can serve as subflows for the flows that are attached to one of these triggers.

Timer Trigger

Use **Timer Trigger** as a process starter when creating flows designed to be activated without external input. It is useful when you want your flow to run at certain time intervals. You can also configure the **Timer Trigger** to activate the flow multiple times at a specified interval or to set a **Cron Job**.

Trigger Settings

Field	Description
Handler Settings	
Scheduler Options	<ul style="list-style-type: none"> • Timer: Runs the flow at the specified time or interval • Cron Job: Provides more customization options for scheduling the runs
Additional information about fields is available when you select the scheduler option as Timer	
Start Time	<p>Use the calendar to set a start date and time. When configuring an app property for the Start Time, use the RFC 3339 date and time formats.</p> <p>Format with UTC offset:</p> <p>YYYY-MM-DDTHH:MM:SS+00:00</p> <p>OR</p> <p>Format without UTC offset:</p> <p>YYYY-MM-DDTHH:MM:SSZ</p> <p>Here,</p> <p>T is used as a separator between date (YYYY-MM-DD) and time (HH:MM:SS). Replace it with a space, if needed.</p>

Field	Description										
	<ul style="list-style-type: none">2021-04-12T23:20:50.52+00:002021-11-22T09:16:47Z1996-12-19T16:39:57-08:001990-12-31 23:59:60+05:00 <div>Note: Start Time by Default is <i>Blank</i>. If there is no date and time mentioned, the flow runs as soon as you push the changes or app binary.</div>										
Use these fields in combination to define the schedule for recurring runs.											
Repeating	When Repeating is enabled, the flow is triggered at the same time as the first run at the specified time interval. (Default: False)										
Delayed Start	Delayed Start delays only the first execution. The successive runs are triggered at the specified time interval. (Default: False)										
Time Interval	This is a period between two successive runs.										
Interval Unit	This is a unit specified for Time Interval .										
Additional information about fields is available when you select the scheduler option as Cron Job											
Cron Expression	<p>You can enter any Cron Expression manually in this field.</p> <p>The standard cron expression should be CRON_TZ=IANA TimeZone * * * * *</p> <p>Here,</p> <p>CRON_TZ=IANA TimeZone is used to set the specific time zone in which the cron job is run.</p>										
<table><tr><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td></tr><tr><td>Minutes</td><td>Hours</td><td>Day of Month</td><td>Months</td><td>Day of Week</td></tr></table>		*	*	*	*	*	Minutes	Hours	Day of Month	Months	Day of Week
*	*	*	*	*							
Minutes	Hours	Day of Month	Months	Day of Week							
	<table><tr><td>Cron expression</td><td>Schedule</td></tr></table>	Cron expression	Schedule								
Cron expression	Schedule										

Field	Description
Cron Expression Builder	<p>If you use the Cron Expression Builder, this field is auto-populated as you build the expression.</p> <p>Simple cron expressions can be built using the Cron Expression Builder. You can use the different tabs to define the frequency: Minutes, Hours, Day of Week, Day of Month, Months.</p> <div> <p>Note:</p> <ul style="list-style-type: none"> Expressions built using Cron Expression Builder can be modified manually. If there is no time zone mentioned while building a cron expression through the app property, by default the time zone of the system is considered and there might be no logs generated for the executed flow. UI validation for the built cron expression is not supported. </div>

Map to Flow Inputs

This tab allows you to map the trigger output to flow input.

REST Trigger - ReceiveHTTPMessage

Use the **ReceiveHTTPMessage** REST trigger when creating flows that are going to be invoked by an external REST call. The **ReceiveHTTPMessage** trigger exposes your flow as an API, making it accessible to other apps running on TIBCO Cloud™ or elsewhere. This trigger must be configured to set up the fields for a request that the server receives from a REST client.



Note:

- If you add or delete path or query parameters in the trigger, you must click **Sync** for the changes to be propagated to the flow input schema
- REST trigger does not support authentication and authorization headers.


Trigger Settings



Field	Description
Trigger Settings	
Port	<p>By default, the trigger listens on port 9999. You can change this to use another open port. Do not use ports 8080 or 7777, as these ports are reserved for internal use.</p> <p>Important: If the app has multiple triggers that require a port to be specified, specify a unique port number for each trigger. Two triggers in the same app cannot run on the same port.</p>
Configure Using API Specs	<p>While creating a REST trigger, you can configure it by uploading an API specification file.</p> <p>To do this, click True (by default, it is set to False) and specify the following:</p> <p>API Spec: Click Browse and then select the specification file to be used for configuring the trigger. Supported specifications are Swagger Specification 2.0 and OpenAPI Specification 3.0.</p> <p>Click the Use app level spec toggle to select an API uploaded on the SPECS tab. Once you select the API Spec, the other fields, such as Path and Method, are populated per the spec.</p>
Handler Settings	
Path	<p>The resource path for the operation.</p> <p>If you upload an API specification file, select a path from the dropdown list. The path parameters are parsed from the API spec file and the data types displayed are string, integer, or boolean as specified in the file.</p> <p>For manual configuration, the data type for the resource path is string.</p> <p>By default, the path displayed here is the resource path you had entered when you created the flow. The Path field is editable if you have not uploaded an API specification file. For example, if you want to add a path parameter for a GET operation, you can do so by editing the resource path in the GET flow. If</p>

Field	Description
	<p>you edit the path in the Path field for a particular REST operation flow, the edited resource path applies only to the flow in which it was edited.</p> <p>The Path field supports wildcard characters, allowing flexible matching of URL path segments. This enables an endpoint to handle dynamic or variable parts of a URL. For example, <code>/api/{*resource}</code> can match URLs, such as <code>/api/users</code>, <code>/api/books/1</code>, or any other path under <code>/api/</code>.</p> <p>Two resource paths with the same base path should not contain the path parameters at the same location. For example, you cannot have the following paths in the same app:</p> <ul style="list-style-type: none"> • <code>/books/{ISBN}/Author/{releaseDate}</code> and <code>/books/{ISBN}/Author/releaseDate</code> is considered the same from a routing perspective. <p>In these two paths, since the ISBN value is dynamic, it causes a conflict during path resolution.</p> <ul style="list-style-type: none"> • <code>/books/{ISBN}/{releaseDate}</code> and <code>/books/{ISBN}/Author</code> in the same app is not supported. <p>Although the two paths appear to be different, when a message comes in, the router mechanism cannot know which path to call (the one with the parameter or the one without) since the actual value has been substituted for the parameter.</p> <ul style="list-style-type: none"> • Resource path with two different path parameters at the same URL subsection. For example, <code>/0.6/api/account/{account}/orderhistory/{orderhistory}/branch/{branch}</code> and <code>/0.6/api/account/{AccountKey}/Price?ProductList={ProductList}</code> <p>In these paths, even though the paths differ from the base path (<code>/0.6/api/account/</code>), there is a conflict when resolving the <code>{account}</code> and <code>{AccountKey}</code> values.</p> <ul style="list-style-type: none"> • Multiple REST resources with the same base path and the same number of path parameters. For example, <code>/resource/{id}</code> and <code>/resource/{id1}</code> • <code>/messages/{messageid}/comments/{commentid}</code> and <code>/messages/{messageid}/likes/{likeid}</code> <p>Where the paths differ after <code>{messageid}</code>.</p>

Field	Description
Method	The REST operation which the flow implements. Supported HTTP methods are: GET, PUT, POST, DELETE, and PATCH.
Request Type	<p>This field appears only when Method is POST, PUT, or PATCH. Select one of the following types from the dropdown list:</p> <ul style="list-style-type: none"> • application/json • application/x-www-form-urlencoded • multipart/form-data <p>Default: application/json</p> <div> <p>Note: If you create a Flogo app using an API specification file having Request Type as application/x-www-form-urlencoded or multipart/form-data, then you must click Sync to update the request parameters with the Flow Input.</p> </div>
Output Validation	<p>When set to True, the incoming data (body, headers, and query parameters) is validated against the configured JSON schema.</p> <p>Default: False</p>

Output Settings

Field	Description
Query Parameters	<p>Query parameters to be appended to the path. To add the query parameters, click  and press Enter to save your changes.</p> <ul style="list-style-type: none"> • parameterName: Name of the query parameter. • type: The data type of the query parameter. Supported types are string, number, and boolean. • repeating: Set to True if more than one value is expected for the query parameter. • required: Set to True if the query parameter is a required configuration. The trigger reports an error if no values are provided to


Field	Description
	the required query parameter.
Path Parameters	Path parameters that are appended to the path.
Headers	<p>Header values for the trigger. To add the header parameters, click  and press Enter to save your changes.</p> <ul style="list-style-type: none"> • parameterName: Name of the header parameter. • type: The data type of the header parameter. Supported types are string, number, and boolean. • repeating: Set to True if more than one value is expected for the HTTP header. • required: Set to True if the header parameter is a required configuration. The trigger reports an error if no values are provided to the required header parameter.
Request Schema	<p>Enter a request schema here. This field is visible only if you selected the POST, PUT, PATCH, or DELETE method on the Settings tab.</p> <div> <p>Note:</p> <ul style="list-style-type: none"> • For the DELETE method, specifying the request schema here is supported for manual configuration only and not when configured with TIBCO Cloud™ Mesh and API specification. • If you selected application/x-www-form-urlencoded as the Request Type on the Settings tab, the default schema is set here. You can edit the default schema or specify your schema. If you specify your schema, it must be a name-value string pair. </div>
Multipart Data	<p>This field is displayed in place of the Request Schema field if you select multipart/form-data as the Request Type in the Trigger Settings. Click  to add the parameters.</p> <ul style="list-style-type: none"> • Name: Parameter name. • Type: Supported types are string, object, and filecontent.

Field	Description
	<ul style="list-style-type: none"> • Required: Check the box if the parameter is a required configuration. • Schema: Enter the JSON schema in this field if the Type is an object. <p>Note: The file content received by the trigger is converted into a byte array. This is passed to the Activity as a Base64-encoded value in an array. If you want to fetch the content of the file, coerce the first element of the array to a string.</p>

Map to Flow Inputs

This tab allows you to map the trigger output to the flow input.

Reply Settings

Field	Description
Configure Response Codes	<p>Allows you to configure response codes.</p> <p>Default: False (See "Reply Data Schema" in this table.)</p> <p>To specify a response code, select True and click . Enter the following details:</p> <ul style="list-style-type: none"> • Code: Enter the response code. • Type: Select the type of response expected for the Code. Supported types are String and Object. • Response Body: If the Object is selected as the Type, enter the JSON schema in the Response Body column. For String, you need not enter anything in the Response Body column. • Response Headers: The header parameters for the reply are in JSON data format. • Actions: The actions displayed change based on the type of the response code. <ul style="list-style-type: none"> ◦ Edit, Delete: For an Object type of response, you can edit the

Field	Description
-------	-------------

details or cancel it.

- **Save, Cancel:** For each **String** type of response, you can save or cancel the changes.

The response codes appear on the Map from Flow Outputs tab.

Note: The REST reply data type is by default set to data type any.

For multiple response codes, use the **ConfigureHTTPResponse** Activity in the flow to map **Response Body** and **Response Headers** from the REST trigger with the **Input** in the Activity. To configure the **ConfigureHTTPResponse** Activity, see [ConfigureHTTPResponse](#).

The image shows the **Reply Settings** with multiple response codes.



Caution: If you modify the response code schema in the table, the corresponding **ConfigureHTTPResponse** activities within that flow do not change appropriately. This happens specifically when removing fields from the **Reply Settings** tab. Redo the mappings for the **ConfigureHTTPResponse** activities.

Reply Data Schema	Note: This field appears only when Configure Response Codes is set to False .
-------------------	--

The schema is used for the reply data of the trigger. Be sure to use straight quotes for element names and values in the schema.

Map from Flow Outputs

Map the flow output to the trigger reply on this tab.

i Note:

- To update these settings for a trigger configured from the Swagger 2.0 or OpenAPI 3.0 specification, update the API specification file and upload it to the **Trigger Settings**. Do not update the settings as manual updates are removed.
- If you are using a REST trigger in your app:
The endpoint URL contains the app name if the app has one trigger.
The endpoint URL contains the app name and trigger name for more than one trigger.

If you add a REST trigger to an existing app, you must reconfigure the client app.

GraphQL Trigger

The **GraphQL** trigger lets a Flogo app act as the GraphQL server. To use this trigger, you simply upload your **GraphQL** schema and TIBCO Flogo® Enterprise automatically creates the flows corresponding to each query or mutation field in your schema.

Trigger Settings

Field	Description
Trigger Settings	
Port	<p>The port on which the trigger listens to requests. By default, it is set to 7879. You can change this to use any other open port. Do not use ports 8080 or 7777, as these ports are reserved for internal use. This field can also be set using an app property.</p> <div> <p>Important: If the app has multiple triggers that require a port to be specified, ensure that the port number is unique for each trigger. Two triggers in the same app cannot run on the same port.</p> </div>
Path	The HTTP resource path for the operation. By default, it is set to /graphql,

Field	Description
	but you can change it to any string that is meaningful to you. It is the single endpoint that GraphQL queries and mutations use to access data from the multiple resources on the server. This field can also be set using an app property.
GraphQL Schema File	<p>When creating a GraphQL trigger, you can configure it by uploading a GraphQL Schema File with a .gql or .graphql extension. To do this, either click Browse and then select the GraphQL schema file from your local disk or click the Use app level spec toggle to select GraphQL schema spec uploaded on the SPECS tab. Once you select the schema file the other fields, such as GraphQL Operation and Resolver for, are populated as per the spec.</p> <p>Note: If you have changed the GraphQL schema file that you uploaded when creating the flow or trigger, you must propagate the changes to the flow input and flow output. To do this, after you select the updated schema file in this field, click Sync on the upper-right corner.</p>
Schema Introspection	By default, it is set to True . You can use it to get the schema details for GraphQL. If you set it to False , it disables introspection and you cannot fetch the schema field details.
Secure Connection	<p>By default, it is set to False. If you set this field to True, you can create a secure endpoint by providing Server Key and CA or Server Certificate.</p> <p>Server Key - A PEM encoded private key file</p> <p>CA or Server Certificate - A PEM encoded CA or self-signed server certificate file</p>
Handler Settings	
GraphQL Operation	The type of GraphQL operation the flow should represent. You can select either Query or Mutation
Resolver for	This field is populated based on the type of GraphQL Operation that you selected. If you selected Query , the Resolver For lists the field names under the query type in the schema. If you select Mutation , the dropdown menu lists the field names under the mutation type in the schema.

Map to Flow Inputs

You can map the trigger output to flow input on this tab. The tab contains an element arguments, which contains a field or objects list that matches the input arguments of the **Resolver For** field in the GraphQL schema. **Fields** and **Headers** are available to map inside **Map to Flow Inputs**. **Fields** contains the fieldName string and the field array passed for the given **Query** or **Mutation**. The following structure is an example of **Fields**:

```
{"fieldName":"employee","fields":[{"fieldName":"id"}, {"fieldName":"name"}]}
```

Map from Flow Outputs

You can map the flow output to the trigger reply on this tab. The tab contains a child element data, which contains either a simple type or an object that matches the output type of the **Resolver For** field in the GraphQL schema. If the output type of the field is an interface type, the data contains a single field of type any. You can use the Error field to map the trigger reply setting for any error in the flow. Its value is parsed as a string.

To avoid any runtime exception, you can define your data and error mapping using the following formats:

i Note: To avoid any runtime exception, you can define your data and error mapping using the following formats:

- Data: `isDefined($flow.data) ? $flow.data : coerce.toObject('{}')`
- Error: `isDefined($flow.error) ? $flow.error : "`

gRPC Trigger

The **gRPC** trigger acts as a server to gRPC clients.

i Note: You must run the preinstall script before running the gRPC trigger or activity binary.

Trigger Settings

Field	Description
Trigger Settings	
Port	The port on which the trigger listens to requests. You can use any open port. This field can also be set using an app property.
Proto File	<p>When creating a gRPC trigger, you can configure it by uploading a .proto file. To do this, either click Browse and then select the .proto file from your local disk or click the Use app level spec toggle to select a spec uploaded on the SPECS tab. Once you select the file, the other fields are populated as per the spec.</p> <p>Note: The gRPC trigger and gRPC Activity do not support options in the .proto file. For more information, see the limitations when creating a .proto file in gRPC Activity.</p>
Secure Connection	<p>By default, it is set to False. If you set this field to True, you can create a secure endpoint by providing values for Server Certificate and Server Key.</p> <p>Note: If the secure connection is True, you can see an optional Mutual TLS field.</p>
Mutual TLS	<p>Set it to True to enable mutual authentication for a secure connection to the server. The default value is False.</p> <p>Note: When Mutual TLS is enabled, it requires the values for CA Certificate, Server Certificate, and Server Key.</p>
CA Certificate	<p>A PEM-encoded CA certificate. This certificate is required only when Mutual TLS is enabled. It is used to authenticate the client's or the server's certificate during the mutual TLS handshake, ensuring mutual trust between both parties.</p> <p>Browse and select a Certificate Authority (CA) certificate that is used for mutual TLS authentication.</p>

Field	Description
	Alternatively, you can configure the app property using the Bind an Application Property toggle. Set the Base64-encoded value of the CA certificate to the corresponding app property.
Server Certificate	<p>A PEM-encoded server certificate.</p> <p>This certificate is used to authenticate the server to the client over TLS. Browse to select the server certificate.</p> <p>Alternatively, you can configure the app property using the Bind an Application Property toggle. Set the Base64-encoded value of the server certificate to the corresponding app property.</p>
Server Key	<p>A PEM-encoded private key file. Browse and select the server key.</p> <p>Alternatively, you can configure the app property using the Bind an Application Property toggle. Set the Base64-encoded value of the server key to the corresponding app property.</p>
Handler Settings	
Service Name	Name of the service defined in the .proto file. You must create one gRPC trigger for any specific .proto file. Any subsequent gRPC triggers using the same .proto file can select the service and method they need from the dropdown list.
Method	Name of the RPC method in the .proto file. Each method in the .proto file is represented by a separate flow and attached to the same gRPC trigger.

Map to Flow Inputs

You can map the trigger output to flow input on this tab. This tab displays fields from your selected method.

Map from Flow Outputs

You can map the flow output to the trigger reply on this tab.

Receive Lambda Invocation

Use the **Receive Lambda Invocation** trigger for AWS to start your flow as a Lambda function. The **Receive Lambda Invocation** trigger can be configured only in blank flows. It must not be used with flows that are created with another trigger.

Trigger Settings

i Note:
An app can contain only one Lambda trigger. An app that has a Lambda trigger cannot contain any other triggers including another Lambda trigger. Also, as the Lambda trigger supports only one handler per trigger, it can have only one flow attached to it. However, the apps that contain a Lambda trigger can contain blank flows that can serve as subflows for the flow attached to the Lambda trigger.

Field	Description
AWS Connection Name	Name of the AWS connector connection you want to use for the flow.
Execution Role Name	(optional) ARN of the role to be used to execute the function on your behalf. The role must be assumable by Lambda and must have CloudWatch logs permission execution role.

Output Settings

Enter the payload schema for the request received on the Lambda function invocation on AWS.

Map to Flow Inputs

This tab allows you to map the trigger output to flow input.

Field	Description
Function	Information about the Lambda function
Context	Envelope information about this invocation
Identity	Identity for the invoking users
ClientApp	Metadata about the calling app
API Gateway Request	Displays the elements in the payload schema that you entered on the Output Settings tab. The elements are displayed in a tree format.

Reply Settings

Enter a schema for the trigger reply in the Reply Data text box.

Map from Flow Outputs

Map the flow output to the trigger reply on this tab.

App Startup Trigger

This trigger allows you to execute flows before other triggers in the app are started. It can be used to specify initialization logic that is specific to an app. For example, this trigger can be used for the following tasks:

- Setting data or cache for later use in other flows
- Initialization of a database (insertion or extraction of data from tables)

Design Considerations

- You can add one or more **App Startup** triggers to an app.
- You can add the **App Startup** trigger along with the Receive Lambda trigger. You cannot add any other trigger with the **Receive Lambda** trigger.
- The trigger supports multiple handlers. So, you can configure more than one flow in

a trigger.

- The flows are executed in the order in which they are configured in the trigger.
- The trigger is executed for all instances of the app. For example, if you scale up to multiple instances of the app, the trigger is executed on each scale-up.
- If a startup flow fails, the engine is terminated.
- In container deployments, the collective execution time of all flows configured to this type of trigger must not exceed the startup time set for the app.

App Shutdown Trigger

This trigger allows you to run flows after all other triggers in the app are successfully stopped. It can be used to specify shutdown logic that is specific to an app. For example, this trigger can be used for the following tasks:

- Cleaning up data or cache
- Deleting tables from a database

Design Considerations

- You can add one or more **App Shutdown** triggers to an app.
- You can add the **App Shutdown** trigger along with the **Receive Lambda** trigger. You cannot add any other trigger with the **Receive Lambda** trigger.
- The trigger supports multiple handlers. So, you can configure more than one flow in the trigger.
- The flows are executed in the order in which they are configured in the trigger.
- The trigger is executed for all instances of the app. For example, if you scale down multiple instances of the app, the trigger is executed on each scale down.
- If an app is forcefully shut down, the trigger and subsequent flows are not executed. This trigger is executed only when an app is gracefully shut down.
- In container deployments, the collective execution time of all flows configured to this type of trigger must not exceed the graceful-stop time set for the app.
- You must exercise caution when defining a flow in the **App Shutdown** trigger. For example, when an app is scaled to more than one instance, cleanup is done when

shutting down one instance may impact other running instances.

Activities

In addition to the activities available for general use, activities that were originally created in Project Flogo® are supported. Such activities are marked with the OSS tag on them. The Project Flogo® activities are placed under the **Default** category.

It is preferable to use the general-purpose activities (the activities that do not have the OSS tag on them), as they have richer functionality.

For more information on the activities that are marked with the OSS tag, see <https://github.com/TIBCOSoftware/flogo-contrib>.

The available activities are placed under the following categories:

- Default
- General



Important: After configuring any activity or modifying the activity configuration, you must explicitly click **Save** for your configuration changes to be saved or click **Discard** to discard your changes.

You can create a flow branch from any Activity except the **Return** Activity.

For quick access to activities specific to the user and browser, you can use the following tabs:

- **Recently Used:** This tab has the activities used recently for the flow configuration. Eight of the last-used activities can be accessed from this tab.
- **Recently Installed:** This tab has custom extension activities installed recently for flow configuration.

ConfigureHTTPResponse

This activity is used to configure HTTP response codes that you want to use in your REST reply.

When using this activity, keep the following considerations in mind:

- The flow in which you want to add the **ConfigureHTTPResponse** activity must have a **ReceiveHTTPMessage** trigger.
- Do not use this activity with subflows.

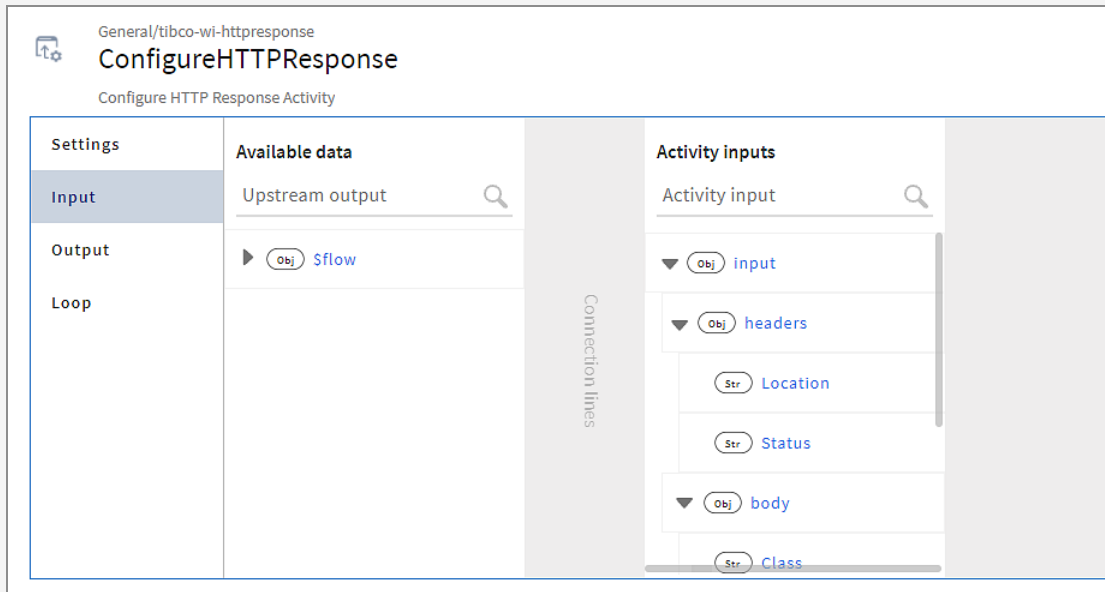
Settings

Field	Description
Trigger Name	The list of triggers to which this flow is attached. This field is displayed only when the flow is attached to multiple REST triggers. Select a trigger in which the code that you want to use is configured.
Code	List of response codes that you have configured in the selected trigger. Select a response code that you want to use.

Input

Displays the schema for the code that you selected on the **Settings** tab. Map the elements in the schema in the mapper or manually enter the values that you want to send as a response.

Note: For multiple response codes in the [REST Trigger - ReceiveHTTPMessage](#), map the **Response Body** and **Response Headers** from the trigger with the **Input** in this Activity. The image shows the **Input** with **headers** and **body** mapped from the REST trigger response codes.



Output

Displays the schema for the code in a read-only tree format.

Run JavaScript

This activity runs a JavaScript code using the specified input parameters and returns the result in the output.

You can use this activity to write complex logic in JavaScript, which may not be straightforward to achieve using the mapper. For example, you can easily filter arrays based on some conditions or looping through arrays using `forEach` and other useful JavaScript functions.

This activity supports both ECMAScript 5-compatible and ECMAScript 6-compatible functions. For more information, see [ECMAScript 5-compatible functions](#) or [ECMAScript 6-compatible functions](#).

Important Considerations

- You may see some code suggestions in the editor that are not valid for ECMAScript 5-compatible or ECMAScript 6-compatible functions. It is not a best practice to implement the suggestions. If you implement them, the Activity may return some errors.
- For ECMAScript 6-compatible functions, the following are not supported.
 - Classes
 - Promises
 - Default parameters
 - Function REST parameter
 - Arrow functions

Settings

Field	Description
Javascript Code	<p>Specify the JavaScript code to be run in the following format:</p> <pre>var result, parameters <JavaScript code></pre> <p>Where:</p> <ul style="list-style-type: none">• The result variable must be defined for the output of the JavaScript.• The parameters variable must be defined for the input of the JavaScript. Set the input parameters on the Input Settings tab. Set the output parameters on the Output Settings tab.

Input Settings

Field	Description
Script Input Parameters	<p>Configure a schema for one or more input parameters to the JavaScript. The elements of this schema are available for mapping on the Input tab.</p> <p>Use app-level schema: Click Use app-level schema and select a schema that you might have defined earlier.</p> <p>Change: To change the schema, click Change.</p> <p>Syntax to access the input parameter value defined in the JavaScript code:</p> <pre>parameters.<parameter_name></pre> <p>For example, if you have defined the input parameter as foo in the JavaScript code, use the following syntax to access the value:</p> <pre>parameters.foo</pre>

Input

The **Input** tab displays the schema that you entered in the **Input Settings** tab in a tree format. Map the elements in the schema using the mapper or manually enter the value for the element in the mapper.

Output Settings

Field	Description
Script Output	<p>Configure a schema for one or more output parameters of the JavaScript.</p> <p>Use app-level schema: Click Use app-level schema and select a schema that you might have defined earlier.</p>

Field	Description
	<p>Change: To change the schema, click Change.</p> <p>Syntax for setting the value of the output parameter in the Javascript code:</p> <pre>result.<parameter_name></pre> <p>Example:</p> <pre>result.foo = bar</pre>

Output

The **Output** tab displays the output parameters from the schema that you entered in the **Output Settings** tab. The output parameters are displayed in the result object in a tree format.

Field	Description
error	Flag indicating whether an error occurred while running the JavaScript.
errorMessage	The error message.
result	The output of the JavaScript code indicates successful execution of the JavaScript.

gRPC Invoke

The **gRPC Invoke** Activity is an implementation of a gRPC client. Use this activity to make outbound gRPC calls.



Note: You must run the preinstall script before running the **gRPC** trigger or activity binary.

Settings

Field	Description
Host URL	The URL used to connect to the gRPC server. The name or IP address of the host on which your .proto file resides. Be sure to append the port number on which the service is running.
Secure Connection	<p>By default, it is set to False. If you set this field to True, you can create a secure endpoint by providing a value for CA Certificate.</p> <p>Note: If secure connection is True, you can see an optional Mutual TLS field. After enabling Mutual TLS, you must add a client certificate and key.</p>
Mutual TLS	<p>Set it to True to enable mutual authentication for a secure connection to the server. The default value is False.</p> <p>Note: When Mutual TLS is enabled, it must provide the values for CA Certificate, Server Certificate, and Server Key.</p>
CA Certificate	<p>A PEM-encoded CA certificate.</p> <p>Browse and select a Certificate Authority (CA) certificate to establish a secure connection. The CA Certificate is required when secure connection is enabled and is necessary for Mutual TLS to validate the server's certificate during the handshake.</p> <p>Alternatively, you can configure the app property using the Bind an Application Property toggle. Set the Base64-encoded value of the CA certificate to the corresponding app property.</p>
Client Certificate	<p>A PEM-encoded client certificate.</p> <p>This field is displayed only if Mutual TLS is set to True. This certificate is used to identify the client by the server during mutual TLS authentication.</p> <p>Browse and select the client certificate. Alternatively, you can configure the app property by using the Bind an Application Property toggle. Set the</p>

Field	Description
	Base64-encoded value of the client certificate to the corresponding app property.
Client Key	<p>A PEM-encoded private key file.</p> <p>This field is displayed only if Mutual TLS is set to True.</p> <p>Browse and select the Client Key. The key file must be PEM-encoded. It is required for mutual TLS authentication along with the client certificate.</p> <p>Alternatively, you can configure the app property by using the Bind an Application Property toggle. Set the Base64-encoded value of the client key to the corresponding app property.</p>
Proto File	<p>A file with the .proto extension that contains the methods and services definition, which Flogo Extension for VS Code uses to create the flows. Each flow corresponds to one method in the .proto file. Currently, importing a .proto file into another .proto file is not supported.</p> <div> <p>Note:</p> <ul style="list-style-type: none"> • The gRPC activity does not support options in the .proto file. If your .proto file contains any options, be sure to remove the options in the .proto file before using it. • You must not use the same gRPC .proto file for the gRPC activities in the same app. The package names for the gRPC activities must be unique. </div>
Service Name	Name of the service you want to invoke. The service is defined in the .proto file that you have selected.
Method	Name of the RPC method in the selected .proto file. Each method in the .proto file is a gRPC request, which is represented by a separate flow.

Input

Field	Description
Activity Input	The Input tab lists the parameters for the method that you chose on the Settings tab so that you can either enter or map their values in the mapper.

Output

The **Output** tab displays a read-only schema of the Activity output (the response that is configured for the selected method).

Iterator

For details on using the iterator, see the "Using the Iterator in an Activity" section.

Limitations When Creating a .proto File

When creating a .proto file, you must adhere to the following limitations:

- Currently, importing a .proto file into another .proto file is not supported. Therefore, you cannot use import statements.
- As import statements are not supported, you cannot use data types, such as `google.protobuf.Timestamp` and `google.protobuf.Any`, which need to be imported from other .proto files.
- Streaming is not supported in the request or the response.
- Cyclic dependency in request or response messages is not supported.
- Setting a default value to a blank field within a message is not supported.
- Maps for data definition are not supported.
- Oneof: gRPC mandates that you enter a value for one field only. All fields are optional, allowing the user to select any field and enter a value. If the user enters multiple values, then the value entered in the last field is considered and the remaining values above that field are ignored.

InvokeLambdaFunction

Use this Activity to invoke a specific Lambda function.

Settings

Field	Description
AWS Connection Name	Select an AWS connection.
ARN (Optional)	Amazon Resource Name. Note: <ul style="list-style-type: none">You can also specify the ARN on the Input tab. If you specify the ARN on both the tabs, the ARN on the Input tab is used.You must specify the ARN on at least one tab. Otherwise, the activity returns an error at runtime.

Input Settings

Field	Description
Payload Schema	Enter a JSON request schema for your payload that is used to invoke the Lambda function.

Input

The payload schema that you entered on the **Input Settings** tab is displayed in a tree format on the **Input** tab. Map the elements in the schema using the mapper or enter values for the element by manually typing the value in the mapper.

Field	Description
LambdaARN	Amazon Resource Name. Note: <ul style="list-style-type: none">You can also specify the ARN on the Settings tab. If you specify the ARN on both the tabs, the ARN on the Input tab is used.You must specify the ARN on at least one tab. Otherwise, the Activity returns an error at runtime.

Output Settings

Field	Description
Result Schema	The schema for the result that is expected from the Lambda function invokes the request.

Output

The Output tab displays the result schema that you entered on the **Output Settings** tab in a tree format.


InvokeRESTService



This Activity is used to request a REST service. It also accepts the reply returned by the service.

Settings

Field	Description
API Spec	(Optional) Click Browse and browse to the file location on the machine. Select a JSON file.

Field	Description
	<p>Supported specifications are Swagger Specification 2.0 and OpenAPI Specification 3.0.</p> <p>Click the Use app level spec toggle to select an API uploaded on the SPECS tab. Once you select the API Spec, the other fields, such as Path and Method, are populated per the spec.</p>
Resource Path	<p>Note: This field is only displayed if you upload a JSON file in the API Spec field.</p> <p>All resource paths available in the JSON file (that is, the Swagger 2.0 or OpenAPI 3.0 specification file you uploaded) are listed in the dropdown menu. Depending on the resource path you select, the supported operations are listed in the Method field.</p>
Enable Authentication	<p>Select True if you want to enable authentication and authorization for your apps using the HTTP Client Authorization Configuration connection.</p> <p>Default: False</p>
Authentication Connection	<p>Note: This field is displayed only if the Enable Authentication field is set to True.</p> <p>Select a connection that you have set up from the dropdown list.</p> <p>For information on setting up a connection, refer to HTTP Client Authorization Configuration.</p>
Method	<p>Select an operation for the request. For example: GET, POST, PUT, DELETE, or PATCH.</p>
URL	<p>An absolute path to the REST service that you want to invoke. For example: http://acme.com or https://acme.com.</p>

Field	Description
	<p>Note: If you upload an OpenAPI 3.0 JSON specification file in the API Spec field, the URL is a dropdown list. This lists the server URLs mentioned in the JSON file. Select a server URL from the list.</p> <p>Tip: To dynamically override the path provided in the URL, you can enter the URL as: <code>http://<host-url>:port/{path}</code> Here, <code>{path}</code> is the parameter that can be modified. You can map this parameter in the Input section and assign values to it from the previous activities or the app properties.</p>
Use certificate for verification	<p>This field is displayed if you enter an absolute path beginning with <code>https://</code> in the URL field. Set this to True to use a certificate for a secure connection to the server.</p> <p>Default: False</p>
Use mTLS	<p>Set to True to enable mutual authentication for a secure connection to the server.</p> <p>Default: False</p>
Client Certificate	<p>This field is displayed only if Use mTLS is set to True.</p> <p>This certificate is used to identify the client by the servers over TLS. The certificate must be PEM encoded. Click Browse and select the client certificate.</p> <p>Alternatively, you can configure the app property by using the Bind an Application Property toggle . Set the Base64-encoded value of the client certificate to the corresponding app property or pass file path as <code>file://path/to/cert/file</code>.</p>
Client Key	<p>This field is displayed only if Use mTLS is set to True.</p> <p>Click Browse and select the client key. The key file must be PEM encoded.</p> <p>Alternatively, you can configure the app property by using the Bind an</p>

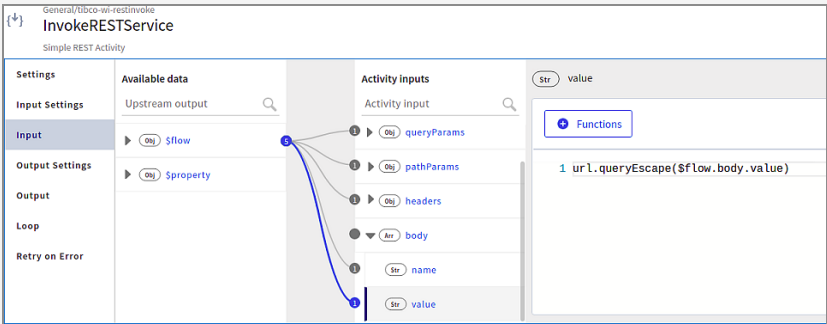
Field	Description
	<p>Application Property toggle . Set the Base64-encoded value of the client key to the corresponding app property or pass the file path as file://path/to/key/file.</p>
CA/Server Certificate	<p>Click Browse and select a certificate authority (CA) certificate that verifies the client's certificate or the server certificate to establish a secure connection during the TLS handshake. The certificate must be PEM encoded.</p> <p>Alternatively, you can configure the app property by using the Bind an Application Property toggle . Set the Base64-encoded value of the CA/Server certificate to the corresponding app property or pass file path as file://path/to/cert/file.</p> <p>You can copy the server/root certificate under the system CA certificates. The activity loads all system certificates and appends any user-configured certificates. Providing a certificate in the CA/Server certificate field is not mandatory.</p>
Disable SSL Verification	<p>Set to True to disable SSL verification when invoking SSL and mTLS-enabled services.</p> <div> <p>Note: It is not a best practice to enable this field for a production environment.</p> </div> <p>Default: False</p>
Close Connection	<p>Select True if you want to terminate the connection to the server after the response is processed. This affects the performance as the connection is no longer cached.</p> <p>Default: False</p>
Follow Redirects	<p>If set to false, the HTTP client does not follow any redirects automatically. It returns the initial response.</p> <p>Default: True</p>
Timeout	<p>Specify the timeout period (in milliseconds) for invoking a service. If a</p>

Field	Description
	<p>timeout value is specified, the Activity waits for the specified time. If the response is not received by the specified time, the request expires with an error.</p> <p>Default: 0 milliseconds (that is, there is no timeout for invoking a service)</p>
Request Type	<p>Note: This field is displayed only for the POST, PUT, and PATCH methods.</p> <p>The Request content type of the REST service. The following content-type are supported:</p> <ul style="list-style-type: none">• text/plain• application/json• application/x-www-form-urlencoded• multipart/form-data

Field	Description
-------	-------------

Note:

- If you select application/x-www-form-urlencoded, the default schema is set in the **Request Schema** field of the **Input Settings** tab. You can edit the default schema or specify your schema. If you specify your schema, it must be a name-value string pair.
- For the application/x-www-form-urlencoded request type, use the url.queryEscape function to get the expected input at the server.



- If you select the multipart or form-data as a request type, then you can pass the content-type for the file and files options in the Input.
- If you do not set the content-type, then the application/octet-stream is set as the default content-type for file and files input.

Proxy	Specify the URL to the HTTP proxy server. If a proxy URL is specified, the request to the REST service (specified in the URL field) is routed via this proxy URL.
-------	--

Note: A secure connection to the proxy server is not supported.



Default: Proxy URL is disabled.


Input Settings



Note:

- If you upload a JSON file in the **API Spec** field, the fields in **Input Settings** are automatically populated according to the **Resource Path** you select.
- The InvokeRESTService supports gzip and deflate encoding.

Field	Description
Query Params	<p>Query parameters to be appended to the path. To add the query parameters, click  and press Enter to save your changes.</p> <ul style="list-style-type: none"> • parameterName: Name of the query parameter. • type: The data type of the query parameter. Supported types are string, number, boolean, and object. <div style="background-color: #f0f0f0; padding: 5px; margin: 10px 0;"> <p>Note: Only a simple JSON object type is supported.</p> </div> <ul style="list-style-type: none"> • required: Set to True if the query parameter is a required configuration. The trigger reports an error if no values are provided to the required query parameter.
Path Params	<p>Path parameters that are appended to the path. This is a non-editable field.</p> <ul style="list-style-type: none"> • parameterName: Name of the path parameter. This is the parameter specified in between { } in the Resource Path field or the URL field in Settings. • type: The data type of the path parameter. The Supported type is string.
Request Headers	<p>Header values for the InvokeRESTService Activity. To add the header parameters, click  and press Enter to save your changes.</p> <ul style="list-style-type: none"> • parameterName: Name of the header parameter. • type: The data type of the header parameter. Supported types are string, number, and boolean.

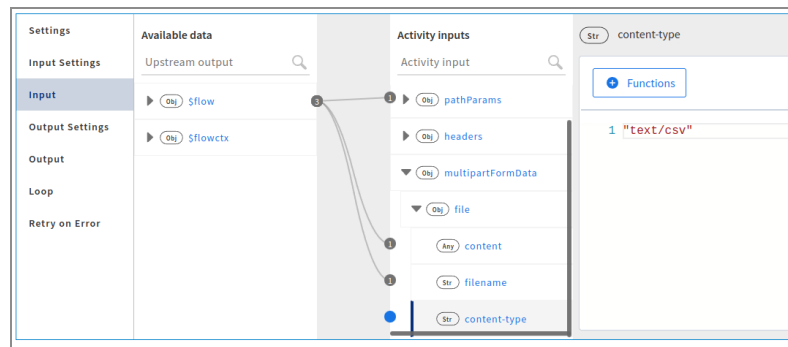
Field	Description
	<ul style="list-style-type: none"> • required: Set to True if the header parameter is a required configuration. The trigger reports an error if no values are provided to the required header parameter.
Request Schema	<p>Enter a request schema here. This field is visible only if you selected the POST, PUT, PATCH, or DELETE method on the Settings tab.</p> <div> <p>Note:</p> <ul style="list-style-type: none"> • For the DELETE method, specifying the request schema here is supported for manual configuration only and not when configured with API specification. • If you selected application/x-www-form-urlencoded as the Request Type on the Settings tab, the default schema is set here. You can edit the default schema or specify your schema. If you specify your schema, it must be a name-value string pair. </div>
Multipart Data	<p>This field is displayed in place of the Request Schema field if you select multipart/form-data as the Request Type on the Settings tab. Click  to add the parameters.</p> <ul style="list-style-type: none"> • Name: Name of the parameter. • Type: The supported types are string, object, and filecontent, file, and files. <ul style="list-style-type: none"> ◦ For file types, such as images and PDF files, convert your file to Base64 format and enter the encoded value as the Input. ◦ To send the file name and the file content in the same request, use file or files types. With file type, you can send one file and a file name. With the type files, you can send multiple file contents for one file name. • Required: Check the box if the parameter is a required configuration. • Schema: Enter the JSON schema in this field if the Type is an object. <div> <p>Note: If you want to pass dynamic data to a multipartFormData field, no data must be defined in the MultipartData table. Otherwise, the dynamic data is not honored and only table field values are considered.</p> </div>

Input

i **Note:** If you upload a JSON file in the **API Spec** field, the **Input** fields are automatically populated according to the **Resource Path** and **Method** you select.

Field	Description
host	Specify the value that must override the hostname:port value specified in the URL at runtime with the value specified in this configuration. Enter a value in the form hostname[:port] where [:port] is optional.
queryParams	Provide a value to the query parameters configured in the Input Settings section.
pathParams	Provide a value to the path parameters defined as part of the URL on the Settings tab.
headers	Header values for the Activity. These values can be manually entered or mapped to the output of the trigger or any preceding Activity.
dynamicRequestHeaders	<p>An array of additional header parameters to add request headers at runtime when invoking a REST Service. The header includes the following information:</p> <p>name: the name of the header</p> <p>value: the value of the header</p> <p>These values can be entered manually or mapped to the output of the trigger or any preceding Activity.</p> <p>Note: If a header is defined in both the header and dynamicRequestHeader (mapped or dynamically provided) fields, then the value from dynamicRequestHeader takes precedence.</p>


Field	Description
body	Request Schema values for the Activity. These values can be manually entered or mapped to the output of the trigger or any preceding Activity. This field is visible only if you selected the POST, PUT, PATCH, or DELETE method on the Settings tab. <div> Note: For the DELETE method, specifying the request schema here is supported for manual configuration only and not when configured with API specification. </div>
multipartFormData	This field is visible if you select file or files type in the Multipart Data field on the Input Settings tab. <p>Each file that you enter in the Multipart Data table appears here. content, filename and content-type fields are displayed for each file.</p>



Output Settings

Note: If you upload a JSON file in the **API Spec** field, the fields in **Output Settings** are automatically populated according to the **Resource Path** you select.

Field	Description
Configure Response	This allows you to configure response codes.

Field	Description
Codes	<p>Default: False (See "Response Schema" and "Response Type" in this table.)</p> <p>To specify a response code, select True and click . Enter the following details:</p> <ul style="list-style-type: none"> • Code: Enter a specific response code or configure a single schema for a category of response codes. For example, if all the status codes are similar (such as 501, 502, 503), you can define a single schema (as 5xx) for them. Defining a single schema saves you time and effort as you do not need to configure each status code separately in the Activity. <div style="border: 1px solid #ccc; padding: 10px; margin: 10px 0;"> <p>Note: If the status code is provided as a range (5xx in the above example) and also in an absolute format (501 in the above example), the status code in the absolute format is given priority. In the above example, status code 501 is given priority over 5xx at runtime.</p> </div> <ul style="list-style-type: none"> • Type: Select the type of response expected for the Code. Supported types are String and Object. • Response Body: If the Object is selected as the Type, enter the JSON schema in the Response Body column. For String, you need not enter anything in the Response Body column. • Response Headers: Specify the response header corresponding to the response code. • Actions: The actions displayed change based on the type of the response code. <ul style="list-style-type: none"> ◦ Edit, Delete: For the Object type of response, you can edit the details or cancel it. ◦ Save, Cancel: For a String type of response, you can save or cancel the changes. <p>The response codes appear on the Output tab.</p>
Throw Error	<p>If set to True, the flow execution goes to the Error Handler for an error response of 400 or higher. The flow continues running for a Success response code of less than 400.</p> <p>Default: False</p>

Field	Description
Response Schema	<p>Note: This field is displayed only when Configure Response Codes is set to False.</p> <p>The schema for the reply that the server sends.</p>
Response Type	<p>Note: This field is displayed only when Configure Response Codes is set to False.</p> <p>The content type of the REST service. The following content types are supported:</p> <ul style="list-style-type: none"> • application/json • application/xml • text/plain • other <p>Default: application/json</p> <p>JSON to XML conversion is not supported by the REST Activity. Any service that requires data in XML format cannot be invoked after providing data in the JSON format using REST Activity.</p> <p>If the content type is other than the types application/json, application/xml, or text/plain, it is converted into Base64 encoded values. You can use the <code>utils.decodeBase64</code> function to get the actual values. This is applicable when Configure Response Codes is set to True or False.</p>
Output Format	<p>Note: This field appears only when the Response Type is set as application/xml.</p> <p>The format of the requested content for the application/xml response type. The following formats are supported:</p> <ul style="list-style-type: none"> • JSON Object • XML String
Response	Sample JSON schema for the response that the REST service returns.

Field	Description
Schema	<p>The JSON schema in this field is editable for the following settings only:</p> <ul style="list-style-type: none"> When the Response Type is set as application/json When the Response Type is set as application/xml and the Output Format is set as JSON Object
Response Headers	<p>The header parameters for the reply.</p> <div> Tip: If you want to fetch a cookie coming with a response, add a new row with Set-Cookie as the parameter. You can also map this parameter to subsequent activities in the flow. </div>

Output

The Output tab displays the headers and responsebody configured for the response in a tree format. The responseTimeInMilliSec parameter specifies the time taken to receive the response in milliseconds.

Loop

For information on the Loop tab, see [Using the Loop Feature in an Activity](#).

Retry on Error

See [Using the Retry On Error Feature in an Activity](#)




Note: To update all these settings for an Activity configured from Swagger 2.0 or OpenAPI 3.0 specification, change the API specification file and upload it to **Settings**. Do not update the Activity settings as manual updates are removed.

Using SSL

If you choose to set up SSL authentication for the **InvokeRESTService** Activity, you must have a self-signed server certificate that you can upload under the **CA/Server Certificate** field when setting up the Activity. Alternatively, you can copy the certificate under the

system CA certificates. The Activity loads all the system certificates and then appends any user-configured certificates in the activity settings.

 **Note:** Use a self-signed PEM certificate for a secure connection.

To set up SSL authentication, perform the following steps:

Before you begin


You must have the self-signed server certificate ready.

Procedure

1. On the flow page, click the **Invoke REST Service** tile to open its properties.
2. On the **Settings** tab, under **Use certificate for verification**, select **True**.
This displays **Browse**. The SSL verification is turned off when **Use certificate for verification** is set to **False**.

3. Use **Browse** to navigate to the location of the server certificate.

Once the server certificate is uploaded successfully in the Activity or copied under system CA certificates, the connection uses this certificate to authenticate before invoking the SSL service.

 **Note:** Due to the Go language API limitations, this feature is supported only on Linux and macOS and not on Windows.

LogMessage

LogMessage is an activity that writes a message to the log. For each app, there is a log file. You can view the logs in the **Log** tab. The log messages generated are independent of the engine log level. It is also independent of the log level set by the PAPI logger or overridden using the environment variables.

Settings

The **Settings** tab has the following fields.

Field	Description
Log Level	<p>Select one of the following log levels:</p> <ul style="list-style-type: none"> • Info: logs informational messages highlighting the app progress. • Warning: is the warning message of an unexpected error when running the flow. • Error: logs error conditions and messages. • Debug: can be used for debug-level messages.
Add Flow Details	<p>Appends Flow Instance ID, Flow Name, and Activity Name to the Log Message.</p> <p>By default, this field is set to False.</p>

Input

Provide the following input for this Activity.

Input Item	Description
message	The message to be displayed in the log.
logLevel	<p>The logLevel to be set for an Activity.</p> <p>Valid Values: INFO, DEBUG, ERROR, WARN</p> <p>While mapping,</p> <ul style="list-style-type: none"> • You can bind the log level from the App Properties or the App level schema. • If you enter any value other than the values that are available in the settings level, you get an error as 'Invalid Log Level'. • The log level value set in mapper level takes precedence over the log level value set in settings level.

Loop

For information on the Loop tab, see [Using the Loop Feature in an Activity](#).

Mapper

Use this Activity to define a schema to get the desired data. This Activity is particularly useful to define a schema for an object of this type any. In the flow, you place the Mapper Activity preceding an Activity whose input requires an object of data type any. This allows you to map the object of type any to the output from the Mapper Activity. An advantage of using this Activity is that you can construct the data for any data type within the flow instead of fetching it from outside.

Input Settings

Field	Description
Input Schema	<p>Enter the JSON schema that is used as the input for this Activity. The elements of this schema are available for mapping on the Input tab and are mappable to the output from any preceding Activity, trigger, or flow input.</p> <p>The Mapper Activity outputs the elements from this schema, so they are also displayed on the Output tab in a tree format. This makes them available for mapping in the following activities.</p>

Input

The **Input** tab displays the schema that you entered on the **Input Settings** tab in a tree format. You can map these elements to the output from any preceding Activity, trigger, or flow input.

Output

The **Output** tab displays the elements from the schema that you entered on the **Input Settings** tab.

Loop

For information on the Loop tab, see [Using the Loop Feature in an Activity](#).

ParseJSON

This Activity takes stringified JSON data as input and converts it into a JSON object, which can then be accessed by the downstream activities that follow. You provide the input to the Activity either by entering the stringified JSON data manually on the **Input** tab or saving it in a file and entering the file path on the **Input** tab. The Activity supports output validation if you opt to validate the input JSON data against the output schema that you configure on the **Output Settings** tab.

Settings

Field	Description
Output Validation	<p>True: Select True to validate that the JSON data in your input string matches the schema that you configure on the Output Settings tab of the Activity.</p> <p>False: Select False to skip the validation of the JSON data in your input string against the schema you configured.</p> <p>This field can be configured with an app property.</p>

Input

Field	Description
jsonString	<p>The string containing the JSON data that you want to parse. This Activity creates a JSON object with the parsed JSON data. Enter the string manually or map it to an element from the output of the trigger, flow input, or one of the preceding activities.</p> <p>You have the option of saving the string in a file and pointing to the file. If you use a file as an input, you must provide the file path here. The path must be prefixed with <code>file://</code> for example, if the path to your file is <code>/users/<yourname>/myfile.json</code>, you would enter <code>file:///users/<yourname>/myfile.json</code></p>

Output Settings

Field	Description
Schema	The schema that you want to use to create the JSON object. You have the option to validate the stringified JSON (input to the Activity) against this schema.

Output

The output schema is displayed in a read-only tree format.

Loop

For information on the Loop tab, see [Using the Loop Feature in an Activity](#).

ReplyToTrigger

This activity is placed under the **General** category. Use this Activity to map output values and to send the reply to the trigger. The flow execution continues after the reply is sent to the trigger, unlike the **Return** Activity.

When using this Activity, keep the following considerations in mind:

- The flow in which you want to add the **ReplyToTrigger** Activity must have a trigger.
- The input of the Activity must be consistent with the flow output schema.
- Do not use this Activity with subflows. To send a reply from subflows, use the **Return** Activity.
- If the flow output schema is updated, you must open the Activity once so that the update reflects in the Activity.
- You can add multiple **ReplyToTrigger** activities in one flow but only the first Activity sends the reply to the trigger.

ProtobufToJSON

This Activity is placed under the **General** category. Use the **ProtobufToJSON** Activity to convert protocol buffer messages to JSON format.

For information about the .proto files, see the [proto3 Language Guide](#).

Settings

Field	Description
Proto File	Click Browse and select a .proto file. The file must contain only the proto3 syntax.
Message Type Name	All the message types from the .proto file are provided as list options. Select the message type from the list whose corresponding proto3 message you want to convert.
Include Default Values	The protocol buffer message fields that have null values are excluded when converting to another format. Set this field to True to include the message fields with their default values in the JSON format output. Default: False

Input

Enter the **protoMessage** as the Activity input. The **protoMessage** must be a Base64-encoded string.

Output

The **Output** tab displays the names of the fields for the selected message type from the .proto file.

Loop

For information on the Loop tab, see [Using the Loop Feature in an Activity](#).

No-Op


This Activity is placed under the **Default** category. You can use the **No-Op** Activity to implement branching of activities in the **Main flow** and the **Error handler**. The **No-Op**

Activity has no input or output fields. It does not affect the values of the previous Activity. You can insert this Activity at any position in the flow and also use it with subflows.

Notify

The **Notify** activity allows a flow instance to send data to a corresponding flow instance containing a **Wait for Notification** activity. The key specified in the input mappings creates the relationship between the **Notify** activity and the corresponding **Wait for Notification** activity.

Input

Field	Description
key	The key to coordinate a Notify activity with the corresponding Wait for Notification activity.
notifyAll	<p>When set to True, a notification is sent to all Wait for Notification activities waiting on the matching key.</p> <p>If set to False and multiple Wait for Notification activities are waiting for the matching key, then the notification is sent to any random Wait for Notification activity.</p> <p>Default: False</p>
data	<p>(Optional) Notification data to be sent for the given key.</p> <p>To configure a schema, hover over this field, click the  icon, and select the Coerce with schema option.</p>

SendMail

Use the **SendMail** Activity to send emails using an SMTP server.

i Note:

- To securely configure the **SendMail** Activity using the smtp.gmail.com server, use **TLS** on port 587 or **SSL** on port 465.
- The SendMail activity allows you to use the password of third-party apps as mandated by popular mailboxes such as Gmail, Yahoo, Outlook, and AOL. You cannot use your regular mailbox username and password with this activity.

Settings

The **Settings** tab has the following fields.

Field	Description
Server	The hostname or IP address for the mail server.
Port	The port used to connect to the server.
Username	The username to use when authenticating to the mail server.
Password	The password to use when authenticating to the mail server.
Connection Type	The type of connection to be used to communicate with the mail server. Select TLS or SSL depending on the security configuration of the mail server. In case no security is enabled on the mail server, select NONE .
Server Certificate	(Available only when Connection Type is set to SSL) The server or CA certificate to be used for the secure connection. The certificate must be PEM encoded.

Input

This tab displays the fields that are used as input for the Activity.

Input Item	Description
message_content_type	The type of message content. Valid types are "text/plain" or "text/html".
sender	The email address of the sender.
recipients	<p>The recipient list for the email.</p> <p>You can send the mail to multiple recipients. Provide a list of recipients in a single string by using a comma as the delimiter.</p>
cc_recipients	<p>The CC recipient list for the email.</p> <p>You can send the mail to multiple recipients. Provide a list of recipients in a single string by using a comma as the delimiter.</p>
bcc_recipients	<p>The BCC recipient list for the email.</p> <p>You can send the mail to multiple recipients. Provide a list of recipients in a single string by using a comma as the delimiter.</p>
reply_to	Email address to which the reply message is to be sent.
subject	The subject of the email.
message	The content of the email message.
attachments	<p>File attachments to be sent along with the email message.</p> <p>To map the child elements, add <code>array.forEach()</code> to the attachments field and then specify the child elements as follows:</p> <ul style="list-style-type: none"> file: Specify the path of the file to be attached using <code>file://<path></code> or specify the content of the file by enclosing it in double-quotes. <p>file: In Flogo, if the file is local to your container, specify the path of the file to be attached using <code>file://<path></code>. For example, if you have downloaded a file using the Box or Dropbox Activity, you can use <code>file://<path></code> to attach the file. Alternatively, enclose the contents of the file in double quotes and specify them in the file field.</p>

Input Item	Description
	<p>Note: In Flow Tester, file://<path> cannot be specified.</p> <ul style="list-style-type: none">• filename: Specify the name of the file to be attached.• base64EncodedContents: If the file is a Base64 encoded file, set this field to true. The default is blank (or false). <p>To send multiple attachments, use the Loop feature.</p>

Loop

For information on the Loop tab, see [Using the Loop Feature in an Activity](#).

SharedData

The **SharedData** Activity enables sharing of runtime data within a flow or across flows in an app. This Activity simplifies your flow designs. The advantage of using this feature is that you can set data anywhere in the main flow, subflow, or error handler and the data can be shared across the entire flow or app.

This Activity involves the following operations:

- Get: Retrieves data from the selected scope (either flow or app) based on a key.
- Set: Sets data for the selected scope based on a key.
- Delete: Deletes data from the app data.

For example, you can set the values in a parent flow. In a subflow, you can add the **SharedData** Activity and use the Get operation to access the values set in the parent flow.

Settings

The **Settings** tab has the following fields.

Field	Description
Scope	<p>Options are:</p> <ul style="list-style-type: none"> • Flow: Data can be shared within the flow instance and its subflow instances only. • Application: Data can be shared across flow instances within an app. <p>Default: Flow</p>
Operation	<p>The operation that needs to be performed. Options are:</p> <ul style="list-style-type: none"> • Get: Retrieve the data from the selected scope by key. • Set: Set the data for the selected scope by key. • Delete: (Available only if Scope is selected as Application) Optionally, you can delete app data based on the input key. For example, if you need data for one-time use only, you can delete the data to avoid storing it in memory unnecessarily. Otherwise, the data is deleted when you scale down or stop the app. <p>Default: Get</p>
Data Type	The data type of the shared data. Supported types are string, integer, number, and object.
Object Schema	(Only if you select Data Type as object) Specify the object's JSON data or schema in Object Schema . You can also specify an app-level schema by using the Use app-level schema option.

Input

This tab displays the fields that are used as input for the Activity.

Input Item	Description
key	<p>Available operations are:</p> <ul style="list-style-type: none"> • Set: Specify any value that you want to use while setting the data. • Get: If you want to retrieve data using the Get operation, you must use

Input Item	Description
	the same value that was specified while using the Set operation for setting the data for a scope. App properties can also be used to set or get the key.
input/data	(Set operation only) You can provide values of the Data Type selected on the Settings tab.

Output

This tab displays the output of the Activity. Note that the **Set** operation does not have any output.

Input Item	Description
exist	Indicates whether the data for the key specified on the Input tab exists.
data	Output data based on the input specified on the Input tab.

Loop

For information on the Loop tab, see [Using the Loop Feature in an Activity](#).

Sleep

The **Sleep** Activity is an asynchronous Activity that suspends the execution of a flow for a specified time.

Settings

Field	Description
Interval	The unit of the time interval for which the execution of the flow must be

Field	Description
Type	suspended. Supported types are Millisecond , Second , and Minute . Default: Millisecond
Interval	The time interval for which the execution of the flow must be suspended. Default: 0

Input



Note: The fields on the **Input** tab are required only if you need to pass values from the output of a previous Activity or trigger. Otherwise, you can directly specify the values on the **Settings** tab. Values specified on the **Input** tab take precedence over values specified on the **Settings** tab, if values are configured in both the tabs.

Field	Description
Interval Type	The unit of the time interval for which the execution of the flow must be suspended. Supported types are Millisecond , Second , and Minute . Default: Millisecond
Interval	The time interval for which the execution of the flow must be suspended. Default: 0

Loop

For information on the Loop tab, see [Using the Loop Feature in an Activity](#).

Throw Error

Throw Error is an Activity that throws an error when executing a flow. Depending on your business logic, you can also use this Activity along with the error handler and branching conditions.

Input

The **Input** tab has the following fields.

Field	Description
message	The message element is used to specify the error message string to be output.
data	The data element is used to configure any data related to the error.

Loop

For information on the Loop tab, see [Using the Loop Feature in an Activity](#).

Wait for Notification

The **Wait for Notification** activity suspends the execution of the flow instance and waits for a **Notify** activity with a matching key to be executed in another flow instance. The key specified in the input mapping creates a relationship between the **Wait for Notification** activity and the corresponding **Notify** activity.

Input

Field	Description
key	The key to coordinate a Wait for Notification activity with the corresponding Notify activity.
timeout	<p>The timeout (number of milliseconds) for this Wait for Notification activity. Set it to 0 for no timeout.</p> <p>Default: 60 seconds</p> <div>Note: When the first Wait for Notification activity times out for the given key, all the waiting activities for that key are timed out. This is a technical limitation as the individual timeout configuration of the waiting activities is not considered. In such scenarios, configure Retry On Error for the Wait for Notification activity.</div>

Output

Field	Description
data	(Optional) Notification data to be received for the given key. To configure a schema, hover over this field and select the Coerce with schema (...) option.

Loop

For information on the Loop tab, see [Using the Loop Feature in an Activity](#).

Retry on Error

For more information, see [Using the Retry On Error Feature in an Activity](#).

JSONToXML

This Activity takes stringified JSON as input and converts it into an XML String, which can then be accessed by the downstream activities that follow.

JSONtoXML Conversion Limitation

The difference between XML and JSON imposes the following limitation:

- JSON is a map-like structure with key-value pairs and XML stores data in a tree structure with namespaces for different data categories. So the input JSON without any tree structure does not result in a valid XML.

Input

Field	Description
jsonString	This field takes stringified JSON data. We can directly pass stringified JSON data manually or map the textContent coming from previous activity (for example, GET JSON file from S3).

Output

Display output as read-only jsonObject.

XMLToJSON

This Activity takes stringified XML data as input and converts it into a JSON object, which can then be accessed by the downstream activities that follow.

XMLtoJSON Conversion limitations

The differences between XML and JSON impose some of the below limitations.

- The XML element and attribute names should not contain any delimiter used in JSON
- XML comments (<!-- comment -->) are ignored in the JSON document
- DTD declarations are ignored.
- XML processing instructions are ignored.
- All XML element/attribute values are transformed to a JSON string as the conversion is not schema (xsd) aware. Set typecast to true to convert string data starting with number to integer and string data starting with true/false to boolean. Set ordered field as true to set the key in alphabetical order in JSON.
- Entity references are ignored.
- XML attribute then while converting to json, hyphen (-) used as a prefix in json to indicate attribute and # indicate key.

Input

Field	Description
xmlString	This field takes stringified XML data. We can directly pass stringified xml data manually or map the textContent coming from previous activity (e.g. GET XML file from S3).
Ordered	This field takes boolean value. When set ordered to true, set key in JSON in alphabetical order.

Field	Description
typeCast	This flag controls the conversion of string data. When enabled (True): String data starting with a number is converted to an integer. String data starting with "true" or "false" is converted to a boolean.

Output Settings

Field	Description
Schema	Configure expected object structure using JSON schema or JSON sample.

Output

Display output as read-only xmlString.

Connections

Along with the activities and triggers, some connections are also available for general use. Connections contain the parameters that are needed for a client connection to an external data provider or interface. These details are then used by activities or triggers to connect at runtime.

HTTP Client Authorization Configuration

You can set up the **HTTP Client Authorization Configuration** connection from the **Connections** tab to add authentication and authorization to your Flogo apps. To enable the connection you have set up, refer to the [InvokeRESTService](#) Activity.

The connection has the following fields:

Field	Description
Name	Enter a name for the connection.

Field	Description
Description (optional)	Enter a description for the connection.
Authorization Type	<p>Select an authentication type.</p> <p>The connection supports two types of authorization:</p> <ul style="list-style-type: none"> • Basic • OAuth2
If you select Basic as the Authorization Type , the following fields are displayed:	
User Name	Enter a username for the connection.
Password (optional)	<p>Enter a password for the connection.</p> <p>Some services can send authentication data with username only. In such cases, you need not provide any password.</p>
If you select OAuth2 as the Authorization Type , the following fields are displayed:	
Grant Type	<p>Indicates the method by which an app can obtain an access token.</p> <p>Select one of the following supported types:</p> <ul style="list-style-type: none"> • Authorization Code • Client Credentials
Callback URL	<p>The connection is redirected to this URL after authorization. Your app's Callback URL must match this URL.</p> <p>This is a read-only field.</p>
Auth URL	<p>Authorization server API endpoint. For example, the Google authorization URL is: https://accounts.google.com/o/oauth2/v2/auth</p> <p>This field is an app-property enabled field.</p>
Additional Auth	Additional query parameters to get the refresh token based on the

Field	Description
URL Query Parameters (optional)	<p>service you request for. For example:</p> <p>access_type=offline&prompt=consent</p> <p>token_access_type=offline</p>
Access Token URL	<p>The token API endpoint is used to get access tokens. For example:</p> <ul style="list-style-type: none"> Google: https://oauth2.googleapis.com/token Salesforce: https://login.salesforce.com/services/oauth2/token <p>This field is an app-property enabled field.</p>
Client Id	<p>The client id of the OAuth2 app. You can change this value at runtime.</p> <p>This field is an app-property enabled field.</p>
Client Secret	<p>The client secret of the OAuth2 app. You can change this value at runtime.</p> <p>This field is an app-property enabled field.</p>
Scope	<p>Specifies the level of access that the app is requesting. You can specify multiple space-delimited values. For example:</p> <p>Salesforce - chatter_api refresh_token</p> <p>You can also set the value of this field from an application property at runtime.</p> <div> <p>Note: If Authorization Code is selected as Grant Type for the connection, you cannot override the scope value.</p> </div>
Audience	<p>The unique identifier of the audience for an issued token.</p> <p>The audience value is an app client ID for an ID token or the API that is being called for an access token.</p> <p>This field is an app-property enabled field.</p>

Field	Description
Client Authentication	<p>The method by which authentication parameters are sent. Based on the service request, you can send authentication parameters in Header, Body, or Query.</p> <ul style="list-style-type: none"> • Header - Indicates sending authentication parameter through headers. • Body - Indicates sending authentication parameters through the body with application/x-www-form-urlencoded. • Query - Indicates sending authentication parameters through query parameters.
Token	<p>Indicates the token, which is a Base64 encoded value with app property enabled.</p> <p>This is a read-only field.</p>



Note: For refresh tokens:

Only the standard OAuth2 workflow is supported.

The OAuth 2.0 service provider must also return refresh tokens when you obtain an access token from the OAuth flow. For information on obtaining refresh tokens, refer to your OAuth 2.0 provider. You can then add this information in the **Additional Auth URL Query Parameters** field or the **Scope** field. This is necessary for long-running apps where the access tokens may expire.

The refresh token operation only happens when the server returns HTTP status code 401.

File Category Trigger and Activities

You can use these activities to perform create, read, write, copy, remove, rename, and list files operation. It also provides activities to archive and unarchive files. You can use the trigger File Poller to watch the directory to poll the file events, such as Create, Write, Rename, Remove, and Move.

File Poller Trigger

The **File Poller Trigger** polls for files or directories when a change (Create, Write, Rename, Remove, Move file event) is detected in the specified directory. To use this trigger, you simply configure your File Poller Trigger settings and add it to the trigger pane and create a flow.

Trigger Settings

Field	Description
Handler Settings	
Polling Directory	The path and name of the directory to watch for the file events.
Include Sub-Directories	When this field is set to true, the trigger includes all sub-directories in the Polling Directory to watch for the file events.
File Filter	Files that match the regular expression will be watched. For example, enter <code>^abc[a-zA-Z0-9]*.xml\$</code> to poll the xml files starting with abc
Polling Interval (in milliseconds)	Name of the service defined in the .proto file. You must create one gRPC trigger for any specific .proto file. Any subsequent gRPC triggers using the same .proto file can select the service and method they need from the dropdown list.
Mode	<p>The type of listing you want to retrieve. You can select from the following options:</p> <ul style="list-style-type: none"> • Only Files • Only Directories • Files and Directories (Default)
Poll File Events	<p>The file poller trigger polls for following events.</p> <ul style="list-style-type: none"> • Create • Write

Field	Description
	<ul style="list-style-type: none"> • Rename • Remove • Move <p>Select the checkbox next to these event you want to poll inside the specified directory.</p> <p>For example: If the Create event is not selected, the newly created files or directories are ignored while polling for any change.</p>

Map to Flow Inputs

You can go to this tab to map the trigger output to flow input. The output for the activity is described in the following table.

Output Item	Description
action	The occurred event to trigger the File Poller activity. The possible values are: Create, Write, Rename, Remove, or Move.
fileMetadata	This object contains the fullPath, name, oldPath, size, mode, modTime and isDir.

Create File

The Create File activity creates a new file or directory with the specified name at specified location.

Settings

Field	Description
Is a Directory	When this field is set to true, the activity creates a directory instead of a file.
Create Non-Existing Directories	When this field is set to true, the activity creates all directories in the specified path, if they do not already exist. If this field is set to false with non-existing one or more directories in the specified path, it throws an error.

Input

The **Input** tab displays the following fields.

Field	Description
fileName	The path and name of the file to create. Set the Is a Directory field to true on the Settings tab to specify that it is a directory you want to create.
overwrite	<p>Set this field to true to overwrite the existing file with the same name if it exists. In case of directory, overwrite happens only if existing directory is empty.</p> <p>If the specified file or directory exists and this field is set to false, then the activity throws an error.</p>

Output

The output schema displays all the metadata fields of a new file/directory in a read-only format. The metadata object contains fullPath, name, size, mode, modTime and isDir fields.

Loop

For information on Loop, see the "Using the Loop Feature in an Activity" section in the *TIBCO Flogo® Extension for Visual Studio Code User Guide*.

Read File

The Read File activity is used to read a file and place its contents into the output of the activity.

Settings

Field	Description
Read as	Representation type of content in the file you want to read. You can select from the following options: <ul style="list-style-type: none">• Text• Binary With 'Text', output 'textContent' will hold the plain string. With 'Binary', output 'binaryContent' will hold the base64 encoded string.
Uncompress	This field specifies whether to uncompress the file using GUnZip format. Select GUnzip to unzip the file. Choose None for no decompression.

Input

The **Input** tab displays the following fields.

Field	Description
fileName	The name and path of the file to read. For example: /opt/flogo/filename

Output

The output schema displays the metadata fields and the file content of the file in a read-only format. This object contains fileContent in either text or binary and metadata fields, such as fullPath, name, size, mode, modTime, and isDir.

Loop

For information on Loop, see the "Using the Loop Feature in an Activity" section in the *TIBCO Flogo® Extension for Visual Studio Code User Guide*.

List Files

The List Files activity returns information about files or directories, or a listing of all the files in the specified directory.

Settings

Field	Description
Mode	The type of listing you want to retrieve. You can select from the following options: <ul style="list-style-type: none">• Only Files• Only Directories• Files and Directories (Default)

Input

The **Input** tab displays the following fields.

Field	Description
fileName	<p>The path and name of the file or directory to monitor. You can also use wildcard characters to monitor a directory for files that match the provided specification.</p> <p>For example, <code>flogo/files/*.log</code></p> <p>It lists the files directory that have a .log extension</p>

Output

The output schema displays all the metadata fields of a file/directory in a read-only array format. The metadata object contains `fullPath`, `name`, `size`, `mode`, `modTime`, and `isDir` fields.

Loop

For information on Loop, see the "Using the Loop Feature in an Activity" section in the *TIBCO Flogo® Extension for Visual Studio Code User Guide*.

Write File

The Write File activity writes content to the specified file.

Settings

Field	Description
Write as	<p>Representation type of content in the file you want to write. You can select from the following options:</p> <ul style="list-style-type: none">• Text• Binary <p>With 'Text', input textContent will hold the plain string. With 'Binary', input binaryContent will hold the base64 encoded string.</p>
Create Non-Existing Directories	<p>When this field is set to <code>true</code>, the activity creates all directories in the specified path, if they do not already exist. If this field is set to <code>false</code> with non-existing one or more directories in the specified path, it throws an error.</p>
Compress	<p>This field specifies whether to compress the file using GZip format. Specify None for no compression, and GZip for a compressed output file.</p> <div><p>Note: When you specify GZip for this field, rename the file to use the <code>.gz</code> suffix and use <code>gunzip</code> to decompress the file.</p></div>

Input

The **Input** tab displays the following fields.

Field	Description
fileName	The name and path of the file or directory. For example: /opt/flogo/filename
overwrite	Set this field to true to overwrite the existing file with the same name. If the specified file exists and this field is set to false, then the activity appends the content to an existing file.
textContent	The contents of the file (text files). This field is present when Write as is set to Text.
binaryContent	The base64 encoded content of the file (binary files). This field is present when Write as is set to Binary.

Output

The output schema displays all the metadata fields of a file in a read-only format. The metadata object contains fullPath, name, size, mode, modTime, and isDir fields.

Loop

For information on Loop, see the "Using the Loop Feature in an Activity" section in the *TIBCO Flogo® Extension for Visual Studio Code User Guide*.

Copy File

The Copy File activity copies files and directories to a new location.

Settings

Field	Description
Create Non-Existing Directories	When this field is set to true, the activity creates all directories in the specified path, if they do not already exist. If this boolean is set to false with non-existing one or more directories in the specified path, it throws an error.
Include Sub-Directories	When this field is set to true, the activity includes all sub-directories in the source directory, when the source to copy is a directory.

Input

The **Input** tab displays the following fields.

Field	Description
fromFileName	<p>The path and name of the file or directory to copy.</p> <p>For example, to copy a directory, specify <code>/opt/flogo/myDirectory</code>.</p> <p>To copy all text files in a directory, specify <code>/opt/flogo/myDirectory/*.txt</code>.</p>
toFileName	The destination for the copy operation. For example: <code>/opt/flogo/copyfilename</code>
overwrite	<p>Set this field to true to overwrite the existing file with the same name, if it exists. In case of directory, overwrite happens only if existing directory is empty.</p> <p>If the specified file or directory exists and this field is set to false, then the activity throws an error.</p>

Loop

For information on Loop, see the "Using the Loop Feature in an Activity" section in the *TIBCO Flogo® Extension for Visual Studio Code User Guide*.

Rename File

The Rename File activity is used to rename file or directory.

Settings

Field	Description
Create Non-Existing Directories	When this field is set to true, the activity creates all directories in the specified path, if they do not already exist. If this field is set to false with non-existing one or more directories in the specified path, it throws an error.

Input

The **Input** tab displays the following fields.

Field	Description
fromFileName	The path and name of the file or directory to rename. For example, to rename a directory, specify <code>/opt/flogo/myDirectory</code>
toFileName	The new name and path of the file or directory. For example, <code>/opt/flogo/newfilename</code>
overwrite	Set this field to true to overwrite the existing file with the same name when renaming. If the specified file or directory exists and this field is set to false, then the activity throws an error. Note: Overwrite feature is only applicable for files.

Output

The output schema displays the metadata fields of a new file/directory in a read-only format. The metadata object contains `fullPath`, `name`, `size`, `mode`, `modTime`, and `isDir` fields.

Loop

For information on Loop, see the "Using the Loop Feature in an Activity" section in the *TIBCO Flogo® Extension for Visual Studio Code User Guide*.

Remove File

The Remove File activity removes the specified file or directory.

Settings

Field	Description
Remove Directory Recursively	When this field is set to true, this activity will delete the entire directory and all its subfolders and files recursively.

Input

The **Input** tab displays the following fields.

Field	Description
fileName	The path and name of the file or directory to remove. For example: /opt/flogo/filename

Output

The output schema displays the metadata fields of a new file/directory in a read-only format. The metadata object contains fullPath, name, size, mode, modTime, and isDir fields.

Loop

For information on Loop, see the "Using the Loop Feature in an Activity" section in the *TIBCO Flogo® Extension for Visual Studio Code User Guide*.

Archive Files

The Archive Files activity is a synchronous activity that archives content to the specified file destination.

Settings

Field	Description
Archive Type	This field specifies the type of archive file format that supported by this activity. As of now only ZIP archive file format is supported.

Input

The **Input** tab displays the following fields.

Field	Description
sourceFilePath	The path and name of the file or directory to archive. For directories, you must specify an absolute path. For example: /opt/flogo/myDirectory
destinationPath	The name and location of the archive file or directory where you want the compressed file. For example: /opt/flogo/filename

Output

The output schema displays the destination path of the new archive file in a read-only string format.

Loop

For information on Loop, see the "Using the Loop Feature in an Activity" section in the *TIBCO Flogo® Extension for Visual Studio Code User Guide*.

Unarchive Files

The Archive Files activity is a synchronous activity that archives content to the specified file destination.

Settings

Field	Description
Archive Type	This field specifies the type of archive file format that supported by this activity. As of now only ZIP archive file format is supported.

Input

The **Input** tab displays the following fields.

Field	Description
sourceFilePath	The path and name of the file or directory to archive. For directories, you must specify an absolute path. For example: <code>/opt/flogo/myDirectory</code>
destinationPath	The name and location of the archive file or directory where you want the compressed file. For example: <code>/opt/flogo/filename</code>

Output

The output schema displays the destination path of the new archive file in a read-only string format.

Loop

For information on Loop, see the "Using the Loop Feature in an Activity" section in the *TIBCO Flogo® Extension for Visual Studio Code User Guide*.

Developing APIs

You can take an API-first development approach to implement APIs from a Swagger Specification 2.0, OpenAPI Specification 3.0, or GraphQL schema. You can upload an API specification file to a **ReceiveHTTPMessage** trigger or a GraphQL schema to a **GraphQL** trigger.

Using an OpenAPI Specification

To configure a flow with an API specification file, you can add a **ReceiveHTTPMessage** trigger to the flow and upload an API specification file.

Configure trigger: Receive HTTP Message Step 1 of 1

Step 1

Port

Configure Using API Specs ☒ True ☐ False

API Spec
 PetstoreOASApp_APISpec.json

Path


Method

When you implement a Spec, the **ConfigureHTTPResponse** and **Return** activities are automatically added to the flow. The mappings from trigger output to flow inputs get configured for you based on the definitions in the specification. The output of the **ConfigureHTTPResponse** activity is automatically mapped to the **Return** activity input. However, you must configure the input to the **ConfigureHTTPResponse** activity manually.

If you have multiple response codes configured in the REST trigger, the first response code is configured in the **ConfigureHTTPResponse** activity by default. However, if you have the 200 response code configured, then that is the default value in **ConfigureHTTPResponse** activity.

Considerations when using an API specification file to create a flow:

- Swagger Specification 2.0 and OpenAPI Specification 3.0 are supported.
- Currently, only the JSON format is supported.
- API specification files with cyclic dependency are supported in **ReceiveHTTPMessage** trigger and **InvokeRestService** activity. Schema properties with cyclic dependency are converted into empty schema objects. For example, if you have a type Book that contains an object element of the type Author. The type Author contains an element of the type Book, which represents the books written by the author. To retrieve the Author, it creates a cyclic dependency where the Author object contains the Book object and the Book type contains the Author object. Here, the Author type is converted to an empty object to support cyclic schema rendering in the mapper.
- String, integer, and boolean are the supported data types. A data type that appears in your specification but is not supported results in an error being displayed.
- Schema references within schemas are not supported.
- If the specification has a response code other than 200 (OK) or 500 (Error), the method that contains the unsupported response code is not created.
- You can enter a schema for the response code 200, but the 500 response code must be a string.
- Basepath element in the schema is not supported.

 **Note:** The REST reply data type is by default set to any data type. To configure the reply to an explicit data type, see [Configuring the REST Reply](#) section.

To create an app using an API specification and upload the specification file:

Procedure

1. Create a New Flogo app.
2. Open the created flow and drag the **ReceiveHTTPMessage** trigger to it.
3. Set the **Configure Using API Specs** radio button to True.
4. Browse for the API spec file. If the API spec file is valid, then the supported path and associated methods populate the dropdown menu.
5. In the dropdown menu, select the path and method and finish the trigger

configuration.

You have added the **ReceiveHTTPMessage** trigger with parameter values, Request body, and Response body from the selected API specification file.



Note: Available data and flow inputs are auto-mapped in **Map to Flow Inputs**. Available data and trigger reply are auto-mapped in **Map from Flow Outputs**.

To implement flows in an app using an API specification, see [Implementing Specs](#).

The supported Swagger 2.0 and OpenAPI Specification 3.0 features are given in the following list:

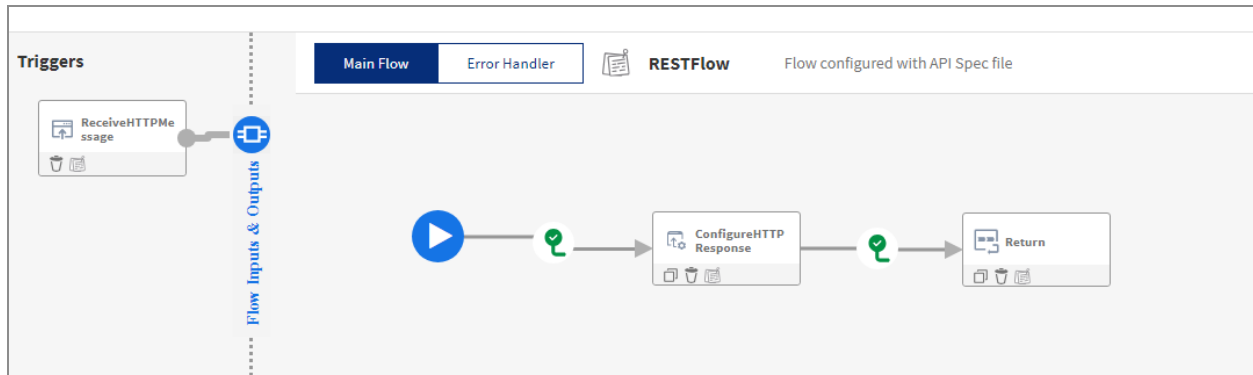
- Path Templating
- Media Type
 - Request Types: application/json, multipart/formdata, x-www-form-urlencoded
 - Response Types: text/plain, application/json
- Multiple Status Codes
- Path Item Object
- Parameter Object
- Request Body Object
- Reference Object
- Header Object
- Security Scheme Object
- allOf, oneOf, and anyOf keywords

For more information, refer to [OpenAPI Specification](#).

Configuring the REST Reply

When creating a REST app from a Swagger 2.0 or OpenAPI 3.0 API specification, the **ReceiveHTTPMessage** reply data type is set to any by default. You must explicitly configure the reply type.

To explicitly configure the reply type, add a **ConfigureHTTPResponse** Activity in the flow. This Activity must immediately precede the **Return** Activity in the flow.



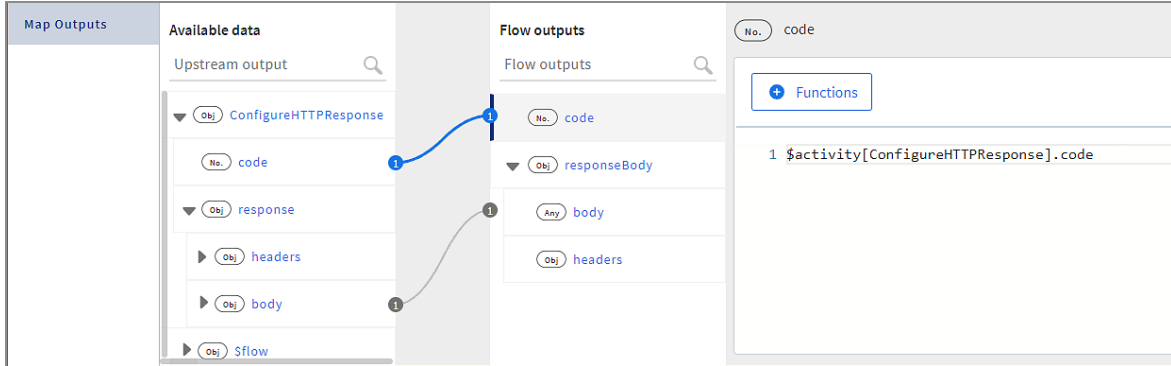
You can configure custom codes that you want to use in the HTTP reply on the **Reply Settings** tab of the **ReceiveHTTPMessage** trigger.

To configure your HTTP reply, perform the following steps:

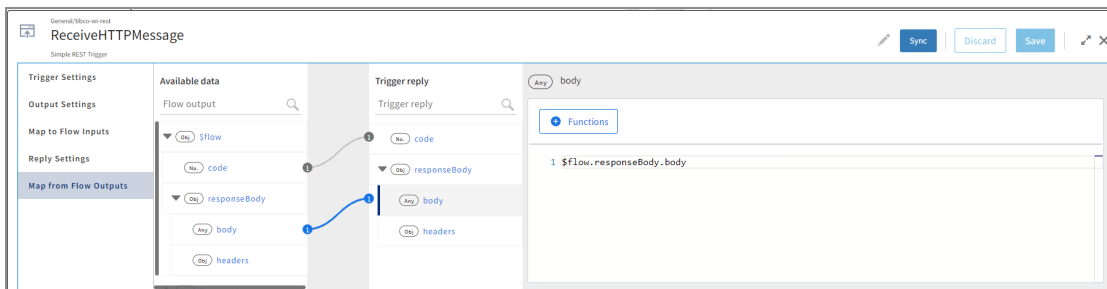
Procedure

1. Open the REST trigger configuration pane.
2. On the **Reply Settings** tab of the **ReceiveHTTPMessage** REST trigger, configure the custom codes that you want to use. For more information, see the "REST Trigger" section in the *TIBCO Flogo® Extension for Visual Studio Code User Guide*.
3. Add a **ConfigureHTTPResponse** Activity immediately preceding the **Return** Activity in the flow.
4. Open the **ConfigureHTTPResponse** Activity by clicking it and configure it as follows:
 - a. On the **Settings** tab:
 - i. If your flow is attached to multiple REST triggers, select the trigger in which you have configured the code you want to use from the **Trigger Name** dropdown menu. The **Trigger Name** field does not display if your flow is attached to only one REST trigger.
 - ii. Select a response code from the **Code** field menu. Only the codes configured in the selected trigger are displayed in the menu.
 - b. The **Input** tab displays the schema for the response code. Map the elements or manually enter a value for the elements.
 - c. Click **Save**.
5. Configure the **Return** Activity by mapping the **code** and **body** (which is currently of

data type any).



6. Click **Save**.
7. On the **Map from Flow Outputs** tab in the **ReceiveHTTPMessage** trigger, map the **code** and **body** to the corresponding elements from the flow output.



8. Click **Save**.

Using GraphQL Schema

GraphQL provides a powerful query language for your APIs enabling clients to get the exact data that they need. It can get data from multiple resources in a single request by aggregating the requested data to form one result set. GraphQL provides a single endpoint for accessing data in terms of types and fields.

The GraphQL trigger turns your app into a GraphQL server implementation. Each flow in the app acts like a GraphQL field resolver. So, the output of the flow must match the return type of the field in the schema.

You can create a flow, choose the GraphQL trigger, and drop it to the trigger panel. You must then select the type of GraphQL operation (query or mutation) and resolver. Save and sync your trigger input and output.

i Note: This section assumes that you are familiar with GraphQL. To learn about GraphQL, refer to the GraphQL documentation.

GraphQL server implementation

To obtain samples of GraphQL schemas and app JSON files, go to <https://github.com/project-flogo/graphql>.

The GraphQL schema must have either .gql or .graphql extension.

For more information, see [GraphQL Trigger](#).

Using App Properties, Schemas, and Specs

This section discusses how to create app properties, which you can use when populating field values. It describes how to create a schema that can be reused in your app. In addition, this section elaborates on app-level specs and how those can be used in your app.

App Properties

App properties provide a way to override property values after an app has been deployed to Data Plane. You can configure some supported fields with app properties when configuring triggers and activities. Connection-related app properties cannot be used for configuration anywhere within an app. Their only purpose is to allow you to change a connection value if there is a need during runtime. Configuration fields in your flow that require their values to be changed when the app goes from a testing stage to production are best configured using app properties instead of hard coding their values. App properties for triggers and activities reside within the app. App properties for connections are not modifiable from the **App Properties** tab in the app.


The URL field in an Activity is a good example of a field for which you would want different values, for instance, an internal URL when testing the app and an external URL when the app goes into production. You may want the URL used in the Activity to change when the app goes from a test environment to production. In such a case, it is best to configure the URL field in the Activity with an app property instead of hard-coding the URL. This way, you

can change the URL by changing the value of the app property used to configure the URL field.

Before deploying the app, you can change the default value of an app property from the [App Properties tab](#). Once you have deployed your app, you can change the default values of the [Environment Controls tab](#).

An app property value can have one of the following data types:

- string
- boolean
- number
- password

Values for the password data type are encrypted and are not visible by default. But when configuring the password value, you can click the **Show/Hide password property value** icon () to see the value temporarily to verify that it has been entered correctly.

When you deploy an app configured with app properties, the properties appear under the **Environment controls** tab in Data Plane. You can change the values for any property from this tab, but doing so requires a redeploying the app for the new values to take effect. Similarly, any property that is added, deleted, or updated from the **App Properties** tab within an app, is reflected in the **Environment controls** tab only after redeploying the app.

Creating App Properties

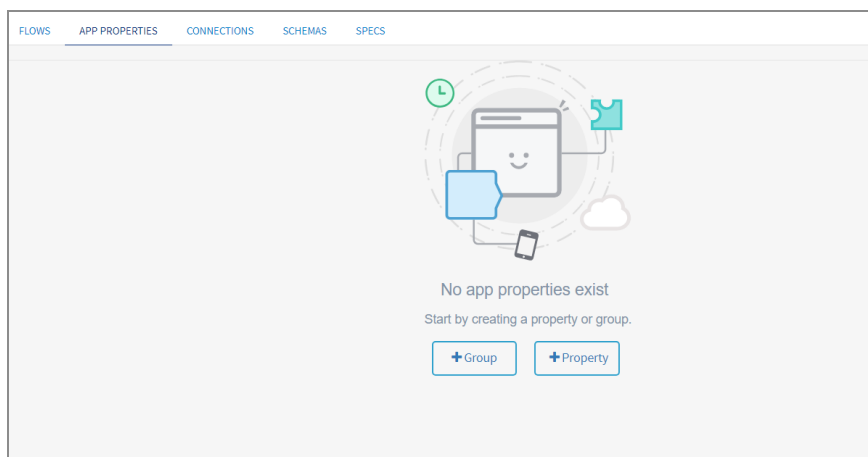
You can create an app property as a standalone property or as a part of a group. Use a group to organize app properties under a parent. A parent acts as an umbrella to hold related app properties and is labeled with a meaningful name. A parent does not have a data type associated with it. For instance, if you want to group all app properties associated with a particular Activity, you can create a group with a parent that has the Activity name and create all that Activity-related app properties under that parent.

As an example, you can create LOG_LEVEL as a standalone app property without a parent. Or you can create it as a part of a hierarchy such as LOG.LOG_LEVEL with the parent of the hierarchy being LOG and LOG_LEVEL being the app property under LOG. Keep in mind that if you group properties, you must refer to them using the dot notation starting from the parent. For example, the LOG_LEVEL property must be referred to as LOG.LOG_LEVEL. You can nest a group within a group.

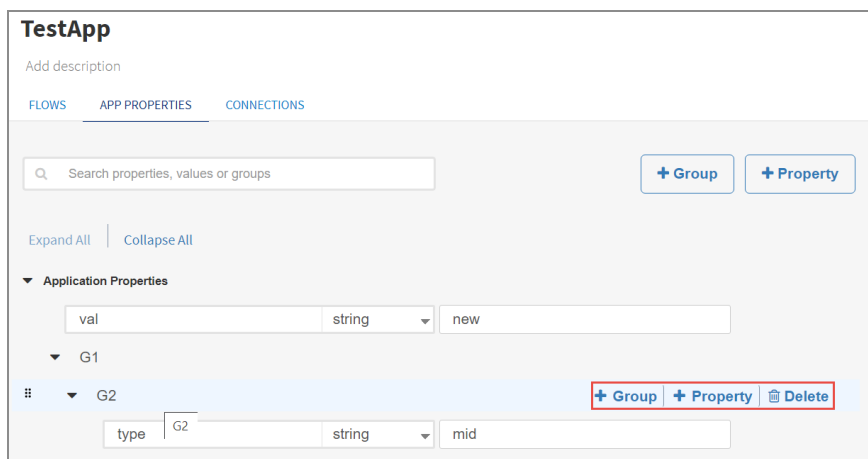
After you have created an app property, you must redeploy the app for the property to be available from the **Environment Controls** tab in the UI. The same is true if you modify an existing app property. You can modify app properties from the **Environment Controls** tab, but you need to redeploy the app for the modifications to take effect.

App Properties Tab

You can view existing app properties for an app in the **App Properties** tab. By clicking the **+Group** or **+Property**, you can add a group or a property and rename it. The following image shows an empty app properties tab.

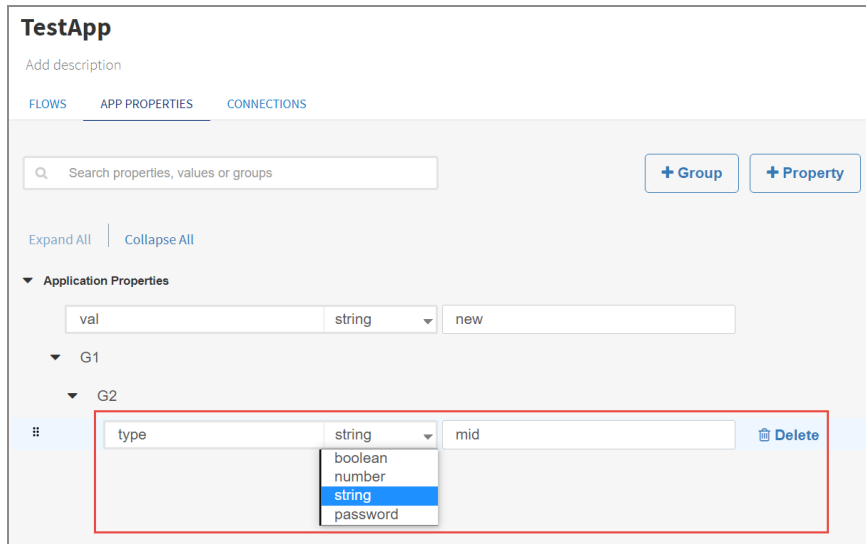


Nested groups and properties can also be created from the app properties tab by clicking the **+group** or **+property** of each group.



The name of the property added can be changed from default to anything you want. Even the type of property value can be changed by selecting it from the dropdown menu. You

can drag a property with unique names from one group to another but not within the same group.



Creating a Standalone App Property

To create a standalone app property for your app, follow the steps below.

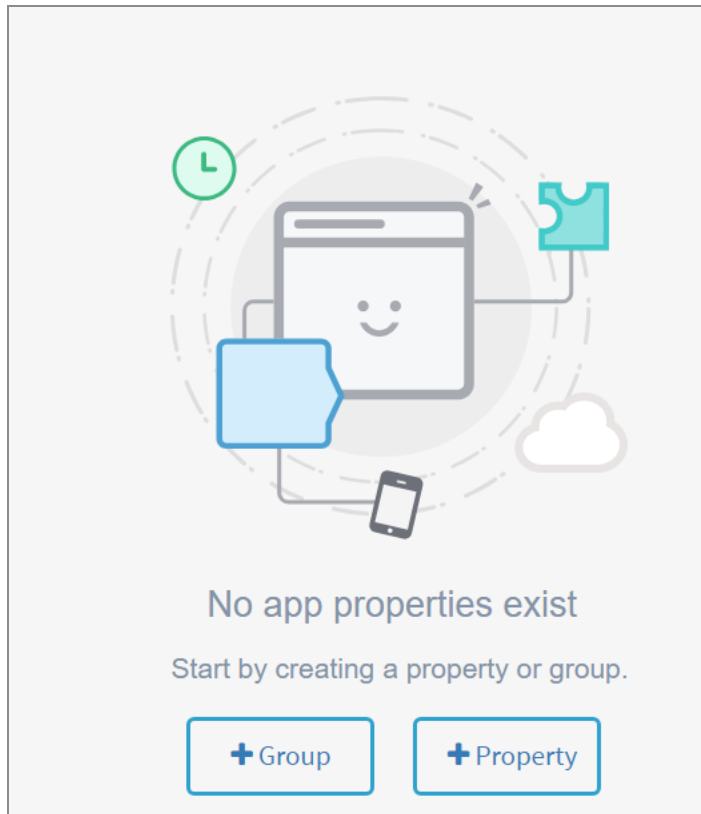
To create a group, see [Creating a Group](#).



Note: The standalone properties (properties that are not in a group) or the properties within the same group must have unique names.

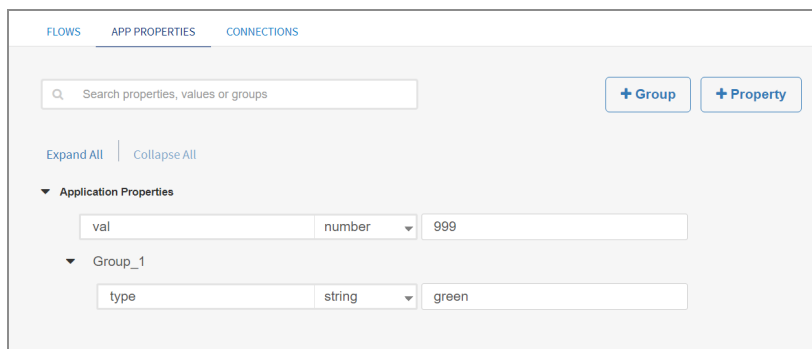
Procedure

1. If your app does not exist, create a new app, and click the **Property** button shown on the screen below.



If your app exists, then open the app details page and click **Property**. The **App Properties** tab opens.

If you already have existing properties, they are displayed. Click **+Property** to add another property.



2. Click the newly created property to make it editable and rename it. The property gets created.

Note: The property name must not contain any spaces or special characters other than a dash (-) or an underscore (_).

3. Select the data type for the new property from its dropdown list.
4. Enter a default value for the property in the text box next to the property.

Note: Only for certificates, the value must be of the format: <encoded_value>. To get the encoded value of the contents, you can use <https://www.base64encode.org/> or any other base64 encoding tool.

For example, for an SSL certificate, you can specify the app property as follows:

Note: You can secure an application running on Kubernetes by creating a secret that contains a Transport Layer Security (TLS) private key and certificate. This secret can be used by the client or service deployment at runtime.

5. Click the **Save** button to save your changes.

i Note: TIBCO Flogo® Enterprise runs validation in the background as you create a property. The validation considers the property type and default value of the property that you entered. The field is highlighted in red for user input until validation is successful for the entered value. Ensure you do not skip this step of saving your newly created property or group.

Procedure

1. Deploy (or redeploy) the app to the Data Plane for the property to be available from the **Environment Controls** tab in the UI.

Creating a Group

You can create one or more standalone app properties or group app properties such that they show up in a hierarchy. A group (or hierarchy) consists of a parent node, which is just a label and does not have a data type associated with it. You must create properties within the parent. You can do so in the **Application Properties** tab. When creating a group, you must add the parent first and then create the app properties under the parent.

i Note:

- With the drag option, a standalone property can be rearranged to another location or a property under the group can be moved to another group.
- A group with its nested groups and properties can be dragged to move from one location to another. Also a nested group can be moved up in the hierarchy or to the root level. However, no two groups can have the same name on the same level.
- Group names within an app must be unique. Also, property names within a group must be unique.
- You cannot create a group and an app property with the same name in the same hierarchy.

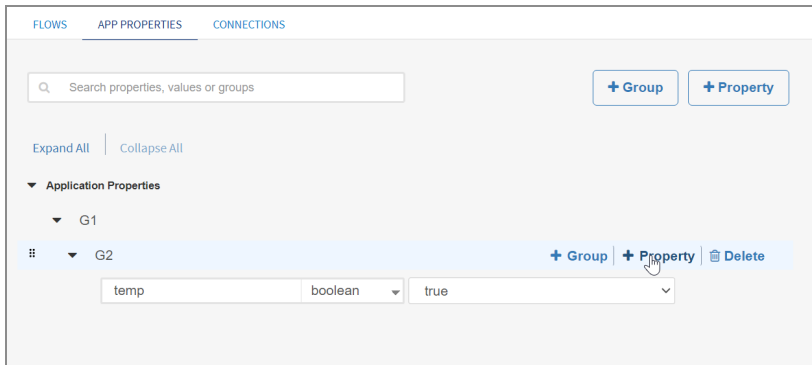
Procedure

1. Open the app details page and click **App Properties**.
2. Click **+Group** on the upper-right corner to add the group.

- Click the newly created group name to make it editable and enter a meaningful name for the group.

The group gets created. The group is simply a label and cannot be used by itself. So, you must add a group or a property within the group.

- To add a property within the group, hover your mouse cursor to the extreme right of the group until the **+Property** button appears in the group row.



- Click the **+Property** button to add the property and rename it.
- Select a data type for the property and enter a value. Entering a value and selecting a data type is mandatory.
- Save your changes for the newly created property after populating them with the correct value.

The property gets created under the parent.

Note: You can even add a nested group under the parent group by clicking **+Group** in the group row.

- Deploy (or redeploy) the app to Data Plane for the group to be available from the **Environment Controls** tab in the UI.

Deleting a Group or Property

An existing group or a property can be deleted in the following ways.

To delete a property:

- Open the **App Properties** tab from the app details page.

2. Hover your mouse cursor to the extreme right end of the property and click **Delete**.

To delete a group or a nested group:

1. Open the **App Properties** tab from the app details page.
2. Hover your mouse cursor to the extreme right end of the group and click **Delete**. A confirmation window appears.

Here,

- **Delete all child properties and groups** deletes all the standalone properties and nested groups and properties under the group.
- **Fold all children into Group_2 > Group_1** deletes the nested group but the properties under the nested group will be shifted into the parent group.
- **Move all children to top level** deletes the parent or a nested group and shift all the properties to the top level as a standalone property.

3. Select the desired delete option on the confirmation window and click **Confirm**.



Caution: The property path mappings may update on editing the property or on moving a property from a nested group to a parent group or if the property is shifted out of the group to the top level as a standalone property.


Using App Properties in a Flow

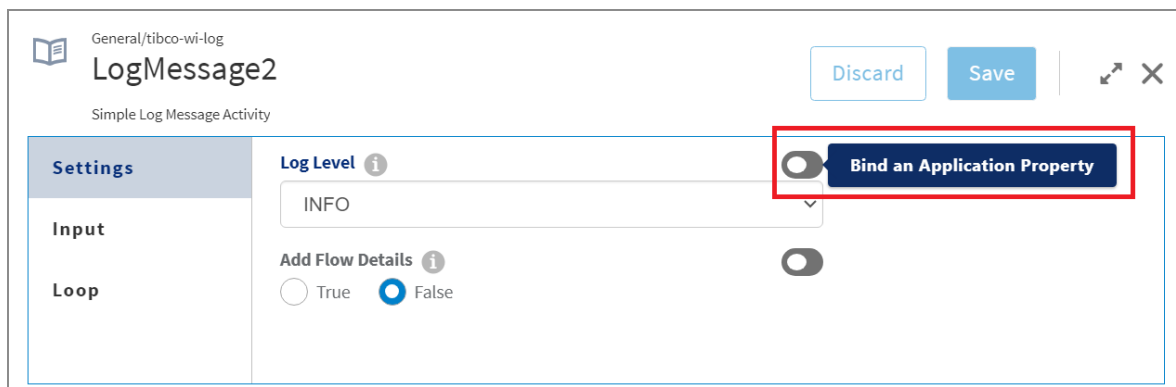
Configuring a field with an app property is recommended for fields that require their values to be overridden when the app goes into production. Hence, the decision as to which fields in an Activity should support app properties (which fields can be configured using an app property) must be decided at the time when the extension for the category is being developed. The fields that can be configured using an app property display a slider button against their names in the UI.

Connection-specific app properties are visible in the **App Properties** dialog after you select a connection when configuring the Activity or trigger, but they appear in read-only mode. This is because connections are reusable across apps and connection-related app properties are managed (refreshed) automatically. Connection-related app properties cannot be used for configuration anywhere within an app. Their only purpose is to allow you to change a connection value if required during runtime. For more details on how the connection properties get created and used, see [Using App Properties in Connections](#).

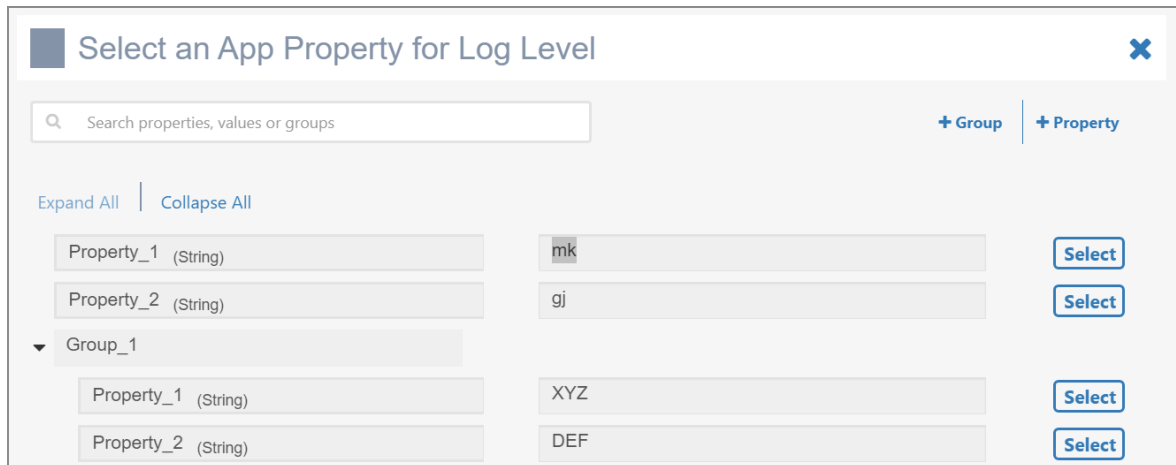
To configure a field with an app property:

Procedure

1. Open the flow details page.
2. Click the Activity whose field you want to configure with an app property.
This opens the configuration pane for the Activity.
3. Click the slider () against the name of the field that you want to configure with an app property. If the field does not display a slider, the field cannot be configured with an app property.



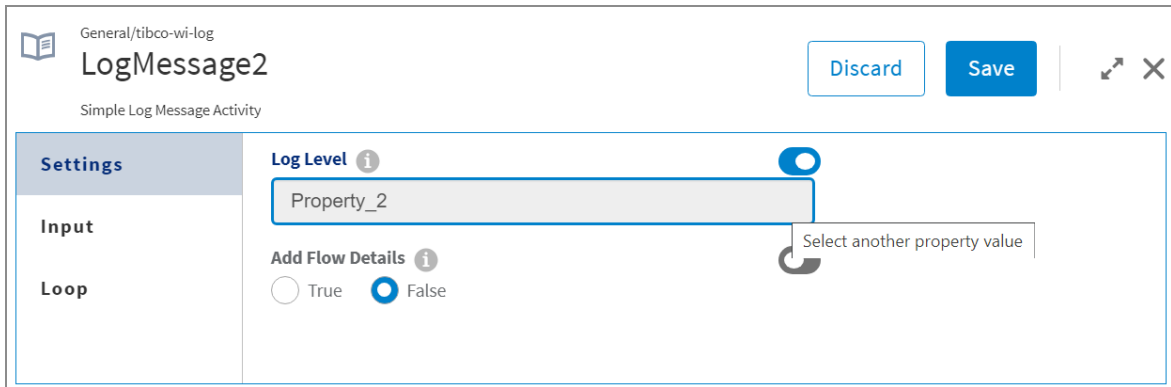
4. The **App Properties** dialog opens. Only those app properties whose data type matches the data type of the field are displayed. You can also create a new group or a property in this view. Here, you can add a single property or a group at a time.



5. Select the property that you want to configure for the field.
The property name appears in the text box for the field and the default value of the

property gets implicitly assigned to the field.

After configuring the property, if you want to change a field to use a different property, hover your mouse cursor over the end of the text box for the field until the **Select another property value** icon appears. Click the **Select another property value** icon.




For a field that has been configured with an app property, you can unlink the property from the field. For more information, see [Unlinking an App Property from a Field](#).

Using App Properties in the Mapper

You can use app properties when mapping an input field. The app properties available for mapping are grouped under the **\$property** domain-specific scope in the mapper. All mapper rules and conditions apply to the use of app properties as well. For example, the data type of the app property value must match with the input field data type when mapping. Connection-related app properties that are used by any connection field in an Activity do not appear under **\$property** since they cannot be accessed. Connection-related app properties cannot be used for configuration anywhere within an app. Their only purpose is to allow you to change a connection value if required during runtime. Hence, they cannot be used to map input fields.

For more information on how to use the mapper, see [Data Mappings](#).

Unlinking an App Property from a Field Value

For a field that has been configured with an app property, if you decide later not to use the app property, you can click and slide its slider ball () to the left. This removes the app property from the field (unlink it from the field) but leaves the field configured with the

default value of the app property. The field retains the default value of the app property, but it gets disassociated from the app property and appears as a manually entered value. Hence, if you change the default value of the app property beyond this point, it does not affect the value of the field.

Using App Properties in Connections

Connection-related app properties can be used to modify or configure app properties anywhere within an app. If needed, the connection-related app properties also allow you to change the connection values during runtime. Before you deploy your app, their values can only be edited in the connection details dialog, the dialog where you provided the credentials for the connection. You can open this dialog by editing the connection from the **Connections** page in the UI. Connection-related properties are useful when you want to change the value for one of the connection fields, for example, a URL, when an app goes from the testing stage to production.

How the connection-related app properties get created

You cannot explicitly create connection-related properties. When you select a connection in the **Connection** field of an Activity, the supported properties associated with that connection automatically get created and populated in the **App Properties** dialog.

One property gets created for both **Connection URL** and **Authentication Key** fields in the connection details dialog. The values you enter for the above two fields in the connection details dialog become the default values for the connection properties. The properties take their name from the connection field that they represent in the connection details dialog.

You begin by creating a connection. In the example below, only the **Connection URL** and **Authentication Key** fields support app properties.

TIBCO Cloud Messaging eFTL Connection

Connection Name ⓘ
MyTCMConnection

Description ⓘ

Connection URL ⓘ
http://google.com

Authentication Key ⓘ
.....

Once the connection is created, you can use it to configure the **Connection** field in an Activity. In the example below, the connection created above is being used to configure the **Connection** field of the **TIBCO Cloud Messaging eFTL Connection** Activity.

Messaging/tibco-messaging-tcm-pub
eFTLMessagePublisher Discard Save

This activity sends a message to TIBCO Cloud Messaging(eFTL) service

Settings

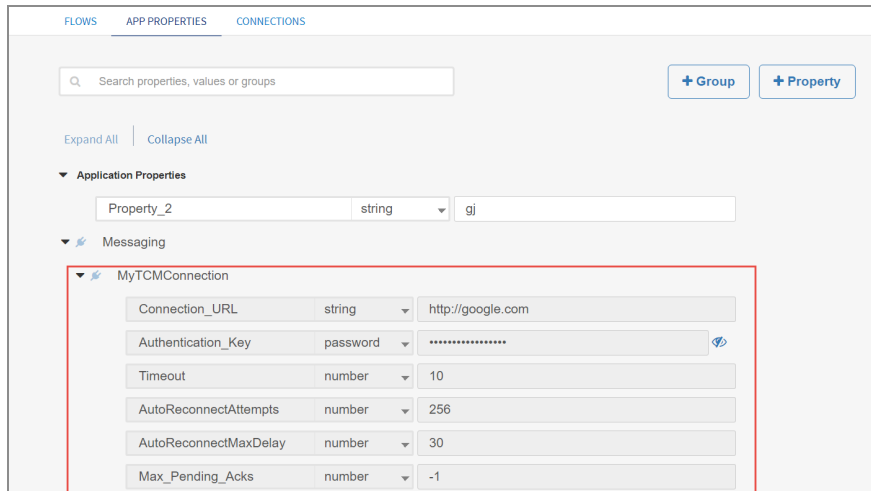
- Input Settings
- Input
- Loop

Connection ⓘ

MyTCMConnection ▼

MyTCMConnection

After configuring the Connection field with the connection, if you open the **App Properties** tab, the connection properties for the field (enclosed in the red box in the image below) is displayed. Notice that only the supported properties (Connection URL and Authentication Key) are displayed in a read-only mode.



The properties that are displayed in the **App Properties** tab change dynamically based on your selection of the connection to use. You can only view the connection properties. You cannot edit or delete them from the **App Properties** tab. Deleting the Activity that uses the connection automatically removes the associated connection properties that the Activity used from the **App Properties** tab.

Using connection-related app properties

Connection-related app properties are available for use from the mapper. You can use these properties to change a connection value (for example, a URL or password) just before an app goes from a testing stage to production. All the mapped configurations can be pre-checked using a flow tester or by creating a pre-check flow. Their values cannot be changed from the **App Properties** tab. Change their values in the connection details dialog before pushing the app. You can change their values after an app has been deployed from the Environment Controls tab. For more information, see [Changing the Default Value of a Property in the Environment Controls Tab](#).

Editing an App Property

You can change the default value or data type of an app property at any time.

After the app has been deployed to Data Plane, the app properties display in the Environment Controls tab of the UI. You can change their values in this tab, but you need to redeploy the app for your edits to take effect. For more information, see [Changing the Default Value of a Property in the Environment Controls Tab](#).

Changing the Default Value of a Property from the App Properties Tab

You can change the default value of an existing app property at any time after creating the property. You must redeploy the app for the updated value to take effect and be visible in the **Environment controls** tab. Before you deploy the app to the cloud, you can change the default value in the **App Properties** tab.

To change the default value of an existing app property:

Procedure

1. Open the **App Properties** tab by clicking **Properties** on the app details page.
2. Click inside the text box for the property value that you want to change.
3. Edit the value.
4. Save your changes for the edited property.
5. Deploy (or redeploy) the app for your changes to take effect.

Changing the Default Value of a Property in the Environment Controls Tab

The app properties that you created in the **App Properties** tab become available in the **Environment Controls** tab in the UI after you deploy (or redeploy) your app to Data Plane. You can modify the values of the properties in the **Environment Controls** tab. The value for the data type password remains encrypted in the **Environment Controls** tab. You can modify the password values, but you cannot view the default password that was set in the **App Properties** tab for the property.

To change the value of a property in the **Environment Controls** tab:

Before you begin

You must have deployed (or redeployed) the app to Data Plane after adding or modifying a property in the **App Properties** tab.

Procedure

1. Click the **Value** column for the property to put it in edit mode.

2. Edit the property value and click **Save**.
3. Click **Push Updates** for your changes to take effect.

Changing the Name or Data Type of an App Property after Using It

If you change either the name of an app property or its data type after you have used the property to configure a field in an Activity or trigger, the field displays an error message. You must explicitly reconfigure the field to use the modified property by deleting the property from the text box for the field and adding the modified property.

Exporting App Properties to a File

You can export the app properties to a .flogo file or a .properties file. The exported FLOGO file can be used to override app property values. The .properties file can be used to create a ConfigMap in Kubernetes. When using the exported properties file, the values in the properties file get validated by the app during runtime. If a property value in the file is invalid, you get an error saying so and the app proceeds to use the default value for that property instead.

Overriding an App Property Locally

When running an app, you can override an app property by performing the following steps:

Procedure

1. Select the app that you want to run in **Explorer View**.
2. Navigate to **Explorer View > Flogo App Workspace** and click the dropdown icon.
3. Click the **Run** icon and select the **Configure and Run** icon.
4. Set the following environment variable FLOGO_APP_PROPS_ENV=auto and add the app properties as comma-separated key-value pairs.

App Schemas

You can define a JSON schema that is available for reuse across an app. Creating an app-level schema saves you the time and effort of entering the same schema multiple times. An app-level schema can be used in any flow, Activity, or trigger configuration where a schema editor is provided. You can simply pick an existing schema from a list. For example, app-level schemas are available from the following locations:

- Inputs or Outputs tab of a flow (including Error Handler flows and subflows)
- Input or Output Settings tab of an Activity
- Output or Reply Settings tab of a trigger

App-level schemas are filtered based on the type of Activity or trigger. For example, only JSON schemas are displayed in a REST trigger or Activity configuration.

Currently, Flogo Extension for Visual Studio Code only supports JSON type of schema.


Defining an App-Level Schema

Procedure


1. On the App Details page, click the **SCHEMAS** tab.

The **SCHEMAS** tab opens.

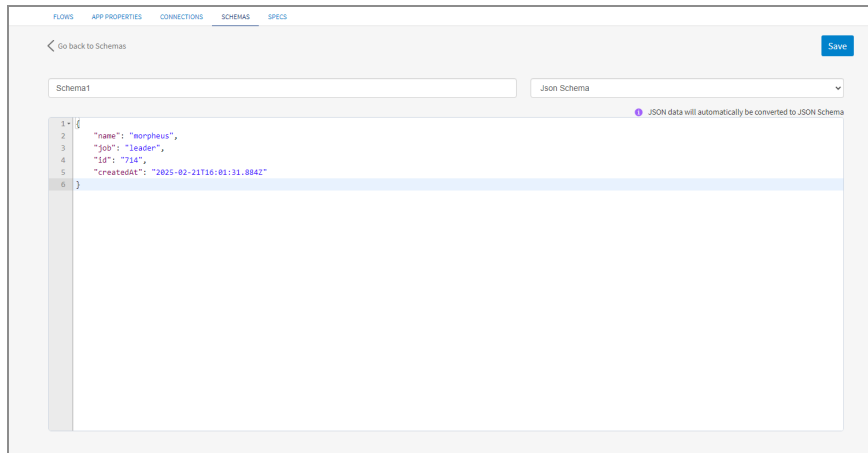
2. Click **Create Schema**.

 **Note:** If your app has a schema already, click **+Create** to create a schema.

3. In **Schema Name**, enter a schema name.
4. **JSON Schema** is the only available schema type.
5. Enter the schema in the schema editor.

 **Note:** If you enter JSON data in the editor, it is automatically converted to JSON schema.

6. To save the schema, click **Save**.



Result

After the schema is defined, it can be used in any Activity or trigger configuration by using the **Use an app-level schema** button in the schema editor of the Activity or trigger.

Editing an App-Level Schema

When you make changes to an app-level schema, the changes are automatically reflected everywhere the schema is used.

To edit an app-level schema:

Procedure

1. On the App Details page, click the **SCHEMAS** tab.
2. Click on the schema to be edited.

The schema page opens up.

3. Edit the schema name or the schema in the editor, as required.

If the app-level schema is used in any flow, Activity, or trigger, a warning is displayed.

Deleting an App-Level Schema



Warning: Deleting a schema removes its reference from all the places where it is used, but it retains a copy of the schema in the fields that use the schema.

Procedure

1. On the App Details page, click the **SCHEMAS** tab.
2. Click the **Delete** icon beside the schema to be deleted.

Result

After confirmation, the selected schema is deleted.


Using an App-Level Schema

You can use an app-level schema from a flow, trigger, or Activity from the following tabs:

- Inputs or Outputs tab of a flow
- Input or Output Settings tab of an Activity
- Output or Reply Settings tab of a trigger

Flow Input & Output Tab

Use these tabs to configure the input to the flow and the flow output. These tabs are particularly useful when you create blank flows that are not attached to any triggers.

 **Note:** The schemas for input and output to a flow can be entered or modified only on this **Flow Inputs & Outputs** tab. You cannot coerce the flow input or output from outside this accordion tab.

Both these tabs (the **Input** tab and the **Output** tab) have two views:

- **JSON schema view:**

You can enter either the JSON data or the JSON schema in this view. Click **Save** to save your changes or **Discard** to revert the changes. If you entered JSON data, the data is converted to a JSON schema automatically when you click **Save**.

- **List view:**

This view allows you to view the data that you entered in the JSON schema view in a list format. In this view, you can:

- Enter your data directly by adding parameters one at a time

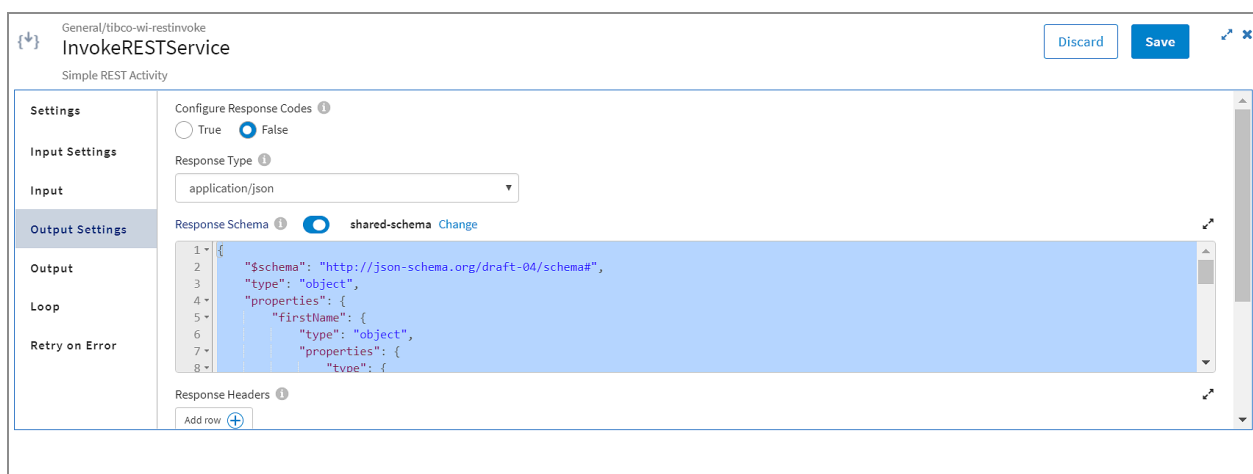
- Mark parameters as required by selecting its checkbox.
- When creating a parameter, if you select its data type like an array or an object, an ellipsis (...) appears to the right of the data type. Click the ellipsis to provide a schema for the object or array.
- Use an app-level schema by selecting the **Use an app-level schema** button. On the **Schemas** page that appears, click **Select** beside the schema that you want to use. The name of the schema is displayed beside the **Use an app-level schema** button and the schema is displayed in a read-only mode.

Note: You cannot edit an app-level schema in the **List** view if the **Use an app-level schema** button is selected. To edit an app-level schema, follow the instructions in the section [Editing an App-level Schema](#). You can, however, switch to another app-level schema by clicking **Change** and selecting another app-level schema. You can also unbind the app-level schema (by deselecting the **Use an app-level schema** button) from a trigger, activity, or the input and output of a flow. After you unbind the app-level schema, you can make changes to it using the schema editor in the **List** view.

- Click **Save** to save the changes or **Discard** to discard your changes.

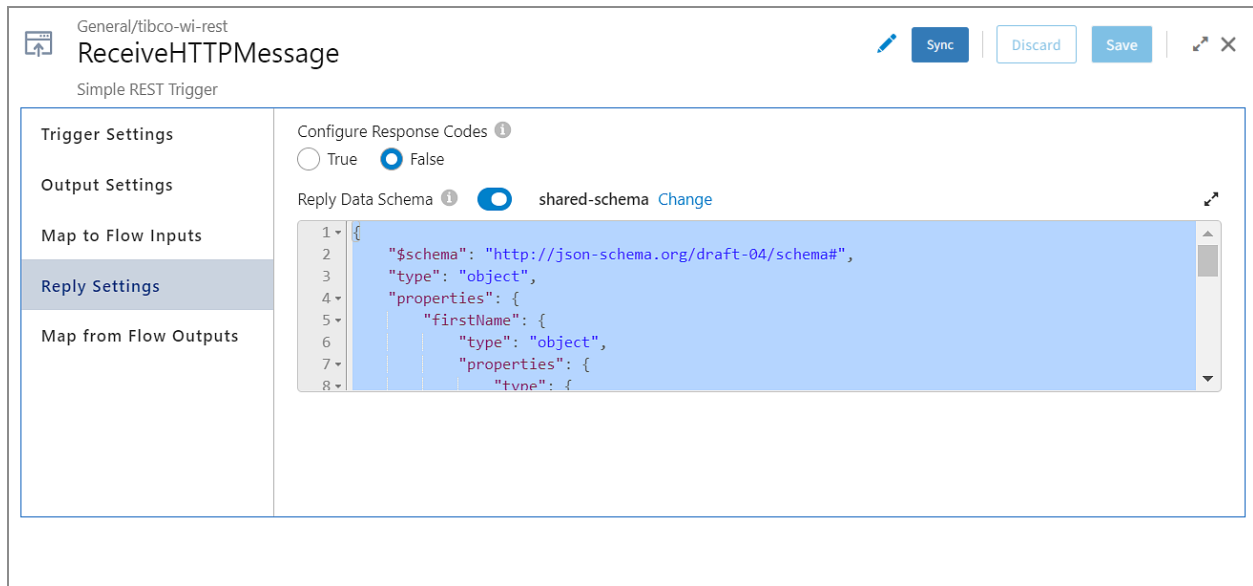
Input or Output Settings Tab of an Activity

When configuring an Activity, you can select an app-level schema on its **Input** or **Output Settings** tab. For example, the following screenshot shows an app-level schema selected in the **Response Schema** field of the **Output Settings** tab of an InvokeRESTService Activity.



Output or Reply Settings Tab of a Trigger

When configuring a trigger, you can select an app-level schema on its **Output** or **Reply Settings** Tab. For example, the following screenshot shows an app-level schema selected in the **Reply Data Schema** field of the **Reply Settings** tab of a ReceiveHTTPMessage trigger.



Note: If there is a change in the schema attached to a trigger, click **Sync** to synchronize it with the input and/or output of the flow.

App Specs

You can import a spec and that is available for reuse across an app. Importing an app-level spec saves you the time and effort of uploading the same spec in multiple places. An app-level spec can be used in **InvokeRESTService**, **gRPC Invoke**, **gRPC trigger**, **REST Trigger - ReceiveHTTPMessage**, and **GraphQL Trigger**.

You can simply select an imported spec from a list by clicking the **Use app level specs** toggle.

Currently, three types of specs are supported:

- **OpenAPI:** OpenAPI spec file or Swagger spec file, which is a type of .json file

- **gRPC**: A .proto file
- **GraphQL**: A .gql or .graphql file


App-level specs are filtered based on the type of Activity or Trigger. Only OpenAPI specs are displayed on the **Browse API Specs** dialog in a **REST Trigger - ReceiveHTTPMessage** or **InvokeRESTService** activity configuration when using app level specs. Only **GraphQL** specs show on **GraphQL** trigger and **gRPC** specs type for **gRPC** trigger and **gRPC Invoke** activity.

Importing App Specs


You can import specs locally or from a runtime.


From TIBCO Cloud™ Integration, all the uploaded API specs in the API modeler populate the list. You can select and import any API in the app. From TIBCO Platform™, API specs uploaded on TIBCO Developer Hub show in the list from which you can select and import a spec in the app. All uploaded specs have individual tiles that give **Type** and **Source Details**.

Importing Specs from Runtime


 **Note:** You must first create a runtime before proceeding with this import. For more information, see [Configuring the Runtime](#).

1. Click **Import Specs**.
2. Choose the runtime from where you want to import a spec.
3. You can select TCI or Platform Runtime.

 **Note:** For importing specs from the TCI runtime, the TCI OAuth token should be given access to the API modeler.

 **Note:** For importing specs from the TIBCO Platform, TIBCO Developer Hub must be provisioned in the Data Plane.

4. After selecting a runtime, all the OpenAPI specs show in the list. Select the spec that you want to import and click the **Import** button.

 **Note:** Currently, only OpenAPI spec can be imported from a runtime.


Importing Specs Locally

To import specs from a local disk, perform the following steps:

1. Click **Import Specs**.
2. Choose **Import spec from disk**.
3. From the **Select Spec Type** dropdown list, choose the type of spec.
4. Now, you can select and upload a file of the type you have previously selected.

The uploaded spec shows up on a tile on the **SPECS** tab.

Implementing App Specs

 **Note:** Implementing a spec is supported for OpenAPI specs only.

You can create the Flogo app logic (flows) by importing and implementing an API specification file. For importing an API specification, see [Importing App Specs](#). To implement a spec, perform the following steps:

1. Hover over the tile of the spec.
2. Click the vertical ellipsis.
3. Select Implement.

When you implement a spec, the **ConfigureHTTPResponse** and **Return** activities are automatically added to the flow. The mappings from trigger output to flow inputs get configured for you based on the definitions in the specification. The output of the **ConfigureHTTPResponse** activity is automatically mapped to the **Return** activity input. However, you must configure the input to the **ConfigureHTTPResponse** activity manually.

Using App Specs

When configuring a trigger supporting specs, you can select an app-level specs. This section describes how to use specs for different triggers and activities.

REST Trigger - ReceiveHTTPMessage

1. Set the **Configure Using API Specs** option to True.
2. Click the **Use app level spec** toggle.
3. Select the required OpenAPI spec from the **Browse App Specs** dialog and click **Done**.
4. Select the **Path** and **Method** inputs, which you want to implement, from the dropdown menus.

InvokeRESTService Activity

1. After adding the activity, click **Use app level spec** next to the API spec picker.
2. Select the required OpenAPI spec from the **Browse App Specs** dialog and click **Done**.
3. Select the **Path** and **Method** inputs, which you want to use, from the dropdown menus.

GraphQL Trigger

1. When adding a trigger, click **Use app level spec** toggle next to the GraphQL schema File picker.
2. Select the required GraphQL spec from the **Browse App Specs** dialog and click **Done**.
3. Select the **GraphQL Operation** and **Resolver** inputs, which you want to implement, from the dropdown menus.

gRPC trigger

1. When adding a trigger, click the **Use app level spec** toggle next to the Proto File picker.
2. Select the required gRPC spec from the **Browse App Specs** dialog and click **Done**.
3. Select the **Service Name** and **Method Name** inputs, which you want to implement, from the dropdown menus.

gRPC Invoke

1. Click the **Use app level spec** toggle next to the Proto File picker.
2. Select the required gRPC spec from the **Browse App Specs** dialog and click **Done**.
3. Select the **Service Name** and **Method Name** inputs, which you want to use, from the dropdown menus.

Updating App Specs

i Note: When you change an app-level spec, the changes are automatically reflected everywhere the spec is used, such as all activities and triggers.

To update a spec, you can perform one of the following procedures:

Update the spec directly

1. Click the spec to open it in an inline editor.
2. Make the required changes to the editable spec.
3. Save it.

Upload an updated spec

1. Click the vertical ellipsis on the spec tile.
2. Select **Update**.

i Note: You can update a spec by importing an updated one from local disk or from a runtime.

1. After uploading an updated spec, click the **Update** button. Approve the confirmation dialog.

i Note: Spec can be updated with the same type of spec. For example, you can update an OpenAPI spec with another OpenAPI spec.

Deleting App Specs

To delete an app spec, perform the following steps:

1. Go to the **SPECS** tab.
2. Hover over the tile of the spec that you want to delete.
3. Click the delete icon on the app tile.
4. Approve the deletion confirmation dialog to delete the Spec from the app.



Warning: Deleting a spec does not remove it from the triggers and activities in which it is used. You must update those manually.

Using Connectors

To use the Flogo connectors:

1. [Create one or more connections.](#)
2. If you do not already have an app, [create an app](#).
3. Create a flow.
4. Add the activities to the connector that you use as needed.
5. Build and test the app locally using the appropriate runtime. For more information, see [Running Apps Locally](#)
6. Deploy the app to the desired runtime environment.

For more details on connectors, see [Supported Flogo Connectors](#).

Supported Connectors

Flogo® Extension for Visual Studio Code (VS Code) supports several Flogo connectors. For more information, see [Supported Flogo Connectors](#).

Prerequisites for Connectors

This section covers the prerequisites for connectors. It includes the following topics:

- [OAuth Connections](#)
- [Installing Driver from Visual Studio Code UI](#)
- [Installing ODBC Driver Manager](#)
- [Installing Oracle Database ODBC Drivers](#)
- [Installing TIBCO Data Virtualization ODBC Driver](#)

- [Installing Oracle MySQL ODBC Driver](#)
- [Installing Microsoft SQL Server ODBC Driver](#)
- [Installing PostgreSQL and Greenplum ODBC Drivers](#)
- [Installing Amazon Redshift Database Drivers](#)
- [Installing Snowflake ODBC Drivers](#)
- [Setting up EMS for Local Runtime](#)

OAuth Connections

For the Flogo Connectors that support OAuth 2.0 connection type, set the Callback or redirect URL to <https://vscode.dev/redirect>.


Installing Driver from Visual Studio Code UI

Note:

- On Windows, you need Admin privileges to run **Install Prerequisites for Flogo Connectors**. Uninstall the existing Chocolatey package manager.
- On Linux and macOS, you need Admin privileges to install the packages that enable **Install Prerequisites for Flogo Connectors**.

To install the prerequisites for connectors, perform the following steps:

1. Go to the Flogo icon on the left side of the menu.
2. Click **HELP AND FEEDBACK** and select **Install Prerequisites for Flogo Connectors**.
3. Now, you can select the available connectors from the list for driver installation.

 **Note:** In prerequisites, you can also install Go version 1.24.4 and the Flogo Golang dependencies.

4. Install the driver for the selected connector.

i Note: Installing Driver from Visual Studio Code UI Script includes installing ODBC Driver Manager and the selected Connector Driver.

i Note: On Windows, the prerequisite installation scripts that use the MSI installer might not display the correct success message when performing driver removal, modification, or repair actions.

Installing ODBC Driver Manager

Before creating a database server connection, you must install ODBC Driver Manager and Database drivers. Install ODBC Driver Manager by running one of the following platform-specific commands.

Platform	Command
Windows	The ODBC Driver Manager is prepackaged with Windows as ODBC Data Source Administrator.
macOS	<p>To install the ODBC Driver Manager, run the following command:</p> <pre>brew install unixodbc</pre> <p>The above command creates the configuration file <code>odbcinst.ini</code> at the default location: <code>/usr/local/etc/odbcinst.ini</code></p> <p>If <code>brew</code> is not installed on your system, you can install it by running the following command:</p> <pre>/bin/bash -c "\$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"</pre>

Platform	Command
	<p>Note: If the <code>odbcinst.ini</code> configuration file is not created automatically, then you must create it by running the following command:</p> <pre>touch \$(brew --prefix)/etc/odbcinst.ini</pre>
	<p>Note: TIBCO Flogo® Connector for Snowflake requires an iODBC driver manager. You can download the iODBC from here. After downloading the iODBC driver manager, double-click the downloaded .dmg file. Double-click the installer file <code>iODBC-SDK.pkg</code> and follow the instructions.</p>
Ubuntu	<p>To install the unixODBC Driver Manager, run the following command:</p> <pre>apt install unixodbc</pre> <p>For TIBCO Data Virtualization Flogo Connector, run the following command:</p> <pre>apt-get install unixodbc unixodbc-dev</pre> <p>The above command creates the <code>odbcinst.ini</code> configuration file at the default location: <code>/etc/odbcinst.in</code></p> <p>Note: If the <code>odbcinst.ini</code> configuration file is not created automatically, then you must create it by running the following command:</p> <pre>touch /etc/odbcinst.ini</pre>

Installing Oracle Database ODBC Drivers

i Note: Due to a dependency on the OS-specific Oracle client library, app executable for apps using Oracle connector cannot be built for Windows and macOS with TIBCO Cloud Integration runtime and TIBCO Platform™ runtime. With local runtime, building a cross-platform executable, such as a Linux executable on Windows, is not supported when using Oracle connector.

To install Oracle ODBC drivers on specific platforms:

On Windows

1. Download and unzip the Instant Client Basic package from the Oracle website.
2. Download the Instant Client ODBC package. Unzip it in the same directory as your basic package.
3. Execute `odbc_install.exe` from the Instant Client directory as administrator.

Result

After installation, the Oracle ODBC Driver in the ODBC Data Source Administrator (x64) program is now visible on the **Drivers** tab.

For more information, see:

<https://www.oracle.com/in/database/technologies/releasenote-odbc-ic.html>

Windows Prerequisite script for Oracle Driver installation fails for chocolatey installation. Perform the following steps for driver installation:

1. Copy the following command: `powershell.exe -ExecutionPolicy Bypass -File ".\vscode-preinstall.ps1" oracle -Verb RunAs`
2. Open a new PowerShell window as administrator.
3. Go to the script path in PowerShell or update the command with the appropriate path to the script.
4. Paste and run the command in the new PowerShell window.

On macOS

1. To install Oracle ODBC driver, run the following commands,

```
brew tap InstantClientTap/instantclient  
brew install instantclient-basic  
brew install instantclient-odbc
```

The driver is installed at the default location.

For example: `/usr/local/Cellar/instantclient-odbc/<driver-version>dbcu/lib/libsqora.dylib.<version>`

2. Edit the `odbcinst.ini` file with the following text with driver file path and name as per your system.

```
[Oracle ODBC Driver  
Description=Oracle 19 ODBC driver  
Driver=<default driver location>
```

On Ubuntu

Oracle ODBC driver requires the `libaio` package to be installed as a prerequisite, run the following command to install the package:

```
apt install libaio1
```

1. Download and unzip the Instant Client Basic package from the Oracle website.
2. Download the Instant Client ODBC package. Unzip it in the same directory as your basic package.
3. Execute `odbc_update_ini.sh` from the Instant Client directory.

Result

After installation, the Oracle ODBC Driver in the `odbcinst.ini` file is now visible with the driver path.

To know more about installation steps, see:

<https://www.oracle.com/in/database/technologies/releasenote-odbc-ic.html>

Installing TIBCO Data Virtualization ODBC Driver

i Note: Due to dependency on the OS-specific TIBCO Data Virtualization client library, app executable for apps using TDV connector cannot be built for Windows and macOS with TIBCO Cloud Integration runtime and TIBCO Platform runtime. With local runtime, building a cross-platform executable, such as building a Linux executable on Windows, is not supported when using the TIBCO Data Virtualization connector.

On Windows:

Procedure

1. From [TIBCO eDelivery](#), download the TIBCO Data Virtualization ODBC Driver package.
2. Extract the contents of the package to a local directory.
3. Run the installer file.
4. Confirm that the installation of the driver is successful.
 - a. Click **Start > All apps > Windows Tools > ODBC Data Sources (64-bit)**.
 - b. Go to the **Drivers** tab, and verify an entry for TIBCO Data Virtualization ODBC Driver.

On Ubuntu:

Procedure

1. From [TIBCO eDelivery](#), download the TIBCO Data Virtualization ODBC Driver package.
2. Extract the contents of the package to a local directory.
3. Set the environment variables ODBCINI and ODBCINSTINI to locations of odbc.ini and odbcinst.ini files, respectively.
4. From the local directory, run the `/bin/driverConfig` utility. When the utility prompts for the driver path, provide the driver path from the `lib/libcomposite.so` file.
5. Confirm that the installation of the driver is successful. In the `odbcinst.ini` file, verify an

entry for the TIBCO Data Virtualization ODBC Driver with the driver path.

Installing Oracle MySQL ODBC Driver

To install the database driver:

On Windows

1. Download the MSI installer from the [Oracle MySQL](#) website.
2. Run the executable installer.

Oracle MySQL ODBC Driver in the ODBC Data Source Administrator (x64) program is now visible on the **Drivers** tab.

On Ubuntu

1. Download the Debian installer file from the [Oracle MySQL](#) website.
2. Run the executable installer.

Oracle MySQL ODBC Driver is now visible in the `odbcinst.ini` file with the driver path.

For prior versions of Ubuntu 22.04:

1. Download and extract the driver compressed `.tar` file from the MySQL website to the required location.
2. Use the installer utility from the `bin` folder of the extracted driver folder, run the following command:

```
./myodbc-installer -d -a -n "MySQL ODBC Unicode Driver" -t  
"DRIVER=/path/to/driver/libmyodbc8w.so;"
```

Installing Microsoft SQL Server ODBC Driver

To install the database driver:

On Windows

1. Download the Microsoft SQL Server ODBC Driver MSI installer file from the [Microsoft](#)

website.

2. Run the executable installer.

Microsoft SQL Server ODBC Driver in the ODBC Data Source Administrator (x64) program is now visible on the **Drivers** tab.

On macOS and Ubuntu

To install the Microsoft ODBC Driver for SQL Server:

- [Microsoft ODBC driver for SQL Server - macOS](#)
- [Microsoft ODBC driver for SQL Server - Ubuntu](#)

Microsoft SQL Server ODBC Driver is now visible in the odbcinst.ini file with the driver path.

Installing PostgreSQL and Greenplum ODBC Drivers

To install the PostgreSQL and Greenplum ODBC drivers on specific platforms:

On Windows

1. Download the PostgreSQL ODBC Driver MSI Installer file from <https://odbc.postgresql.org/>.
2. Run the executable installer.

Result

After installation, the PostgreSQL ODBC Driver in the ODBC Data Source Administrator (x64) program is now visible on the **Drivers** tab.

On macOS

1. Run the following command:

```
brew install psqlodbc
```

The driver is installed at the default location.

For example: `/usr/local/Cellar/psqlodbc/16.00.0000/lib/psqlodbcw.so` or
`/opt/homebrew/Cellar/psqlodbc/16.00.0000/lib/psqlodbcw.so`

2. Edit the `odbcinst.ini` file with the following text with driver file path and name as per your system.

```
[PostgreSQL]
Description=PostgreSQL ODBC Driver
Driver=<default driver location>
```

On Ubuntu

- Run the following command:

```
apt install odbc-postgresql
```

Result

After installation, the PostgreSQL ODBC driver in the `odbcinst.ini` file is now visible with the driver path.

Installing Amazon Redshift Database Drivers

To install the Amazon Redshift database drivers on specific platforms:

On Windows

1. Download the Amazon Redshift ODBC Driver MSI installer file from the Amazon Redshift website.
2. Run the executable installer.

Result

After installation, the Amazon Redshift ODBC driver in the ODBC Data Source Administrator (x64) program is now visible on the **Drivers** tab.

On macOS

1. Download the Redshift ODBC Driver installer from the Amazon Redshift website.
2. Extract and install the package.

The driver is installed at the default location.

For example: `/opt/amazon/redshift/lib/libamazonredshiftodbc.dylib`

3. Edit the `odbcinst.ini` file with the following text with driver file path and name as per your system.

```
[Amazon Redshift]
Description=Amazon Redshift ODBC Driver
Driver=<default driver location>
```

On Ubuntu

1. Download the driver file provided by Amazon.
2. Run the executable installer.

Result

After installation, the Amazon Redshift ODBC driver in `odbcinst.ini` file is now visible with the driver path.

For more information, see <https://docs.aws.amazon.com/redshift/latest/mgmt/odbc20-install-config-linux.html>

Installing Snowflake ODBC Drivers

To install the Snowflake ODBC drivers on specific platforms:

On Windows

1. Download and install the MSI Installer file from the Snowflake ODBC installation page.
2. Configure the ODBC Driver. Ensure that you see an entry for SnowflakeDSIIDriver in the ODBC Data Source Administrator(x64) program under the **Drivers** tab.
 - a. Launch the Windows Data Source Administration tool.
 - b. Verify if the Snowflake ODBC driver is installed.
 - c. Create a new data source name (DSN).

Result

After installation, the SnowflakeDSIIDriver in the ODBC Data Source Administrator(x64) program is now visible on the **Drivers** tab. For more information, see [Snowflake Developer Guide](#).

On macOS

1. Download the Snowflake ODBC driver installer from the Snowflake website.
2. Mount and install the package as mentioned [here](#).

If you choose the default directory when prompted, the installer installs the ODBC driver files in the `/opt/snowflake/snowflakeodbc/Library/ODBC` directory.

To create a DSN, edit the appropriate `odbc.ini` file.

Result

You can use the `iodbctest` command-line utility provided with iODBC to test the DSNs you create.

On Ubuntu

- Download and extract the TGZ file. Install and set up the ODBC driver as described in the [Snowflake](#) documentation.
- Configure the environment with unixODBC.

In a terminal window, change to the `snowflake_odbc` directory, and run the `./unixodbc_setup.sh` command to install the Snowflake ODBC.

This script completes the following:

- Adds a Snowflake connection to your system-level `/etc/odbc.ini` file.
- Adds the Snowflake driver information to your system-level `/etc/odbcinst.ini` file.
- Adds the certificate authority (CA) certificates required by the Snowflake ODBC driver to your system-level `simba.snowflake.ini` file.
- By running the `unixodbc_setup.sh` command, you do not need to set any environment variables.

Result

Test the DSNs you created using the `isql` command-line utility provided with unixODBC. On the command line, specify the DSN name, user login name, and password.

Setting up EMS for Local Runtime

i Note: Due to dependency on the OS-specific EMS client library, app executable for apps using EMS connector cannot be built for Windows and macOS with TIBCO Cloud™ Integration runtime and TIBCO Platform runtime. With local runtime, building a cross-platform executable, such as a Linux executable on Windows, is not supported when using the EMS connector.

i Note: TIBCO Enterprise Message Service™ (EMS) Client 10.3.0 or above is required.

- For Windows
 1. Download the TIBCO Enterprise Message Service™ (EMS) Client package from [TIBCO eDelivery](#) and extract to `TIB_ems_<version>_win_x86_64/TIB_ems_<version>`.
 2. Set the `EMS_HOME` environment variable.

3. Set the EMS_HOME=<EMS installation path>.
4. For running the binary, add the following to the system path:
 <EMS installation path>\<version>\bin
5. Install PowerShell on Windows and then run Get-ExecutionPolicy.
6. If it returns the status as "restricted", then run Set-ExecutionPolicy AllSigned.
7. If it is not restricted, then directly run the following command in PowerShell:

```
Set-ExecutionPolicy Bypass -Scope Process -Force;
[System.Net.ServicePointManager]::SecurityProtocol =
[System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-Object
System.Net.WebClient).DownloadString
('https://community.chocolatey.org/install.ps1'))
```

8. Run the following command:

```
choco install mingw -y
```

9. Run the following command:

```
choco install pkgconfiglite
```

- For macOS

1. Download the TIBCO Enterprise Message Service™ (EMS) Client package from [TIBCO eDelivery](#) and extract to TIB_ems_<version>_macosx_x86_64/TIB_ems_<version>.
2. Set the EMS_HOME environment variable.
 export EMS_HOME=<EMS installation path>
3. Install Xcode IDE.
 Xcode-select -install
4. To run the binary, export DYLD_LIBRARY_PATH.
 export DYLD_LIBRARY_PATH=<EMS installation path>/lib

- For Linux

1. Download the TIBCO Enterprise Message Service™ (EMS) Client package from [TIBCO eDelivery](#) and extract to TIB_ems_<version>_linux_x86_64/TIB_ems_<version>.

2. Run the following commands:
 - `sudo su`
 - `for f in tar/*; do tar -C / -xvf $f;`
3. Set the following variables in the `./bashrc` directory:
 - Export `LD_LIBRARY_PATH=<EMS installation path>/<version>/lib`
 - Export `EMS_HOME=<EMS installation path>/<version>`



Note: EMS installation for ARM-based machines is not supported. Therefore, EMS binary generation, Docker image generation, and unit testing is not supported in ARM-based machines.

Creating Connections

You must create connections before using the connectors in a flow. TIBCO Flogo® Enterprise uses the configuration provided in these connections to connect to the respective app, data sources, systems, or SaaS.

Before you begin

You must have valid accounts for the SaaS apps to which you want to connect. To create a connection, click the **Connections** tab on the TIBCO Flogo® Enterprise page.

To create a connection using a connector tile:

1. If this is the first connection you are creating, click the **Create connection** link. For subsequent connections, click the **Create** button on the **Connections** page.
2. Click the connector tile for which you want to create a connection.
3. Follow the instructions to configure the connection when prompted. For details on the connection dialog, refer to the specific connector documentation. You can do so by clicking the specific connector on the [Supported Flogo Connectors](#) page

i Note:

- You cannot use the same connection for multiple apps in Visual Studio Code. A connection is specific to a single app.
- You can have a maximum of four active Salesforce connections for one user at any time. If you create more than four connections for the same user, the first connection that you created gets deactivated. This limit is enforced by Salesforce.
- Make sure that the pop-up blocker in your browser is configured to always allow pop-ups from an app site. On macOS, clicking the link to the site results in the connection details page being unresponsive, so make sure to select the radio button for "**Always allow pop-ups from <site>**".
- Ensure that your app does not have unused connection(s). All connections within the app(used or unused) are initialized and started at runtime during app startup.

Editing Connections

You can edit the name and other settings of your connection.
To edit an existing connection:

Procedure


1. Click the **Connections** tab to open its page.
2. In the list of existing connections, click the connection that you want to edit. Edit the connection details in the connection details dialog that opens.
3. Click **Save**.

Deleting Connections

You can delete an existing connection.

Procedure

1. In TIBCO Flogo® Enterprise, click the **Connections** tab to open its page.

2. In the list of existing connections, hover over the connection name that you want to delete until you see that the **Delete connection** icon () appears at the end of the row.
3. Click the **Delete connection** icon.
4. On the confirmation dialog, click **Delete connection**.

Result


The selected connection is deleted.

Developing for Lambda

AWS Lambda is a serverless compute service provided by Amazon Web Services (AWS). Lambda functions automatically run pieces of code in response to specific events while also managing the resources that the code requires to run. Refer to the AWS documentation for more details on AWS Lambda.

Creating a Connection with the AWS Connector

You must create AWS connections before you use the Lambda trigger or Activity in a flow.

 **Note:** AWS Lambda is supported on the Linux platform only.

To create an AWS connection:

Procedure

1. In TIBCO Flogo® Extension for Visual Studio Code, click **Connections** to open its page.
2. Click the **AWS Connector** card.
3. Enter the connection details. Refer to the section [AWS Connection Details](#) for details on the connection parameters.

Field	Description
Name	A meaningful string identifying the AWS connection you are creating. The dropdown menu from which you select a connection when creating a Lambda Activity displays this string.
Description	A brief description of this connection
Region	Select a geographic area where your resources are located
Access key ID	Access ID to your AWS account
Secret Access key	Secret access key to your AWS account
Use Assume Role	<p>An AWS Assume role that allows you to assume a role from another AWS account. By default, it is set to False indicating that you cannot assume a role from another AWS account. When set to True, provide the following:</p> <p>Role ARN - Amazon Resource Name of the role to be assumed</p> <p>Role Session Name - Any string used to identify the assumed role session</p> <p>External ID - A unique identifier that might be required when you assume a role in another account</p> <p>Expiration Duration - The duration in seconds of the role session. The value can range from 900 seconds (15 minutes) to the maximum session duration setting that you specify for the role</p> <p>Refer to the AWS documentation for more details on these fields.</p>

4. Click **Save**.

Your connection gets created and is available for you to select in the dropdown menu when adding a **Lambda** Activity or trigger.

AWS Connection Details

To establish the connection to the AWS connector, you must specify the following configurations in the AWS Connector dialog.

The **AWS Connector** dialog contains the following fields:

Field	Description
Name	Specify a unique name for the connection that you are creating. This is displayed in the Connection Name dropdown list for all the AWS activities.
Description	A short description of the connection.
Custom Endpoint	(Optional) To enable the AWS connection to an AWS or AWS compatible service running at the URL specified in the Endpoint field, set this field to True .
Endpoint	<p>This field is available only when Custom Endpoint is set to True.</p> <p>Enter the service endpoint URL in the following format: <code><protocol>://<host>:<port></code>. For example, you can configure a MinIO cloud storage server endpoint.</p>
Region	Region for the Amazon connection.
Access key ID	Access key ID of the AWS account (from the Security Credentials field of IAM Management Console). For details, see the AWS documentation.
Secret access key	Enter the secret access key. This is the access key ID that is associated with your AWS account. For details, see the AWS documentation.
Session token	(Optional) Enter a session token if you are using temporary security credentials. Temporary credentials expire after a specified interval. For more information, see the AWS documentation.
Use Assume Role	<p>This enables you to assume a role from another AWS account. By default, it is set to False (indicating that you cannot assume a role from another AWS account).</p> <p>When set to True, provide the following information:</p>

Field	Description
	<ul style="list-style-type: none">• Role ARN - Amazon Resource Name of the role to be assumed• Role Session Name - Any string used to identify the assumed role session• External ID - A unique identifier that might be required when you assume a role in another account• Expiration Duration - The duration in seconds of the role session. The value can range from 900 seconds (15 minutes) to the maximum session duration setting that you specify for the role.
For details, see the AWS documentation.	

Creating a Flow with Receive Lambda Invocation Trigger

The **Receive Lambda Invocation** trigger allows you to create a Flogo flow to create and deploy as a Lambda function on AWS.

Refer to the [Receive Lambda Invocation](#) for details on the trigger.

To create a flow with the **Receive Lambda Invocation** trigger:

Procedure

1. Create a Flogo app.
2. Open the flow page.
3. From the **Triggers** palette, select **Receive Lambda Invocation** and drag it to the triggers area.
4. To configure a trigger, enter the JSON schema or JSON sample data for the operation. This is the schema for the request payload.
5. Click **Continue**.

A flow beginning with the **ReceiveLambdaInvocation** trigger gets created.

6. Click the **ReceiveLambdaInvocation** trigger tile and configure its properties.

Refer to the [Receive Lambda Invocation](#) for details on the trigger.

Deploying a Flow as a Lambda Function on AWS

After you have created the flow, you can deploy it as a Lambda function on AWS.

Before you begin

Note the following points:

- The flow must be configured with the **ReceiveLambdaInvocation** trigger.
- Run and Deploy actions do not deploy Lambda function on the AWS Lambda directly. You must deploy the Lambda function by following the steps in this section.
- Deploying an app having a **ReceiveLambdaInvocation** trigger on the TIBCO Platform does not deploy the app as a Lambda function on AWS Lambda. You must create a Linux/amd64 binary and deploy the binary by following the steps in this section.
- If the execution role name is not provided in the **ReceiveLambdaInvocation** trigger, then the Lambda function is created with the default **AWSLambdaBasicExecutionRole** role. It has the following Amazon CloudWatch permissions:
 - Allow: logs:CreateLogGroup
 - Allow: logs:CreateLogStream
 - Allow: logs:PutLogEvents

If a non-existing execution role is provided, then the user whose AWS credentials are used in the AWS connection should have the following permissions:

- iam:CreateRole
- sts:AssumeRole

To deploy a Flogo app as a Lambda function, user role can have access to following AWSLambda_FullAccess policy which has all the required access.

To deploy a flow as a Lambda function on AWS:

Procedure

1. Build your Flogo app(<myApp>) with the Linux/amd64 target. This is because Lambda deployments are Linux-based and building the binary for Linux/amd64 generates the appropriate artifact to deploy in your AWS Lambda function.
2. Add execution permission to the native Linux/amd64 executable file that you built. Run

```
chmod +x <myApp>-linux_amd64
```

3. You can deploy the <myApp>-linux_amd64 in one of two ways:

- If you are using a Linux environment to design, build, and deploy your apps, you can directly run the following command:

```
<LambdaTriggerBinary> --deploy lambda --aws-access-key <secret_key>
```

For example, myApp-Linux64 --deploy lambda --aws-access-key xxxxxxxxx



Note: Ensure that the aws-access-key is identical to the one configured in the Flogo UI for the selected AWS connection. This is used for validation with the aws-access-key configured as part of the AWS Connection within the UI and the value provided here does not overwrite the aws-access-key used while designing the app.

This approach of deploying to AWS Lambda works only on Linux platforms.

- .JSON file is passed to Lambda as environment variables.

If you are using a non-Linux environment to design, build, and deploy apps, then perform the following steps:

Procedure

1. Build your Flogo app (<myApp>) with the *Linux/amd64* target.
2. Rename the Flogo executable file to bootstrap. This is mandatory per new provided.al2 and provided.al2023 runtimes.
3. Compress the executable file and rename it to <myFunctionName>.zip.
4. From the AWS Lambda UI, create a Lambda function with Amazon Linux 2023 runtime.
5. Create a role or attach an existing role in the Execution role.
6. Click **Create function**.
7. Go to **Code source**, click **Upload from** and upload the compressed file.

After successful deployment, the Lambda function is created in the AWS Lambda console.

Deploying a Flow as a Lambda Function on AWS using AWS CLI

Procedure

1. Build your Flogo app (<myApp>) with the Linux/amd64 target. This is because Lambda deployments are Linux-based and building the binary for Linux/amd64 generates the appropriate artifact to deploy in your AWS Lambda function.
2. Rename the Flogo executable to bootstrap (this is mandatory as per new provided.al2 and provided.al2023 runtimes).
3. Zip the executable.
4. Zip myFunction.zip bootstrap.
5. Run the AWS CLI:

```
aws lambda create-function --function-name myFunction \  
--runtime provided.al2023 --handler bootstrap \  
--architectures x86_64 \  
--role arn:aws:iam::111122223333:role/lambda-ex \  
--region us-west-2 \  
--zip-file fileb://myFunction.zip
```

Receive Lambda Invocation

Use the **Receive Lambda Invocation** trigger for AWS to start your flow as a Lambda function. The **Receive Lambda Invocation** trigger can be configured only in blank flows. It must not be used with flows that are created with another trigger.

Trigger Settings

**Note:**

An app can contain only one Lambda trigger. An app that has a Lambda trigger cannot contain any other triggers including another Lambda trigger. Also, as the Lambda trigger supports only one handler per trigger, it can have only one flow attached to it. However, the apps that contain a Lambda trigger can contain blank flows that can serve as subflows for the flow attached to the Lambda trigger.

Field	Description
AWS Connection Name	Name of the AWS connector connection you want to use for the flow.
Execution Role Name	(optional) ARN of the role to be used to execute the function on your behalf. The role must be assumable by Lambda and must have CloudWatch logs permission execution role.

Output Settings

Enter the payload schema for the request received on the Lambda function invocation on AWS.

Map to Flow Inputs

This tab allows you to map the trigger output to flow input.

Field	Description
Function	Information about the Lambda function
Context	Envelope information about this invocation
Identity	Identity for the invoking users

Field	Description
ClientApp	Metadata about the calling app
API Gateway Request	Displays the elements in the payload schema that you entered on the Output Settings tab. The elements are displayed in a tree format.

Reply Settings

Enter a schema for the trigger reply in the Reply Data text box.

Map from Flow Outputs

Map the flow output to the trigger reply on this tab.

InvokeLambdaFunction

Use this Activity to invoke a specific Lambda function.

Settings

Field	Description
AWS Connection Name	Select an AWS connection.
ARN (Optional)	Amazon Resource Name. <div> <p>Note:</p> <ul style="list-style-type: none"> You can also specify the ARN on the Input tab. If you specify the ARN on both the tabs, the ARN on the Input tab is used. You must specify the ARN on at least one tab. Otherwise, the Activity returns an error at runtime. </div>

Input Settings

Field	Description
Payload Schema	Enter a JSON request schema for your payload that is used to invoke the Lambda function.

Input

The payload schema that you entered on the **Input Settings** tab is displayed in a tree format on the **Input** tab. Map the elements in the schema using the mapper or enter values for the element by manually typing the value in the mapper.

Field	Description
LambdaARN	Amazon Resource Name.

Note:

- You can also specify the ARN on the **Settings** tab. If you specify the ARN on both the tabs, the ARN on the **Input** tab is used.
- You must specify the ARN on at least one tab. Otherwise, the Activity returns an error at runtime.

Output Settings

Field	Description
Result Schema	The schema for the result that is expected from the Lambda function invokes the request.

Output

The Output tab displays the result schema that you entered on the **Output Settings** tab in a tree format.

Using Extensions

You can create extensions for Flogo or you can upload a Project Flogo extension on Flogo® Extension for Visual Studio Code (VS Code).

Configuring Extensions

To configure an extension:

i Note: This procedure assumes that you have the unzipped folder of the .zip extension file available.

i Note: Ensure that the go.mod file is present in the unzipped folder or else it generates an error.

Procedure

1. In **File Explorer**, create a folder under your workspace directory.
2. Add the unzipped version of your extensions file to the newly created folder.
3. In Visual Studio Code, navigate to **View > Command Palette**, search **Open Users Settings**, expand **Extensions**, and select **Flogo**.
4. Specify the path of your extensions folder in the **Extension:Local** field.
5. Refresh Visual Studio Code for the changes to be reflected.

i Note: Use the path of a local directory instead of a shared location path. Entering a shared location might not work.

Result

The extension is now available for you to use.

Activity extensions are available in the default category, functional extensions are available in the mapper functions, and trigger extensions are available in the trigger list.

Go Mod Compatibility

For custom extensions to run on Visual Studio Code, extension must be go mod compatible to generate app binary and run successfully.

Procedure

1. Add the unzipped version of your extensions file to go/src.
2. Navigate to go/src/{extension} and open Visual Studio Code .
3. Run the below commands on the terminal:
 - a. `go mod init {ref}`
exa. `go mod init github.com/TIBCOSoftware/tci-flogo/samples/extensions/AWSSQS/activity/sqssendmessage`
 - b. `go mod tidy`

Result

After the commands are executed successfully, go.mod and go.sum files are generated.

Download Extensions from Runtime

To download extensions from the TIBCO Cloud™ Integration environment:

Procedure

1. Select the Flogo icon in **Activity Bar**.
2. In the Runtime Explorer, select the desired TIBCO Cloud Integration runtime to download its extensions.
3. Ensure that you are connected to the runtime.
4. Right-click and select **Download Extensions**.

Result

The extensions are downloaded to your **extensions** folder as specified in the **Extension:Local** field.



Note: Only extensions promoted to org are downloaded to the specified folder.

Deleting Extensions

You can delete custom extensions by:

- Selectively removing the specific extension folder
- Removing the root **extensions** folder

Result

The extensions used in the Flogo app are not available after deletion.

Unit Testing

With unit testing, you can monitor the health of your application and detect errors in individual flows or activity levels.

While designing an application with multiple flows and activities, it becomes cumbersome to detect runtime errors at the flow and activity levels. Using unit testing, the errors at the micro level are easily handled. You can run unit testing at any phase of the development cycle to verify whether activities in the process are working as expected. Using testing processes in the development stage (before you push the app to the production environment) helps detect errors and identify issues at an early stage.

Terminologies in Unit Testing

- **Test case:** A test case is the individual unit for testing a flow. For a given set of inputs, the test case checks for a specific output for an activity or the flow output. The expected versus actual output is compared by adding assertions to the test case. A test case can have multiple assertions added on activities and flow output. The test case is considered as passed when all the assertions in that test case pass.
- **Assertion:** An assertion is a logical expression that evaluates to a Boolean value. The expected versus actual output is compared by using an assertion expression. For the passed assertion, the expression evaluates to true. A non-Boolean expression always evaluates to false.
- **Flow Input:** Flow input is a set of input data used to run the test on the given flow. Each test case has its own set of inputs.


Execute (Default)	This is the default mode for all activities. When an activity is set to this mode, it runs as per the definition and configuration and does not affect the unit test execution. You can use this option to reset your unit test configurations on the activity.
Assert on Outputs	Adds an assertion for flow output.
Assert on Error	Asserts if an activity generates an expected exception. It is applicable for a given input of the flow. Users should add proper error handlers through an error branch or by defining the error handler flow.
Mock Outputs	Use mock data for the activities that have an output. For more information, see Using Mock Data .
Mock Error	Simulates an exception with an optional message instead of executing the activity during a test run. It helps test the error handler design of the flow.
Skip Execution	Skip an activity in unit testing if the activity does not have any output. For example, you can skip activities, such as Sleep , eFTL Publish message , or StartaSubFlow . Note: To skip an activity with an output, you can mock it without any configuration data. For activities that do not have an output, you can select Skip Execution . For more information, see Skipping an Activity with No Output .

i Note: In the **Assert on Output** or **Asset on Exception** modes, you cannot delete all the assertions and save the configuration. You can set the **Execute (Default)** mode to reset or remove all the assertions.

Creating a Test Case for an Existing App

Before you start unit testing, you must create a test case.


Procedure

1. In the Explorer view, right-click the app for which you want to create a unit test and select **Create Unit Tests for Flogo App**.
2. Alternatively, when you click  (**Open Unit Test**) at the upper-right corner and the .flogotest file for the app is not found, you are prompted to create a unit test file as follows:
Could not find a Unit Test file for this App. (Searched for "<app name>.flogotest"). Do you want to create it?
Click **Create Unit Test File**.

Creating a Test Case Along with a New App

Before you start unit testing, you must create a test case.

Procedure

1. Open the **Explorer view**.
2. Navigate to the workspace where you want to create a Flogo app and click the dropdown icon next to that workspace folder.
3. Click  (**Create a New Flogo App**) to create a new Flogo app.
4. Add the name of the app in the top-most text bar that pops up and press Enter.

i Note: The app name must not contain any spaces. It must start with a letter or underscore. The app name can contain letters, digits, periods, dashes, and underscores.

5. Click **Create App and Unit Tests**.

Result

The app is created. A flow called New_flow is automatically created. You can now create one or more flows for the app. Two files are created in the workspace - <app name>.flogo and <app name>.flogotest. The <app name>.flogo file is the app's JSON file and the <app name>.flogotest is the app's unit test file.

i Note: If you copy a Flogo app that has a password property from one workspace to another and then open that app in the Flogo editor view, the password type values persist. As a workaround, remove the password type values manually before sharing the app.

Defining Flow Input

For a particular activity that has a flow input configured in the actual process, you must assign the flow input parameters before you run a test case. You can add separate test cases for each flow input.

i Note:

- If the flow input is configured for the activity, then you must define the flow input value when running a test case, otherwise the test case fails. However, if the flow input is not configured for the activity, then you need not define the flow input value when running a test case.
- For inputs containing single objects, you must enter the input values at the root level.
- For mapping an array of objects, you must provide inputs at the array root level. Click the root of the array to input values for all objects. You cannot configure the input at the array element level.

To define the flow input parameter:

Procedure

1. In the **Explorer view**, click the <app name>.flogotest file.
2. Under the **Main Flow** tab, provide the required flow input details and save the app.

Creating Assertions

To compare the actual and expected output, you can add multiple assertions on an activity, a flow output, or an error handler . The assertion expression always evaluates to a boolean value.

i Note: You cannot create assertions for the activities that do not return output. Similarly, you cannot create assertions for the flow output when it has no outputs.

To add unit test assertions for a test case, perform the following steps:

Procedure

1. On the Unit Test page, select one of the flows to add an assertion:
 - Error Handler: On the **Error Handler** tab, click **<Activity Name>**.
 - Main Flow: On the **Main Flow** tab, click **<Activity Name>** or **Flow Output**.

The **Unit Test Data Configuration** dialog opens.

2. On the **I want to** dropdown menu, select one of the following assertions:
 - **Assert on Outputs**
 - **Assert on Error**

i Note: The assertion types in the dropdown menu are based on the configuration of the selected activity.

3. Click **New Assertion** to create one with a default name.
4. Map the **Available Data** from assertions dialog with the appropriate values and click **Save**.

i Note:

- When asserting an object, it is a best practice to assert each property individually instead of the entire object.
- You cannot save changes if you delete all assertions. To remove all existing assertions, select a different mode on the **I want to** dropdown menu.

Creating Assertions for Flow Output

i Note: You cannot create assertions for the flow output when it has no outputs.

You can add assertions to the flow output to verify that the flow generates the correct data by comparing the actual output against predefined assertions.

To create assertions for flow output, see [Creating Assertions](#).

Creating Assertions for the Error Handler

You can also add test cases for the flow designed in the error handler. A unit test case designed for the error handler flow is run to detect any runtime exceptions or errors in the flow implementation.

To create assertions for the error handler, see [creating assertions](#).

Skipping an Activity with No Output

You can skip a particular activity such as **Sleep**, **eFTL Publish message**, **StartaSubFlow** and others in unit testing if that activity does not have any outputs.

To skip an activity with no output:

Procedure

1. In the **Explorer View**, click <app name>.flogotest.
2. On the **Main Flow/Error Handler** tab, select the activity you want to skip.
3. On the selected activity dialog, click **Yes, Skip**.

i Note: When you run the **Unit Tester**, the activities you skipped are displayed as "Activity Mocked".

You can later enable the activity you have skipped using the same steps.


Validating a Test Case

Before you run a test case, you can validate the test cases when you land on the unit testing mode.

If there are any errors in mapping expressions, an error sign pops up on the activity level assertions, or on the mock data. If there are any errors in the mapping expressions, the error sign also pops up on the flow inputs and flow outputs.

To validate a test case:



Procedure

1. Click  (**Open Unit Test**) in the Flogo app editor mode at the upper-right corner to switch to the unit testing mode and see if there are any validation errors.
Alternatively, in the **Explorer View**, click <app name>.flogotest.
An error sign is displayed on the assertion level of the activity, flow input, and flow output. An error sign is also displayed on the test case level of the particular activity that has the errors.
2. Fix the validation errors.

Running a Test Case

After creating a test case, it is ready to run.

Procedure

1. Go to the activity bar and click the **Testing** action item .
2. Ensure the .flogotest file is open.
3. Click the **Run Test** icon () next to the test case.

After the test case run is complete, the result is generated.

The test results are displayed on the terminal with the total number of test cases, including the number of passed and failed test cases. It also displays the total number of assertions and the number of passed and failed assertions on activities in the flow, activities in the error handler, and in the flow output. The result for assertions on the flow output is displayed only if the assertion is added to the flow output.

i Note: When running a binary, the test suites or test cases are run by default irrespective of whether they are hidden or unhidden.

- ✓ Tip:**
- Do not close the result terminal to modify your test case. You can minimize the window, make the changes, and retest the case.
 - You can edit, delete, or copy a test case or a test assertion at any point.
 - For an active unit test case, if you change app-level schemas or app properties, close the session and rerun the test case.

Hiding Test Cases

Hidden test cases are not run when you run the <app name>.flogotest file.

- To hide a test case, right-click on the test case and select **Hide Test**.
- To unhide all the tests, select the <app name>.flogotest file, click the **Filter** icon (🔍) and then click **Unhide All Tests**.
- To always show hidden tests, select the <app name>.flogotest file, click the **Filter** icon (🔍) and then click **Show Hidden Tests**.

Using Mock Data

In unit testing, you can mock the data for the unit that is being tested. This is useful during unit testing so that the external dependencies are no longer a constraint to the unit under test. Using mock data, the dependencies are replaced by closely controlled replacements that simulate the behavior of the real ones.



You can use the mock data for the activities that have an output. Expressions and functions are not evaluated in the values given to mock outputs, the input provided is passed as is.

In unit testing, you can either use assertions or mock data to test the activities.

Running a Test Case Using a Runtime Environment

You can run a test case using a runtime environment.


Procedure

1. Go to the activity bar and click the **Testing** action item ()
2. Make sure the .flogotest file is open.
3. To select a runtime environment, click the dropdown icon next to the **Run Tests** icon () . You can select either **Local Runtime**, **TIBCO Platform**, or **TIBCO Cloud™**

Integration (TCI).

The default profile is **Local Runtime**. To change the default profile, select **Select Default Profile** from the dropdown menu and select the profile that needs to be made the default.

For more information on configuring these profiles, see [Configuring the Runtime](#).

4. Click the **Run Tests** icon () to run the test with the selected environment.

After the test case run is completed, the result is generated.

The test results are displayed on the terminal with the total number of test cases, which include the number of passed and failed test cases. It also displays the total number of assertions and the number of passed and failed assertions on activities in the flow, activities in the error handler, and in the flow output. The result for assertions on the flow output is displayed only if the assertion is added to the flow output.

Creating and Running a Test Suite

You can use the **Test Suite** feature to combine different test cases and run them at once.

Creating a Test Suite

To create a test suite, perform the following steps:

Procedure

1. In the Explorer View, click <app name>.flogotest and then click **Test Suites**. The **Configure Test Suites** dialog opens.
2. In the **Configure Test Suites** dialog, click **New Test Suite**. A test suite with a default name gets created.


3. To add test cases to the test suite for the first time, click **Add test cases**. Select the test cases to be added to the suite and click **Save**.
4. To remove test cases or add more test cases to the test suite, click **Add/Remove Tests**. Select or deselect the test cases to be included in the test suite and click **Save**.

Running a Test Suite

After you create a test suite, the test suite is ready to run.

To run a test suite:

Procedure

1. Navigate to the activity bar and click the testing action item.
2. Click  (**Run Test** icon) next to the respective test suite.

After the test suite run is completed, the result is generated.

A result terminal displays the total number of test suites and test cases with the number of passed and failed test suites and test cases.



Note:

- You can hide the test suites by right-clicking on the test suite to exclude it from **Run all Test Suites**.
- When the engine is running in unit test mode, it does not fail fast and continues on connection errors. The connections have a retry mechanism, then the start-up time increases considerably. If any activity that uses connections is not mocked, the test case generates an error.

Unit Testing for the CI/CD

This feature allows you to unit test your Flogo app using the app executable. Once you have built the executable for a Flogo app, you can run a unit test using the `test` command. This feature is also useful to automate the testing process for activities in development in the CI/CD pipeline.

Before you begin

The app executable must be readily accessible on the machine from which you plan to test it.

Follow these steps to get help on the test command:

Procedure

1. Open a command prompt or terminal window depending on your platform.
2. Navigate to the folder where you stored the app executable.
3. Run the following command to get the online help on the test command:

- On Windows: `<app-executable> --test --test-file`
- On Macintosh: `./<app-executable> -test --test-file`
- On Linux: `./<app-executable> -test --test-file`

This command outputs the usage for the test command along with some examples.

4. Run the command with the appropriate option to test your app.

The output of the command generates the `.testresult` file for the unit test suites or test cases that are run.


App Configuration Management

Flogo allows you to externalize app configuration using app properties so that you can run the same app binary in different environments without modifying your app. It integrates with configuration management systems such as Consul and AWS Systems Manager Parameter Store to get the values of app properties at runtime.

You can switch between configuration management systems without modifying your app. You can do this by running the command to set the configuration-management-system-specific environment variable from the command line. Since the environment variables are set for the specific configuration management system, at runtime, the app connects to that specific configuration management system to pull the values for the app properties.

Consul


The Consul provides a key/value store for managing app configuration externally. TIBCO Flogo® allows you to fetch values for app properties from Consul and override them at runtime.

 **Note:** This section assumes that you have set up Consul and know-how Consul is used to store service configuration. Refer to the Consul documentation for consul-specific information.

A Flogo app connects to the Consul server as its client by setting the FLOGO_APPS_PROPS_CONSUL environment variable. At runtime, you must provide the Consul server endpoint for your app to connect to a Consul server. You have the option to enter the values for the Consul connection parameters. You can create a file that contains the values and use the file as an input.

Consul can be started with or without `acl_token`. If using an ACL token, make sure to have the ACL configured in Consul.

Using Consul

 **Note:** The information in this section is applicable for an app executable only.

Below is a high-level workflow for using Consul with your Flogo app.

Before you begin


You must have access to Consul.

Set up Consul and understand how Consul is used to store service configuration. For information on Consul, refer to the Consul documentation.

To use Consul to override app properties in your app (properties that were set in the Flogo app):

Procedure

1. Export your app binary.
2. Configure key/value pairs in Consul for the app properties whose values that you want to override. At runtime, the app fetches these values from the Consul and uses them to replace the default values that were set in the app.
3.

 **Important:** When setting up the Key in Consul, make sure that the Key name matches exactly with the corresponding app property name in the **Application Properties** dialog in Flogo. If the property name does not match exactly, a warning message is displayed, and the app uses the default value for the property that you configured in Flogo.
4. Set the FLOGO_APP_PROPS_CONSUL environment variable to set the Consul server connection parameters. See [Setting the Consul Connection Parameters](#) for details.

Consul Connection Parameters

Provide the following configuration information during runtime to connect to the Consul server.

Property Name	Required	Description
server_address	Yes	Address of the Consul server, which could be run locally or elsewhere in the cloud.
key_prefix	No	<p>Prefix to be prepended to the lookup key. This is essentially the hierarchy that your app follows to get to the Key location in the Consul. This is helpful in case the key hierarchy is not fixed and may change based on the environment during runtime. It is also helpful in case that you want to switch to a different configuration service such as the AWS param store. Although it is a good idea to include the app name in the key_prefix, it is not required. key_prefix can be any hierarchy that is meaningful to you.</p> <p>As an example of a key_prefix, if you have an app property (for example, Message) that has two different values depending on the environment from which it is being accessed (for example, dev or test</p>

Property Name	Required	Description
		environment), your <key_prefix> for the two values can be /dev/<APPNAME>/ and /test/<APPNAME>/. At run time, the right value for Message is picked up depending on which <key_prefix> you specify in the FLOGO_APP_PROPS_CONSUL environment variable. Hence, setting a <key_prefix> allows you to change the values of the app properties at runtime without modifying your app.
acl_token	No	<p>Use this parameter if you have key access protected by ACL. Tokens specify which keys can be accessed from the Consul. You create the token on the ACL tab in Consul.</p> <p>During runtime, if you use the acl_token parameter, Key access to your app is based on the token you specify.</p> <p>To protect the token, encrypt the token for the key_prefix where your Key resides and provides the encrypted value of that token by prefixing the acl_token parameter with SECRET. For example, "acl_token": "SECRET:QZLOrtN3gOEpXgUuud6jprgo/WzLR7j+Twv28/4KCp7573snZWohGuQauuR2o/7TJ+ZLQ==". Note that the encrypted value follows the key_prefix format.</p> <p>Provide the encrypted value of the token as the SECRET. SECRETS get decrypted at runtime. To encrypt the token, you obtain the token from the Consul. Then, encrypt it using the app binaryexecutable by running the following command from the directory in which your app binaryexecutable is located:</p> <pre>./<app_binary> --encryptsecret <token_copied_from_Consul></pre> <p>The command outputs the encrypted token that you can use as the SECRET.</p> <p>Note: Since special characters (such as `! < > & `) are shell command directives, if they appear in the token string when encrypting the token, you must use a backslash (\) to escape such characters.</p>

Property Name	Required	Description
insecure_connection	No	Set to True if you want to connect to a secure Consul server without specifying client certificates. This should only be used in test environments. Default: False

Setting the Consul Connection Parameters

You set the values for app properties that you want to override by creating a Key/Value pair for each property in Consul. You can create a standalone property or a hierarchy that groups multiple related properties.

Before you begin

This section assumes that you have access to Consul and are familiar with its use.

To create a standalone property (without hierarchy), you simply enter the property name as the name of the Key when creating the Key in Consul. When you create a property within a hierarchy, enter the hierarchy in the following format in the **Create Key** field in Consul: <key_prefix>/<key_name> where <key_prefix> is a meaningful string or hierarchy that serves as a path to the key in Consul and <key_name> is the name of the app property whose value you want to override.

For example, in dev/Timer/Message and test/Timer/Message, dev/Timer and test/Timer are the <key_prefix> that could stand for the dev and test environments and Message is the key name. During runtime, you provide the <key_prefix> value that tells your app the location in Consul from where to access the property values.



Warning: The Key name in Consul must be identical to its counterpart in the **Application Properties** dialog in Flogo. If the key name does not match exactly, a warning message is displayed, and the app uses the default value that you configured for the property in Flogo.



Warning: A single app property, for example Message, is looked up by your app as either Message or <key_prefix>/Message in Consul. An app property within a hierarchy such as x.y.z is looked up as x/y/z or <key_prefix>/x/y/z in Consul. Note that the dot in the hierarchy is represented by a forward slash (/) in Consul.

After you have configured the app properties in Consul, you need to set the environment variable FLOGO_APP_PROPS_CONSUL with the Consul connection parameters for your app to connect to the Consul. When you set the environment variable, it triggers the app to run, which connects to the Consul using the Consul connection parameters you provided and pulls the app property values from the key_prefix location you set by matching the app property name with the key_name. Hence, the Key names must be identical to the app property names defined in the **Application Properties** dialog in Flogo.

You can set the FLOGO_APP_PROPS_CONSUL environment variable by placing the properties in a file and using the file as input to the FLOGO_APP_PROPS_CONSUL environment variable.

Setting the Consul Parameter Values Using a File

To set the parameter values in a file, create a .json file, for example, consul_config.json containing the parameter values in key/value pairs. Here is an example:

```
{
  "server_address": "http://127.0.0.1:32819",
  "key_prefix": "/dev/<APPNAME>/",
  "acl_token": "SECRET:b0UaK3bTyD9wN+ZJkmlKRmojhAv+"
}
```

Place the consul_config.json file in the same directory that contains your app binary.

Run the following from the location where your app binary resides to set the FLOGO_APP_PROPS_CONSUL environment variable. For example, to use the consul_config.json file from the example above, run:

```
FLOGO_APP_PROPS_CONSUL=consul_config.json ./<app_binary_name>
```

The command extracts the Consul server connection parameters from the file and connects to Consul to pull the app properties values and runs your app with those values.

Consul properties can also be run using Docker by passing the same arguments for the Docker image of a binary app.

AWS Systems Manager Parameter Store

AWS Systems Manager Parameter Store is a capability provided by AWS Systems Manager for managing configuration data. You can use the Parameter Store to store configuration parameters centrally for your apps.

Your Flogo app connects to the AWS Systems Manager Parameter Store server as its client. At runtime, you are required to provide the Parameter Store server connection details by setting the `FLOGO_APP_PROPS_AWS` environment variable for your app to connect to the Parameter Store server. You have the option to enter the values for the Parameter Store connection parameters by creating a file that contains the values and using the file as input.

Using the Parameter Store

The following section gives a high-level workflow for using AWS Systems Manager Parameter Store with TIBCO Flogo®.

Before you begin

This section assumes that you have an AWS account, have access to the AWS Systems Manager, and know how to use the AWS Systems Manager Parameter Store. Refer to the AWS documentation for information on the AWS Systems Manager Parameter Store.

Overview

To use the Parameter Store to override app properties set in Flogo:

1. Build an app binary that has the app properties already configured in Flogo.
2. Configure the app properties that you want to override in the Parameter Store. At runtime, the app fetches these values from the Parameter Store and uses them to replace the default values that were set in the app.
3. Set the `FLOGO_APP_PROPS_AWS` environment variable to set the Parameter Store connection parameters from the command line.

When you run the command for setting the `FLOGO_APP_PROPS_AWS` environment variable, it runs your app, connects to the Parameter Store, and fetches the overridden values for the app properties from the Parameter Store. Only the values

for properties that were configured in the Parameter Store are overridden. The remaining app properties get their values from the **Application Properties** dialog.

See the [Setting the Parameter Store Connection Parameters](#) and [Parameter Store Connection Parameters](#) sections for details.

Parameter Store Connection Parameters

To connect to the AWS Systems Manager Parameter Store, provide the configuration below at runtime.

Property Name	Required	Data Type	Description
access_key_id	Yes	String	<p>Access ID for your AWS account. To protect the access key, an encrypted value can be provided in this configuration.</p> <div> <p>Note: The encrypted value must be prefixed with SECRET: For example, SECRET:b0UaK3bTyD9wN+ZJkmlKRmojhAv+</p> </div> <p>This configuration is optional if use_iam_role is set to true.</p>
secret_access_key	Yes	String	<p>Secret access key for your AWS account. This account must have access to the Parameter Store. To protect the secret access key, an encrypted value can be provided in this configuration.</p> <div> <p>Note: The encrypted value must be prefixed with SECRET: For example, SECRET:b0UaK3bTyD9wN+ZJkmlKRmojhAv+</p> </div> <p>This configuration is optional if use_iam_role is set to true.</p>
region	Yes	String	<p>Select a geographic area where your Parameter Store is located. This configuration is optional if use_iam_role is set to true and your Parameter Store</p>

Property Name	Required	Data Type	Description
			is configured in the same region as the running service. When running in AWS services (for example, EC2, ECS, EKS), this configuration is optional if the Parameter Store is in the same region as these services.
param_prefix	No	String	<p>This is essentially the hierarchy that your app follows to get to the app property location in the Parameter Store. It is the prefix to be prepended to the lookup parameter. This is helpful in case the parameter hierarchy is not fixed and may change based on the environment during runtime.</p> <p>This is also helpful in case that you want to switch to a different configuration service such as the Consul KV store.</p> <p>As an example of a param_prefix, if you have an app property (for example, Message) that has two different values depending on the environment from which it is being accessed (for example, dev or test environment), your param_prefix property for the two values can be /dev/<APPNAME/ and /test/<APPNAME/. At run time, the right value for Message is picked up depending on which param_prefix you specify in the FLOGO_APP_PROPS_AWS environment variable. Hence, setting a param_prefix allows you to change the values of the app properties at runtime without modifying your app.</p>
use_iam_role	No	Boolean	Set to true if the Flogo app is running in the AWS services (such as EC2, ECS, and EKS) and you want to use the IAM role (such as instance role or task role) to fetch parameters from the Parameter Store. In that case, access_key_id and secret_access_key are not required.

Property Name	Required	Data Type	Description
session_token	No	String	Enter a session token if you are using temporary security credentials. Temporary credentials expire after a specified interval. For more information, see the AWS documentation.

Setting the Parameter Store Connection Parameters

You can use the AWS Systems Manager Parameter Store to override the property value set in your Flogo app. You do so by creating the property in the Parameter Store and assigning it the value with which to override the default value set in the app. You can create a standalone property or a hierarchy (group) in which your property resides.

Before you begin

This section assumes that you have an AWS account and the Parameter Store and are familiar with its use. Refer to the AWS documentation for more information on the Parameter Store.

To create a standalone property (without hierarchy), you simply enter the property name when creating it. To create a property within a hierarchy, enter the hierarchy in the following format when creating the property: `<param_prefix>/<property_name>`, where `<param_prefix>` is a meaningful string or hierarchy that serves as a path to the property name in Parameter Store and `<property_name>` is the name of the app property whose value you want to override.

For example, in `dev/Timer/Message` and `test/Timer/Message/dev/Timer` and `test/Timer` are the `<param_prefix>` which could stand for the dev and test environments respectively, and `Message` is the key name. During runtime, you provide the `<param_prefix>` value, which tells your app the location in the Parameter Store from where to access the property values.

**Warning:**

- A single app property, for example, Message, is looked up by your app as either Message or <param_prefix>/Message in the Parameter Store. An app property within a hierarchy such as x.y.z is looked up as x/y/z or <param_prefix>/x/y/z in the Parameter Store. Note that the dot in the hierarchy is represented by a forward slash (/) in the Parameter Store.
- The parameter name in the Parameter Store must be identical to its counterpart (app property) in the **Application Properties** dialog in Flogo. If the parameter names do not match exactly, a warning message is displayed, and the app uses the default value that you configured for the property in Flogo.

After you have configured the app properties in the Parameter Store, you need to set the environment variable, FLOGO_APP_PROPS_AWS, with the Parameter Store connection parameters for your app to connect to the Parameter Store. When you set the environment variable, it triggers your app to run, which connects to the Parameter Store using the Parameter Store connection parameters you provided and pulls the app property values from the param_prefix location you set by matching the app property name with the param_name. Hence, the property names must be identical to the app property names defined in the **Application Properties** dialog in Flogo.

You can set the FLOGO_APP_PROPS_AWS environment variable by manually placing the properties in a file and using the file as input to the FLOGO_APP_PROPS_AWS environment variable.

If your Container is Not running on ECS or EKS

If the container in which your app resides is running external to ECS, you must enter the values for access_key_id and secret_access_key parameters when setting the FLOGO_APP_PROPS_AWS environment variable.

Setting the Parameter Store values using a file

To set the parameter values in a file, create a .json file, for example, aws_config.json containing the parameter values. Here is an example:

```
{
  "access_key_id": "SECRET:b0UaK3bTyD9wN+ZJkmlKRmojhAv+",
  "param_prefix": "/MyFlogoApp/dev/",
  "secret_access_key": "SECRET:b0UaK3bTyD9wN+ZJkmlKRmojhAv+",
  "region": "us-west-2",
```

```
"session_
token":"SECRET:1UBrElezye8W1mmx7NLAiQzopmp58kUa02XdpmxYqVvkGKUrdN+wgCeH3mx
Z"
}
```

Place the `aws_config.json` file in the same directory, which contains your app binary.

Run the following from the location where your app binary resides to set the `FLOGO_APP_PROPS_AWS` environment variable. For example, to use the `aws_config.json` file from the example above, run:

```
FLOGO_APP_PROPS_AWS=aws_config.json ./<app_binary_name>
```

This connects to the Parameter Store, pulls the overridden app properties values from the Parameter Store, and runs your app with those values.

If your Container is running on ECS or EKS

In case your Flogo apps are running in ECS and intend to use the EC2 instance credentials, set `use_iam_role` to `true`. The values for `access_key_id` and `secret_access_key` are gathered from the running container. Ensure that the ECS task has permission to access the param store.

The IAM role that you use must have permissions to access the parameters from the AWS Systems Manager Parameter Store. The following policy must be configured for the IAM role:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "ssm:GetParamaters",
        "ssm:GetParamatersByPath",
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

The following is an example of how to set the `FLOGO_APP_PROPS_AWS` environment variable when your container is running on ECS. Notice that the values for `access_key_id` and `secret_access_key` are omitted:


```
FLOGO_APP_PROPS_AWS="{\"use_iam_role\":true,\"region\": \"us-west-2\"}\" ./Timer-darwin-amd64
```

Overriding an App Property at Runtime

The following section discusses how to override an app property at runtime in the following scenarios:

- [Without restarting or redeploying the app](#)
- [With new values](#)
- [With new values in the API](#)

Overriding Properties without Restarting or Redeploying the App

When using config management services, such as Consul or AWS Params store, you can update or override an app property at runtime without restarting or redeploying the app.

i Note: Currently, this functionality is only available for app properties mapped in activities. It is not available for app properties in triggers and connections.

Before you begin

Set the following environment variables:

Environment Variable	Description
FLOGO_APP_PROP_RECONFIGURE=true	Specifies that app properties can be updated or overridden at runtime.
FLOGO_APP_PROP_SNAPSHOTS=true	Used along with FLOGO_APP_PROP_RECONFIGURE. If you do not want your application to pick the updated app properties dynamically for a running flow, set this

	variable to true. The updated values are effective only for new flows and not existing ones.
FLOGO_HTTP_SERVICE_PORT=<port number>	Specifies the service port. For apps running in TCI, you do not need to specify the port. The default is 7777.
FLOGO_APP_PROPS_CONSUL="{\"server_address\":\"http://127.0.0.1:8500\"}"	Specifies the Consul server address.

Overriding Values by Specifying New Values

1. In the Flogo app, create an app property and map it to the activities as required.
2. Create the same key as the app property and add some value.
3. Run the app with the environment variables in the "Before you begin" section. The app takes all the configured values.
4. Update the values.
5. To reconfigure the app property values, use the API as follows:

```
curl -X PUT localhost:7777/app/config/refresh
```

A successful response is returned from the API.

6. Open the app property update logs to verify that the new app property values are used by the activities.

Overriding Values by Specifying New Values in the API Directly

You can specify the new values of app properties directly through the body of the reconfigure API. This method takes priority over any other resolver specified.

For example:

```
curl -X PUT -H "Content-Type: application/json" -d '{"Property_1":"Value"}'  
localhost:7777/app/config/refresh
```

Important Considerations

- If the same property exists, the property from the body of the reconfigure API is used.
- Any new request on the API does not save property values provided on a previous request.
- Properties mentioned in an earlier request and not mentioned in the new request take values if present from other resolvers mentioned or the last saved value.
- Properties that are not mentioned in any resolver take the default values.

Azure Key Vault Secrets

Azure Key Vault stores and manages secrets, such as passwords and database connection strings. TIBCO Flogo® Enterprise retrieves values for Flogo app properties from Azure Key Vault Secrets and overrides them at runtime.

Integrating Azure Key Vault

To integrate the Flogo app with Azure Key Vault, set the following environment variables for your application:

- FLOGO_APP_PROPS_AZURE_KEYVAULT: Set to true to enable Azure Key Vault integration.
- FLOGO_AZURE_KEYVAULT_NAME: Specify the name of your Azure Key Vault.

Authentication Methods

Flogo supports the following authentication methods for accessing Azure Key Vault:

- Service Principal with Secret

- Managed Identities for Azure Resources

To configure Azure Key Vault credential management, set the following environment variables at runtime based on your authentication method.

For Service Principal with Secret

Variable	Description
FLOGO_APP_PROPS_AZURE_KEYVAULT	Set this to true to integrate the Flogo app with Azure Key Vault.
FLOGO_AZURE_KEYVAULT_NAME	Specify the Azure Key Vault name.
AZURE_TENANT_ID	The Microsoft Entra tenant (directory) ID.
AZURE_CLIENT_ID	The client (application) ID of an App Registration in the tenant.
AZURE_CLIENT_SECRET	The client secret generated for the App Registration.

For Managed Identities for Azure Resources

Variable	Description
FLOGO_APP_PROPS_AZURE_KEYVAULT	Set this to true to integrate the Flogo app with Azure Key Vault.
FLOGO_AZURE_KEYVAULT_NAME	Specify the Azure Key Vault name.



Note:

- The Flogo app always fetches the current Azure Key Vault Secret version.
- The identity used by your application must have at least the **Key Vault Secrets User** role to retrieve secrets from Azure Key Vault.
- If Azure Key Vault uses access policies to manage permissions, then the identity used by your application must have at least **Get Secret** permissions. It is required to retrieve secrets from Azure Key Vault.

Setting Azure Key Vault Secrets

To override the app property values, create a key-value pair for each property in Azure Key Vault Secrets. You can create standalone properties or organize them in a hierarchy.

- For a standalone property, use the property name as the secret name in Azure Key Vault. If the property name contains an underscore (_), it is replaced with a hyphen (-).

For example, if the app property name is `Property_1`, then the secret lookup is done with name `Property-1` in Azure Key Vault.

- For hierarchical properties, if the property name contains an end period (.) or an underscore (_), these are replaced with a hyphen (-).

For example, if the app property name is `Group_1.Group_2.Property_1`, then the secret lookup is done with name `Group-1-Group-2-Property-1` in Azure Key Vault.



Warning: The secret name in Azure Key Vault must exactly match the property name in the Flogo Application Properties dialog (after replacing underscores and end periods with hyphens). If the names do not match or if the secret is disabled, the Flogo application displays a warning and uses the default property value.

Running Apps Locally

Configuring the Runtime

To run Flogo apps, you must first add a runtime environment.

Procedure

1. Select the Flogo icon in **Activity Bar**.
2. In the Runtime Explorer, click **Add a New Runtime** or if a runtime exists click **+**.
3. In the input field on the top, select the runtime type:
 - Local Runtime
 - TIBCO Platform
 - TIBCO Cloud™ Integration (TCI)

Creating Local Runtime

In the **New runtime** dialog, specify the following details:

- **Name:** name for the runtime.
- **Runtime type:** it is Local Runtime by default.
- **EMS Home:** path to EMS Installation directory.

i Note: To build the binary, the user must have TIBCO Enterprise Message Service™ (EMS) Client 10.3.0 or above.

- **IBM MQ Home:** path to IBM MQ Installation directory.

Always Recreate Flogo Executable Checkbox

The **Always Recreate Flogo Executable** checkbox optimizes the Flogo Executable build time.

i Note: This checkbox is not applicable when running unit tests or running the test in play mode. The Flogo app executable is reused by default. If you are in development, delete the binary every time for the code changes to reflect.

Best Practices

- If you want to rebuild a Flogo executable every time, select this checkbox.
- If the build process is slow or there are no significant changes to application dependencies, do not select this checkbox.

By default, this checkbox is not selected. An existing Flogo executable is reused if the following conditions are met:

- Golang, Flogo Runtime, and connector versions are unchanged.
- No new activities, triggers, or connections are added or removed.
- No new custom extension is added or removed.

i Note: Note: You must refresh Visual Studio Code for any changes in the custom extension to reflect.

- No new function is used in the mapper.

If any of these conditions are not met, the executable is rebuilt.



Caution:

- The Flogo executable must always be started using the `--app` option.
- The embedded app inside the executable might differ from the working copy.
- Exporting an application from the binary might result in an out-of-sync version.

Custom Extension Development: As a best practice, select the **Always Recreate Flogo Executable** checkbox for a fresh build every time.

Creating a Runtime for TIBCO Platform

In the **New runtime** dialog, specify the following details:

- **Name:** name for the runtime
- **Platform URL:** URL of the Control Plane

After entering the Platform URL, click **Authenticate**. This redirects to the TIBCO Platform URL where you need to sign in. After a successful sign-in, you will be redirected to Visual Studio Code.

i Note: You must select the TIBCO End User Agreement checkbox to proceed. If you do not select this checkbox, you cannot deploy apps.

- **Data Plane:** Select the desired data plane from the dropdown menu to which you want to connect.
- **Capability Base URL:** This is the Capability Public Base URL, which is present in the Flogo Capability that is provisioned in a data plane.
- **TIBCO Developer Hub URL** (Optional): Enter the URL of the TIBCO Developer Hub that you want to connect. The Developer Hub URL is auto-populated if you have provisioned Developer Hub in the Data Plane.
- **Template tags** (Optional): Enter the tags to filter the TIBCO Developer Hub templates.
- **Documentation URL** (Optional): URL for platform documentation.

If you use custom certificates for the TIBCO Control Plane setup, you must perform one of the following procedures:

- Configuring the root CA certificate
- Disabling certificate validation

Configuring the Root CA Certificate

If you have access to the root CA certificate, perform the following steps:

1. Set the `NODE_EXTRA_CA_CERTS` environment variable to point to the root CA file. For example, `NODE_EXTRA_CA_CERTS=C:\Users\flogouser\Documents\projects\test\mkcerts\rootCA.pem`.
2. Restart Flogo Extension for Visual Studio Code.

If you are using leaf or intermediate certificates, then perform the following steps to chain these certificates with the Root CA certificate:

1. Combine the certificates into a single file, placing the leaf certificate first, followed by any intermediate certificates, and finally the root certificate at the end.
2. Then set the `NODE_EXTRA_CA_CERTS` environment variable to point to the file you just created.
3. Restart Flogo Extension for Visual Studio Code.


Disabling Certificate Validation

This is not a best practice. However, if you do not have access to the root CA certificate, perform the following steps:

1. Set the value of the `NODE_TLS_REJECT_UNAUTHORIZED` environment variable to 0.
2. Restart Flogo Extension for Visual Studio Code

Creating a Runtime for TIBCO Cloud Integration (TCI)

To access TIBCO Cloud Integration apps, you must enable platform APIs from the Cloud APIs for the organization.

 **Note:** To connect to TIBCO Cloud Integration, ensure that platform API access is enabled by the organization administrator.

In the **New runtime** dialog, specify the following details:

- **Region**
- **Name**
- **OAUTH Token.** For more information, see [How to generate an OAUTH Token?](#)

The following **Advanced** fields are populated by default:

- **Homepage URL:** `https://cloud.tibco.com/`
- **Documentation URL:** `https://integration.cloud.tibco.com/docs/index.html`
- **Designtime URL:** `https://integration.cloud.tibco.com/applications/details/flows/{APPID}?applicationType=flogo`
- **Default App Export File:** The default name with which you want to export an app


After connection to TIBCO Cloud Integration runtime is established, you can see the list of all the apps present in your org by opening the tree dropdown. You can take the following actions for these apps present in your Visual Studio Code org by hovering over the apps:

- **Add app to favorites:** Click to set an app as your favorite app that shows under the Favorite Remote Apps section.
- **Export and open JSON:** Click to export the app.flogo file of the app and download it in your workspace.
- **Open in Browser:** Click to open the app details page, you need to log in to TCI to view this page.
- **Export Binary:** Click to create a binary of the app and download it in your workspace.
- **Export and Run Binary:** Click to create, download, and run the binary of the app.

How to generate an OAUTH Token?

To create a runtime configuration, you need to first generate an OAUTH token.

Procedure

1. Navigate to **Settings** under your user account icon  on [TIBCO Cloud Integration](#).
2. On the left pane, click **OAUTH Access Tokens**
3. Click **Generate token** to enter the **Generate OAuth 2 token** dialog.
4. Specify the following details:
 - a. **Name:** name of your token
 - b. **Valid for duration:** number of days
 - c. **Can be used by these domains:**
 - Integration
 - Connected Intelligence Cloud

5. Click **Generate** to create the access token that can be copied from the subsequent window.

Setting Environment Variables

Flogo Extension for VS Code supports setting the platform-specific environment variables. These variables can be set for app executables, Unit Test Play mode, and all Unit Tests. To set these environment variables, perform the following steps:

Procedure

1. Go to **Settings**.
2. Navigate to **Features > Terminal configuration**.
3. Select **Edit** in settings.json for the current operating system.
4. It opens the settings.json file in the text editor.
5. Add the environment variables in the "<Variable Name>": "<Variable Value>" format.

For example, if you want to set the FLOGO_LOG_LEVEL for Windows, perform the following steps:

- a. Go to the terminal.integrated.env.windows entry.
- b. Add an entry "FLOGO_LOG_LEVEL": "DEBUG" to get the following output:

```
"terminal.integrated.env.windows": {
  "FLOGO_LOG_LEVEL": "DEBUG"
}
```

6. Save the changes to the settings.json file.

Environment variables apply when running app executables and Flogo Unit Tests in Play Mode or Test Mode. Debug logs are visible in the console after setting the environment variable.



Note: These environment variable changes are applicable to all the Flogo Apps. It is a best practice to revisit the settings.json file before running the apps and unit tests.

Environment Variables

This topic lists the environment variables associated with the Flogo Extension for VS Code runtime environment.

Environment Variable Name	Default Values	Description
FLOGO_RUNNER_QUEUE_SIZE	50	The maximum number of events from all triggers that can be queued by the app engine.
FLOGO_RUNNER_WORKERS	5	The maximum number of concurrent events that can be executed by the app engine from the queue.
FLOGO_LOG_LEVEL	INFO	Used to set a log level for the Flogo app. Supported values are: <ul style="list-style-type: none"> • INFO • DEBUG • WARN • ERROR

		This variable is supported for Remote Apps managed with the TIBCO Cloud Integration Hybrid Agent.
FLOGO_ LOGACTIVITY_ LOG_LEVEL	INFO	<p>Used to control logging in the Log activity. Values supported, in order of precedence, are:</p> <ul style="list-style-type: none"> • DEBUG • INFO • WARN • ERROR <p>For example:</p> <ul style="list-style-type: none"> • If the Log level is set to WARN, WARN and ERROR logs are filtered and displayed. • If the Log Level is set to DEBUG, then DEBUG, INFO, WARN, and ERROR logs are displayed.
FLOGO_ MAPPING_SKIP_ MISSING	FALSE	<p>When mapping objects if one or more elements are missing in either the source or target object, the mapper generates an error when FLOGO_MAPPING_SKIP_MISSING is set to FALSE.</p> <p>Set this environment variable to true if you want to return a null instead of receiving an error.</p>
FLOGO_APP_ MEM_ALERT_ THRESHOLD	70	The threshold for memory utilization of the app. When the memory utilization by an app running in a container exceeds the threshold that you have specified, you get a warning log
FLOGO_APP_ CPU_ALERT_ THRESHOLD	70	The threshold for CPU utilization of the app. When the CPU utilization by an app running in a container exceeds the threshold that you have specified, you get a warning log.
FLOGO_APP_ DELAYED_STOP_ INTERVAL	10 seconds	When you scale down an instance, all inflight jobs are lost because the engine is stopped immediately. To avoid losing the jobs, delay the stopping of the engine by setting the

FLOGO_APP_DELAYED_STOP_INTERVAL variable to a value less than 60 seconds. Here, when you scale down the instance, if there are no inflight jobs running, then the engine stops immediately without any delay. In case of inflight jobs:

If there are any inflight jobs running, then the engine stops immediately after the inflight job is completed.

If the inflight job is not completed within a specified time interval, then the job gets killed and the engine stops.

GOGC	100	Sets the initial garbage collection target percentage. Setting it to a higher value delays the start of a garbage collection cycle until the live heap has grown to the specified percentage of the previous size. Setting it to a lower value causes the garbage collector to be triggered more often as less new data can be allocated to the heap before triggering a collection.
------	-----	--

This section lists the user-defined environment variables that are associated with the Flogo Extension for VS Code runtime environment.

Environment Variable	Default Values	Description
FLOGO_MAPPING_OMIT_NULLS	TRUE	Used to omit all the keys in the activity input evaluating to null.
FLOGO_FLOW_CONTROL_EVENTS	N/A	If you set FLOGO_FLOW_CONTROL_EVENTS as true, the Flow limit functionality is enabled, whenever the incoming requests to trigger reach FLOGO_RUNNER_QUEUE_SIZE limit the trigger is paused. When all the requests currently under processing are finished, the trigger is resumed again. All the connectors supporting the flow limit functionality are mentioned in their respective user guides.
FLOGO_HTTP_SERVICE_PORT	N/A	Used to set the port number to enable runtime HTTP service, which provides APIs for health check and statistics.

FLOGO_LOG_FORMAT	TEXT	Used to switch the logging format between text and JSON. For example, to use the JSON format, set FLOGO_LOG_FORMAT=JSON ./<app-name>
FLOGO_MAX_STEP_COUNT	N/A	The application stops processing requests after the FLOGO_MAX_STEP_COUNT limit is reached. The default limit is set to 10 million even when you do not add this variable.
FLOGO_EXPOSE_SWAGGER_EP	FALSE	If you set this property to TRUE, the Swagger endpoint is exposed. The Swagger of the Rest trigger app can be accessed by hitting the Swagger endpoint at http://<service-url>/api/v2/swagger.json.
FLOGO_SWAGGER_EP	NA	To customize the URI for the Swagger endpoint, set this environment variables to the desired endpoint. For example, FLOGO_SWAGGER_EP=/custom/swagger/endpoint This makes the Swagger endpoint available at /custom/swagger/endpoint instead of the default /api/v2/swagger.json location.
FLOGO_APPS_PROPS_CONSUL	NA	Specifies the Consul server address. For example <server_address>:<http://127.0.0.1:8500>
FLOGO_APP_PROP_RECONFIGURE	FALSE	Specifies that app properties can be updated or overridden at runtime.
FLOGO_APP_PROP_SNAPSHOTS	FALSE	Used along with FLOGO_APP_PROP_RECONFIGURE. If this variable is set to true, the app keeps the app properties the same for the entire flow. Even if you change the app properties while the flow is running, the new properties are effective only for new flows and not existing ones.
FLOGO_HTTP_SERVICE_PORT	7777	Specifies the service port. For apps running in TCI, you do not need to specify the port.
FLOGO_OTEL_	INTERNAL	Used to specify the type of span to be used in

SPAN_KIND		<p>OpenTelemetry. The supported values are INTERNAL, SERVER, CLIENT, PRODUCER, and CONSUMER.</p> <p>Note: If no value or an invalid value is provided, the default value will be set to INTERNAL.</p>
FLOGO_LOG_CONSOLE_STREAM	stderr	Used to specify the logging output stream for Flogo engine and app logs. The supported values are stdout and stderr.
FLOGO_LOG_CTX	FALSE	<p>Used to enable context logging for the application. When set to TRUE, context, such as application name, version, tracing details, and flow details, is added to the engine and connectors logs.</p> <p>Note: This context is always logged in JSON format.</p>
FLOGO_LOG_CTX_FIELDS	None	<p>When context logging is enabled, set custom context fields in the logging. These additional fields are added to the logging context.</p> <p>For example, FLOGO_LOG_CTX_FIELDS="service.name=Foo,service.version=1.0.0,service.environment=dev"</p>
FLOGO_APP_METRICS_PROMETHEUS_LABELS	None	<p>Used to set custom labels in the Flogo Prometheus metrics.</p> <p>For example, FLOGO_APP_METRICS_PROMETHEUS_LABELS="environment=dev,job=api-server"</p>
FLOGO_ENV	None	Used to set the name of the deployment environment, such as dev, staging, and production. When enabled, by default, the deployment.environment field is set in the logging context (if enabled) and in Flogo OpenTelemetry traces and metrics.

Building and Running your App Locally

This section has instructions about building and running your apps locally.

Building the App Locally


After you have created an app, you can build it anytime. When you build the app, the app binary gets created in the **bin** folder under your Workspace.

Before you begin

- The app for which an app executable needs to be created must have a trigger and a flow in it. If the app does not have a trigger and flow, the app executable is not created.
- You have associated a runtime with your app that you can see under the **Explorer View > Flogo App Workspace > Actions** tab.
- You need go version=go1.23.9 or above, to be installed on your machine. For information about go installation on Windows, Mac, or Linux, see <https://go.dev/doc/install>.

Procedure

1. Select the app that you want to build in **Explorer View**.
2. Navigate to **Explorer View > Flogo App Workspace** and click the dropdown icon.
3. Click the **Build** icon.

 **Note:** Ensure that the build configuration is set according to your operating system and architecture. If not, click the **Configure and Run** icon next to the **Build** icon to configure the correct settings.

4. Click **Cancel** if you want to stop building the application.

Result

When the build is finished, the app binary is created in the bin folder in your workspace. If you have canceled the compilation, it might delete the current executable.

Running the App Locally

Selecting a Runtime for Your App

To select and set a runtime environment for your app:

Procedure

1. Select the app from **Explorer** for which a runtime needs to be selected.
2. Under **FLOGO APP**, select **Select a runtime to see available actions**.
A list of existing runtimes is displayed at the top. If you do not want to use an existing one, you can create a runtime by clicking **Create new runtime**.
3. Select the desired runtime environment from the list at the top.

i Note: Ensure that the selected runtime has **[Current]** as a suffix.

For information on how to create a local runtime environment, see [Creating Local Runtime](#)

Running the App

Procedure

1. Select the app that you want to run in **Explorer View**.
2. Navigate to **Explorer View > Flogo App Workspace** and click the dropdown icon.
3. Click the **Run** icon. When you click the **Configure and Run** icon, you see the following options:
 - **Environment variables:** To add environment variables when running an app. For more information, see [Overriding an App Property at Runtime](#) and [Environment Variables](#).
 - **Configure Run mode:** To choose between Docker App and Executable modes. To run an app in a docker container, see [Generating an Application Docker Image Locally](#). Select Executable if you want to run the application binary locally.
 - **Save and Run:** To save and run the same configuration for all the apps.

- **Run:** To add a particular app with a selected configuration, without saving it for the next run.

4. Click **Cancel** if you want to stop the application compilation.

For a list of the environment variables, see [Environment Variables](#).

Result

When it runs successfully, you can see the logs in the terminal window of Visual Studio Code. If you have canceled the compilation, it might delete the current executable.

Testing the App on the OS terminal

On the Macintosh and Linux

Procedure

1. Open a terminal.
2. Run:
`chmod +x <app-file-name>`



Note: Make sure that you run the app binary file and not the **.FLOGO** file.

3. Run:

`./<app-file-name>`

On Microsoft Windows

At the command prompt, run:

```
<app-file-name>.exe
```

Exporting App JSON from an Executable

To export the app JSON from an app executable, run the following command:

```
<app_exec> export -o <app-name>.flogo app
```

Where:

- `<app_exec>` is the executable of your Flogo application.
- `<app-name>` is the name of the exported file.

This command extracts the application's configuration and workflow details and saves it as a JSON file.

Building Flogo App Executable and Docker Image Using Flogo - App Build CLI

TIBCO Flogo® - App Build Command Line Interface is a command-line utility for building Flogo applications, especially in CI/CD pipelines. It provides a consistent and easy-to-use interface for building and testing Flogo applications.

To install Flogo® - App Build Command Line Interface (CLI), perform the steps mentioned in [Installing Flogo - App Build CLI](#).

For more information on Flogo - App Build CLI commands, see [Building Flogo App Executable and Docker Image Using Flogo - App Build CLI](#).

Editing a Runtime

To edit an existing runtime configuration:

Procedure

1. Select the Flogo icon in **Activity Bar**.
2. In the Runtime Explorer, select the runtime to be edited and click the **Edit runtime** icon.
3. In the **Edit runtime** dialog, edit the details that you want to modify.
4. Click **Save**.

Deleting a Runtime

To delete a runtime environment:

Procedure

1. Select the Flogo icon in **Activity Bar**.
2. In the Runtime Explorer, select the runtime environment to be deleted.
3. Right-click and select **Delete**.

Generating an Application Docker Image Locally

Before you begin

Install Docker and ensure it is running properly.

To generate an application Docker image, perform the following steps:

1. Choose an app for which you want to build an application Docker image and run as an application Docker container.
2. Select **Runtime: local** and choose the **Configure and Run Action** mode.



Note: The application Docker image generation is only supported for local runtime.

3. In the dropdown list, choose **Configure Run mode**.



Note: To override the app property at runtime, see Overriding Using Configure and Run Locally in [Overriding an App Property at Runtime](#).

4. To generate an application Docker image, select **Run as Docker App**.

i Note: Once you click on **Run as Docker App**, a container instance of the app Docker image is started. The container instance is terminated after you close the application.

5. Rename the application Docker image in the next prompt. By default, it takes the name of the current app and version.
6. Click **Run** to run it once or choose **Save and Run** to save and then generate the application Docker image.

i Note: By default, the Dockerfile used to generate the application Docker image uses amazonlinux as the base image. You can use alpine:3.14 and other compatible base images as well.


i Note: The application Docker images use the Linux binary. Therefore, you cannot generate the application Docker image of apps that use native drivers, such as EMS and OracleDB, on Mac or Windows.

Deployment and Configuration

After you have created and validated your app, you can build an app executable to deploy and run it.

Deploying Flogo Apps from Visual Studio Code

Procedure

1. Open the **Explorer** view in Visual Studio Code.
2. Select the Flogo App to be deployed.
3. Navigate to **FLOGO APPS**, click  beside **Actions** to select any runtime, which has **TIBCO Platform** as its type.
4. Click the **Deploy** option under **Actions** to deploy the app to Platform.



Note: Deploy option only appears for a runtime, which has the type as **TIBCO Platform**.

Result

The Flogo App is deployed on TIBCO Control Plane. You can view the progress on the bottom right of the screen and also navigate to the newly deployed app.

View Flogo App List in Runtime Explorer

For a connected runtime of type TCI or TIBCO Platform, Visual Studio Code displays a list of the Flogo apps and available actions.

The runtime explorer for TIBCO Platform lists all the available apps in the runtime. When you right-click a Flogo app, you can see the following action icons:

For TCI and TIBCO Platform

- ★ Add to favorites
- 🌐 Open in browser

TIBCO Platform Only

- ▶ Start an app
- □ Stop an app

TCI Only

- {} Export and open JSON in Visual Studio Code
- 📄 Export binary
- ▶↓ Export and run binary

Developing for Lambda

AWS Lambda is a serverless compute service provided by Amazon Web Services (AWS). Lambda functions automatically run pieces of code in response to specific events while also managing the resources that the code requires to run. Refer to the AWS documentation for more details on AWS Lambda.

Creating a Connection with the AWS Connector

You must create AWS connections before you use the Lambda trigger or Activity in a flow.

i Note: AWS Lambda is supported on the Linux platform only.

To create an AWS connection:

Procedure

1. In TIBCO Flogo® Extension for Visual Studio Code, click **Connections** to open its page.
2. Click the **AWS Connector** card.

3. Enter the connection details. Refer to the section [AWS Connection Details](#) for details on the connection parameters.

Field	Description
Name	A meaningful string identifying the AWS connection you are creating. The dropdown menu from which you select a connection when creating a Lambda Activity displays this string.
Description	A brief description of this connection
Region	Select a geographic area where your resources are located
Access key ID	Access ID to your AWS account
Secret Access key	Secret access key to your AWS account
Use Assume Role	<p>An AWS Assume role that allows you to assume a role from another AWS account. By default, it is set to False indicating that you cannot assume a role from another AWS account. When set to True, provide the following:</p> <p>Role ARN - Amazon Resource Name of the role to be assumed</p> <p>Role Session Name - Any string used to identify the assumed role session</p> <p>External ID - A unique identifier that might be required when you assume a role in another account</p> <p>Expiration Duration - The duration in seconds of the role session. The value can range from 900 seconds (15 minutes) to the maximum session duration setting that you specify for the role</p> <p>Refer to the AWS documentation for more details on these fields.</p>

4. Click **Save**.

Your connection gets created and is available for you to select in the dropdown menu when adding a **Lambda** Activity or trigger.

AWS Connection Details

To establish the connection to the AWS connector, you must specify the following configurations in the AWS Connector dialog.

The **AWS Connector** dialog contains the following fields:

Field	Description
Name	Specify a unique name for the connection that you are creating. This is displayed in the Connection Name dropdown list for all the AWS activities.
Description	A short description of the connection.
Custom Endpoint	(Optional) To enable the AWS connection to an AWS or AWS compatible service running at the URL specified in the Endpoint field, set this field to True .
Endpoint	<p>This field is available only when Custom Endpoint is set to True.</p> <p>Enter the service endpoint URL in the following format: <code><protocol>://<host>:<port></code>. For example, you can configure a MinIO cloud storage server endpoint.</p>
Region	Region for the Amazon connection.
Access key ID	Access key ID of the AWS account (from the Security Credentials field of IAM Management Console). For details, see the AWS documentation.
Secret access key	Enter the secret access key. This is the access key ID that is associated with your AWS account. For details, see the AWS documentation.
Session token	(Optional) Enter a session token if you are using temporary security credentials. Temporary credentials expire after a specified interval. For more information, see the AWS documentation.
Use Assume Role	<p>This enables you to assume a role from another AWS account. By default, it is set to False (indicating that you cannot assume a role from another AWS account).</p> <p>When set to True, provide the following information:</p>

Field	Description
	<ul style="list-style-type: none">• Role ARN - Amazon Resource Name of the role to be assumed• Role Session Name - Any string used to identify the assumed role session• External ID - A unique identifier that might be required when you assume a role in another account• Expiration Duration - The duration in seconds of the role session. The value can range from 900 seconds (15 minutes) to the maximum session duration setting that you specify for the role. <p>For details, see the AWS documentation.</p>

Creating a Flow with Receive Lambda Invocation Trigger

The **Receive Lambda Invocation** trigger allows you to create a Flogo flow to create and deploy as a Lambda function on AWS.

Refer to the [Receive Lambda Invocation](#) for details on the trigger.

To create a flow with the **Receive Lambda Invocation** trigger:

Procedure

1. Create a Flogo app.
2. Open the flow page.
3. From the **Triggers** palette, select **Receive Lambda Invocation** and drag it to the triggers area.
4. To configure a trigger, enter the JSON schema or JSON sample data for the operation. This is the schema for the request payload.
5. Click **Continue**.

A flow beginning with the **ReceiveLambdaInvocation** trigger gets created.

6. Click the **ReceiveLambdaInvocation** trigger tile and configure its properties.

Refer to the [Receive Lambda Invocation](#) for details on the trigger.

Deploying a Flow as a Lambda Function on AWS

After you have created the flow, you can deploy it as a Lambda function on AWS.

Before you begin

Note the following points:

- The flow must be configured with the **ReceiveLambdaInvocation** trigger.
- Run and Deploy actions do not deploy Lambda function on the AWS Lambda directly. You must deploy the Lambda function by following the steps in this section.
- Deploying an app having a **ReceiveLambdaInvocation** trigger on the TIBCO Platform does not deploy the app as a Lambda function on AWS Lambda. You must create a Linux/amd64 binary and deploy the binary by following the steps in this section.
- If the execution role name is not provided in the **ReceiveLambdaInvocation** trigger, then the Lambda function is created with the default **AWSLambdaBasicExecutionRole** role. It has the following Amazon CloudWatch permissions:
 - Allow: logs:CreateLogGroup
 - Allow: logs:CreateLogStream
 - Allow: logs:PutLogEvents

If a non-existing execution role is provided, then the user whose AWS credentials are used in the AWS connection should have the following permissions:

- iam:CreateRole
- sts:AssumeRole

To deploy a Flogo app as a Lambda function, user role can have access to following AWSLambda_FullAccess policy which has all the required access.

To deploy a flow as a Lambda function on AWS:

Procedure

1. Build your Flogo app(<myApp>) with the Linux/amd64 target. This is because Lambda deployments are Linux-based and building the binary for Linux/amd64 generates the appropriate artifact to deploy in your AWS Lambda function.
2. Add execution permission to the native Linux/amd64 executable file that you built. Run

```
chmod +x <myApp>-linux_amd64
```

3. You can deploy the <myApp>-linux_amd64 in one of two ways:

- If you are using a Linux environment to design, build, and deploy your apps, you can directly run the following command:

```
<LambdaTriggerBinary> --deploy lambda --aws-access-key <secret_key>
```

For example, myApp-Linux64 --deploy lambda --aws-access-key xxxxxxxxx

i Note: Ensure that the aws-access-key is identical to the one configured in the Flogo UI for the selected AWS connection. This is used for validation with the aws-access-key configured as part of the AWS Connection within the UI and the value provided here does not overwrite the aws-access-key used while designing the app.

This approach of deploying to AWS Lambda works only on Linux platforms.

- .JSON file is passed to Lambda as environment variables.

If you are using a non-Linux environment to design, build, and deploy apps, then perform the following steps:

Procedure

1. Build your Flogo app (<myApp>) with the *Linux/amd64* target.
2. Rename the Flogo executable file to bootstrap. This is mandatory per new provided.al2 and provided.al2023 runtimes.
3. Compress the executable file and rename it to <myFunctionName>.zip.
4. From the AWS Lambda UI, create a Lambda function with Amazon Linux 2023 runtime.
5. Create a role or attach an existing role in the Execution role.
6. Click **Create function**.
7. Go to **Code source**, click **Upload from** and upload the compressed file.

After successful deployment, the Lambda function is created in the AWS Lambda console.

Deploying a Flow as a Lambda Function on AWS using AWS CLI

Procedure

1. Build your Flogo app (<myApp>) with the Linux/amd64 target. This is because Lambda deployments are Linux-based and building the binary for Linux/amd64 generates the appropriate artifact to deploy in your AWS Lambda function.
2. Rename the Flogo executable to bootstrap (this is mandatory as per new provided.al2 and provided.al2023 runtimes).
3. Zip the executable.
4. Zip myFunction.zip bootstrap.
5. Run the AWS CLI:

```
aws lambda create-function --function-name myFunction \  
--runtime provided.al2023 --handler bootstrap \  
--architectures x86_64 \  
--role arn:aws:iam::111122223333:role/lambda-ex \  
--region us-west-2 \  
--zip-file fileb://myFunction.zip
```

Receive Lambda Invocation

Use the **Receive Lambda Invocation** trigger for AWS to start your flow as a Lambda function. The **Receive Lambda Invocation** trigger can be configured only in blank flows. It must not be used with flows that are created with another trigger.

Trigger Settings

**Note:**

An app can contain only one Lambda trigger. An app that has a Lambda trigger cannot contain any other triggers including another Lambda trigger. Also, as the Lambda trigger supports only one handler per trigger, it can have only one flow attached to it. However, the apps that contain a Lambda trigger can contain blank flows that can serve as subflows for the flow attached to the Lambda trigger.

Field	Description
AWS Connection Name	Name of the AWS connector connection you want to use for the flow.
Execution Role Name	(optional) ARN of the role to be used to execute the function on your behalf. The role must be assumable by Lambda and must have CloudWatch logs permission execution role.

Output Settings

Enter the payload schema for the request received on the Lambda function invocation on AWS.

Map to Flow Inputs

This tab allows you to map the trigger output to flow input.

Field	Description
Function	Information about the Lambda function
Context	Envelope information about this invocation
Identity	Identity for the invoking users

Field	Description
ClientApp	Metadata about the calling app
API Gateway Request	Displays the elements in the payload schema that you entered on the Output Settings tab. The elements are displayed in a tree format.

Reply Settings

Enter a schema for the trigger reply in the Reply Data text box.

Map from Flow Outputs

Map the flow output to the trigger reply on this tab.

InvokeLambdaFunction

Use this Activity to invoke a specific Lambda function.

Settings

Field	Description
AWS Connection Name	Select an AWS connection.
ARN (Optional)	Amazon Resource Name. <div> <p>Note:</p> <ul style="list-style-type: none"> You can also specify the ARN on the Input tab. If you specify the ARN on both the tabs, the ARN on the Input tab is used. You must specify the ARN on at least one tab. Otherwise, the Activity returns an error at runtime. </div>

Input Settings

Field	Description
Payload Schema	Enter a JSON request schema for your payload that is used to invoke the Lambda function.

Input

The payload schema that you entered on the **Input Settings** tab is displayed in a tree format on the **Input** tab. Map the elements in the schema using the mapper or enter values for the element by manually typing the value in the mapper.

Field	Description
LambdaARN	Amazon Resource Name.

Note:

- You can also specify the ARN on the **Settings** tab. If you specify the ARN on both the tabs, the ARN on the **Input** tab is used.
- You must specify the ARN on at least one tab. Otherwise, the Activity returns an error at runtime.

Output Settings

Field	Description
Result Schema	The schema for the result that is expected from the Lambda function invokes the request.

Output

The Output tab displays the result schema that you entered on the **Output Settings** tab in a tree format.

TIBCO Documentation and Support Services

For information about this product, you can read the documentation, contact TIBCO Support, and join TIBCO Community.

Product-Specific Documentation

The following documentation for this product is available on the [TIBCO Flogo® Extension for Visual Studio Code Product Documentation](#) page:

- *TIBCO Flogo® Extension for Visual Studio Code Release Notes*
- *TIBCO Flogo® Extension for Visual Studio Code Installation*
- *TIBCO Flogo® Extension for Visual Studio Code User Guide*
- *TIBCO Flogo® Extension for Visual Studio Code Mapper Functions Guide*

Other TIBCO Product Documentation

When working with TIBCO Flogo® Extension for Visual Studio Code, you may find it useful to read the documentation of the following TIBCO products:

- [TIBCO® Control Plane](#)
- [TIBCO Flogo® Enterprise](#)
- [TIBCO Cloud™ Integration](#)

How to Access Related Third-Party Documentation

When working with TIBCO Flogo® Extension for Visual Studio Code, you may find it useful to read the documentation of Visual Studio Code: <https://code.visualstudio.com/>.

How to Join TIBCO Community

TIBCO Community is the official channel for TIBCO customers, partners, and employee subject matter experts to share and access their collective experience. TIBCO Community offers access to Q&A forums, product wikis, and best practices. It also offers access to extensions, adapters, solution accelerators, and tools that extend and enable customers to

gain full value from TIBCO products. In addition, users can submit and vote on feature requests from within the [TIBCO Ideas Portal](#). For a free registration, go to [TIBCO Community](#).

Legal and Third-Party Notices

SOME CLOUD SOFTWARE GROUP, INC. (“CLOUD SG”) SOFTWARE AND CLOUD SERVICES EMBED, BUNDLE, OR OTHERWISE INCLUDE OTHER SOFTWARE, INCLUDING OTHER CLOUD SG SOFTWARE (COLLECTIVELY, “INCLUDED SOFTWARE”). USE OF INCLUDED SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED CLOUD SG SOFTWARE AND/OR CLOUD SERVICES. THE INCLUDED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER CLOUD SG SOFTWARE AND/OR CLOUD SERVICES OR FOR ANY OTHER PURPOSE.

USE OF CLOUD SG SOFTWARE AND CLOUD SERVICES IS SUBJECT TO THE TERMS AND CONDITIONS OF AN AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER AGREEMENT WHICH IS DISPLAYED WHEN ACCESSING, DOWNLOADING, OR INSTALLING THE SOFTWARE OR CLOUD SERVICES (AND WHICH IS DUPLICATED IN THE LICENSE FILE) OR IF THERE IS NO SUCH LICENSE AGREEMENT OR CLICKWRAP END USER AGREEMENT, THE LICENSE(S) LOCATED IN THE “LICENSE” FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENT IS SUBJECT TO THOSE SAME TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

This document is subject to U.S. and international copyright laws and treaties. No part of this document may be reproduced in any form without the written authorization of Cloud Software Group, Inc.

TIBCO, the TIBCO logo, the TIBCO O logo, Flogo Enterprise, TIBCO Cloud Integration, and TIBCO Control Plane are either registered trademarks or trademarks of Cloud Software Group, Inc. in the United States and/or other countries.

All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only. You acknowledge that all rights to these third party marks are the exclusive property of their respective owners. Please refer to Cloud SG’s Third Party Trademark Notices (<https://www.cloud.com/legal>) for more information.

This document includes fonts that are licensed under the SIL Open Font License, Version 1.1, which is available at: <https://scripts.sil.org/OFL>

Copyright (c) Paul D. Hunt, with Reserved Font Name Source Sans Pro and Source Code Pro.

Cloud SG software may be available on multiple operating systems. However, not all operating system platforms for a specific software version are released at the same time. See the “readme” file for the availability of a specific version of Cloud SG software on a specific operating system platform.

THIS DOCUMENT IS PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. CLOUD SG MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S), THE PROGRAM(S), AND/OR THE SERVICES DESCRIBED IN THIS DOCUMENT AT ANY TIME WITHOUT NOTICE.

THE CONTENTS OF THIS DOCUMENT MAY BE MODIFIED AND/OR QUALIFIED, DIRECTLY OR INDIRECTLY, BY OTHER DOCUMENTATION WHICH ACCOMPANIES THIS SOFTWARE, INCLUDING BUT NOT LIMITED TO ANY RELEASE NOTES AND "README" FILES.

This and other products of Cloud SG may be covered by registered patents. For details, please refer to the Virtual Patent Marking document located at <https://www.cloud.com/legal>.

Copyright © 2023-2025. Cloud Software Group, Inc. All Rights Reserved.