API Canvas Graph

Visual API integration for Unity. Drag, drop, and call any REST endpoint. No code, no hassle

The most recent version of this documentation can always be found at https://chocaroonstudios.ca/apicanvas-readme.

Before you begin, there are just a few small things you need to do setup.

Requirements & Setup

- **Unity** (of course): Tested on 6.3 LTS, should work on any LTS version from 2021.3 and up.

Navigate to your package manager and ensure these packages are in your project.

- **Newtonsoft.Json:** Tested on version 3.2.2 and should work on 2.0.0 or higher. To add it, open the package manager window and select `Add package by name` > `com.unity.nuget.newtonsoft-json`.
- **Visual Scripting (optional):** Tested on versions 1.8.0, 1.9.4 & 1.9.11. If you do not plan to use Visual Scripting integration this can be skipped, but if you are going to use it you must generate the custom inspector properties. Navigate to `Edit > Project Settings > Visual Scripting` and select `Generate (Custom Inspector Properties)`.
- **Text Mesh Pro: (optional)** Needed for the sample scenes to work properly. Version 3.2.0-pre.10 or higher is recommended for everything to display properly (older versions have different wrapping options, etc. that might mess up some labels). Lowest version tested on is 3.0.6. If you are not downloading/viewing the sample scenes this is not needed.

How to use the Graph

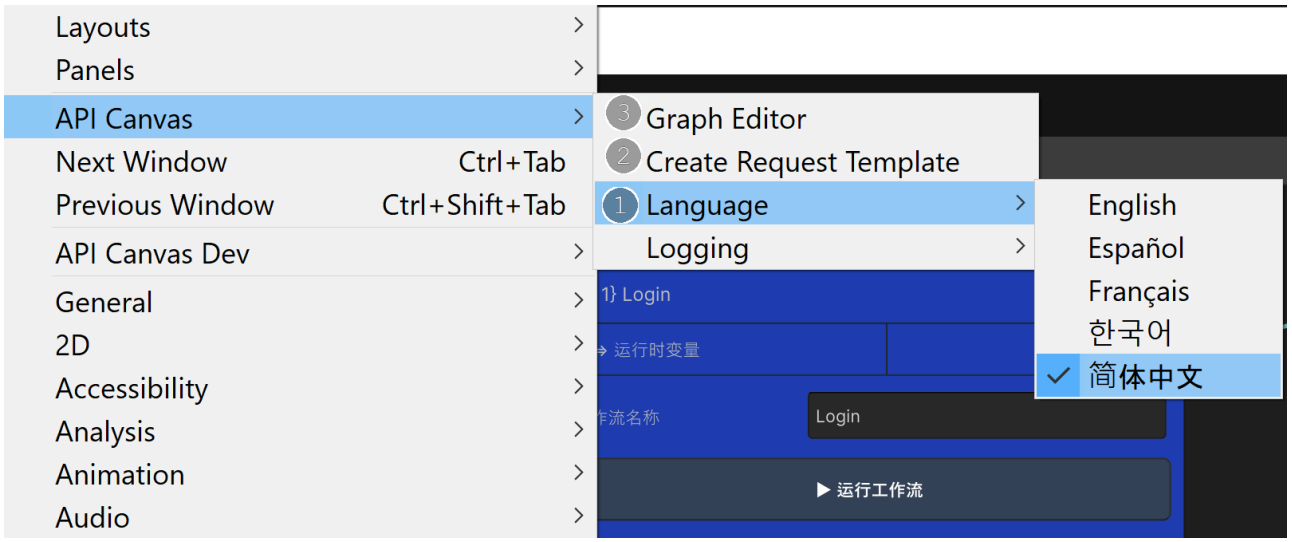
Navigation

- Move Scroll View up/down on nodes: Mouse wheel up & down respectively.
- Zoom in/out: Hold ctrl/cmd & use mouse wheel up & down to explicitly zoom. Using the mouse wheel alone will also work but scrolling scroll views will take precendence over zoom this way.
- Move view: Hold down alt/option and the left mouse button, or hold down the middle mouse wheel button, and move your mouse.
- Zoom out all the way: Press A.
- Zoom to the currently selected node: Press F.
- Return to the center of the graph: Press O.

Menu

- [1]**Language:** Switch the UI language of the tool.
- [2]**Create Request Template:** Create a new Request Template (see Creating Request Templates).

- [3]**Graph Editor:** Open the Graph Editor.



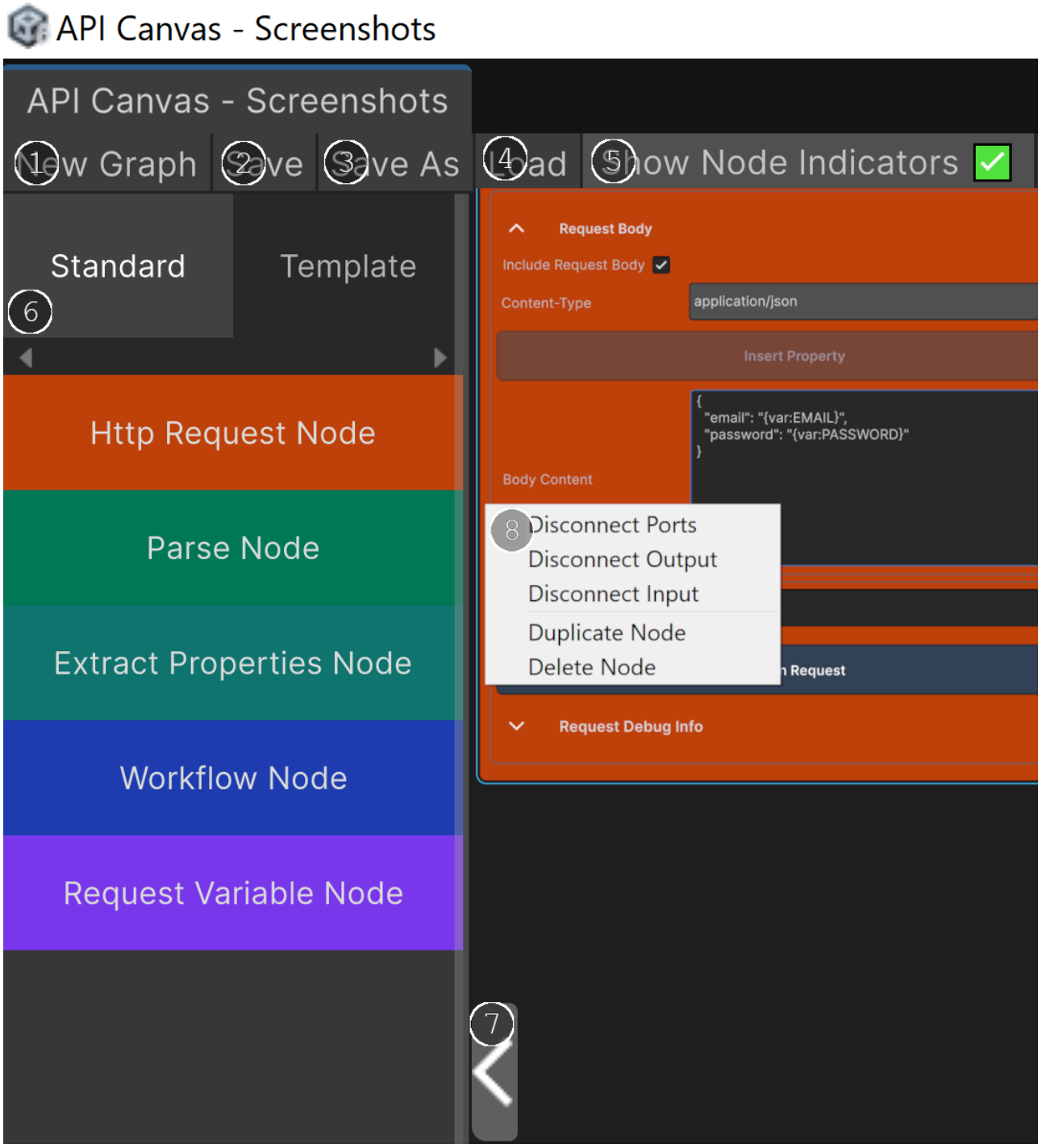
Graph Editor

- [1]**New Graph:** Create a new graph.
- [2]**Save:** Save the current graph.
- [3]**Save as:** Save the current graph as a different file.
- [4]**Load:** Load a graph.
- [5]**Show Node Indicators:** Toggle the visiblility of indicators showing the direction of nodes that are out of view.
- [6]**Side Pane:** Here is where you'll find base nodes that you can drag into the main area to create a node of that type (you can also right click an empty space on the graph and click the appropriate menu button), as well as request templates that will populate an HTTP Request Node with default settings & parameters for calling specific API's (see Creating Request Templates for information on how to create your own).
- [7]**Side Pane Expand/Collapse Button:** As the name implies, expand/collapse the side pane by selecting this.

[8]Right Click Menu (Node)

- **Disconnect Ports:** Disconnect all ports on this node.
- **Disconnect Output:** Disconnect output port.
- **Disconnect Input:** Disconnect input port.
- **Duplicate {Node Type}:** Duplicate the node with all of its data intact (HTTP Request & Extract Properties nodes only).

- **Delete:** Delete this node.



The Nodes

HTTP Request Node

The majority of your logic will be in this node. Let's break down the different sections & properties.

Ports

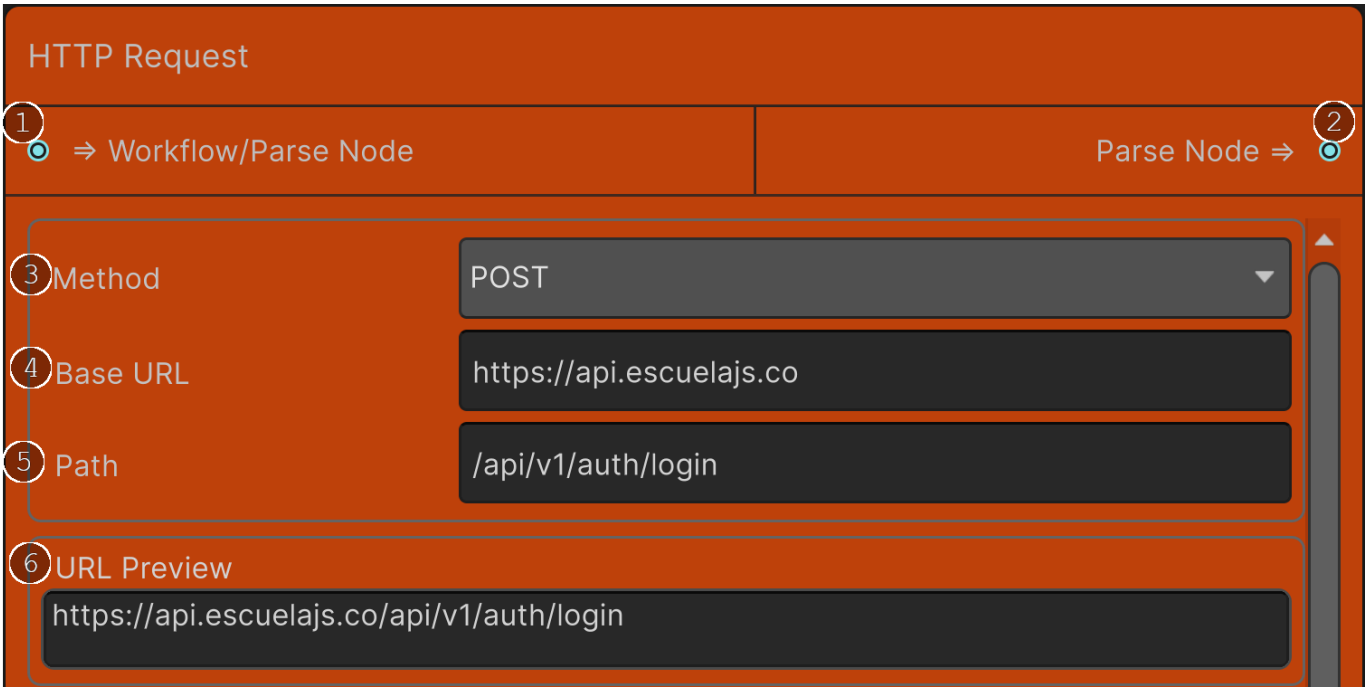
[1]**Input:** Can receive a single connection from a Parse Node or Workflow Node. [2]**Output:** Can send a single connection to a Parse Node.

Base Fields

[3]**Method:** The method used to call the API. These include: **GET:** Retrieves data from a server. **POST:** Sends new data to the server to create a new resource. **PUT:** Replaces or fully updates an existing resource. **PATCH:** Modifies an existing resource. **DELETE:** Removes a specific resource from the server.

[4]**Base URL**L The basic URL of the API endpoint you are calling. For example, `https://12345.playfabapi.com`.

[5]**Path:** The rest of the path of the endpoint you are calling. This combines with the Base URL to form the complete URL (you can also place the full URL in the Base URL field and leave this blank). For example, `/Client/LoginWithEmailAddress` would result in the full endpoint of `https://12345.playfabapi.com/Client/LoginWithEmailAddress` being called. [6]**URL Preview:** See how your final URL will look.



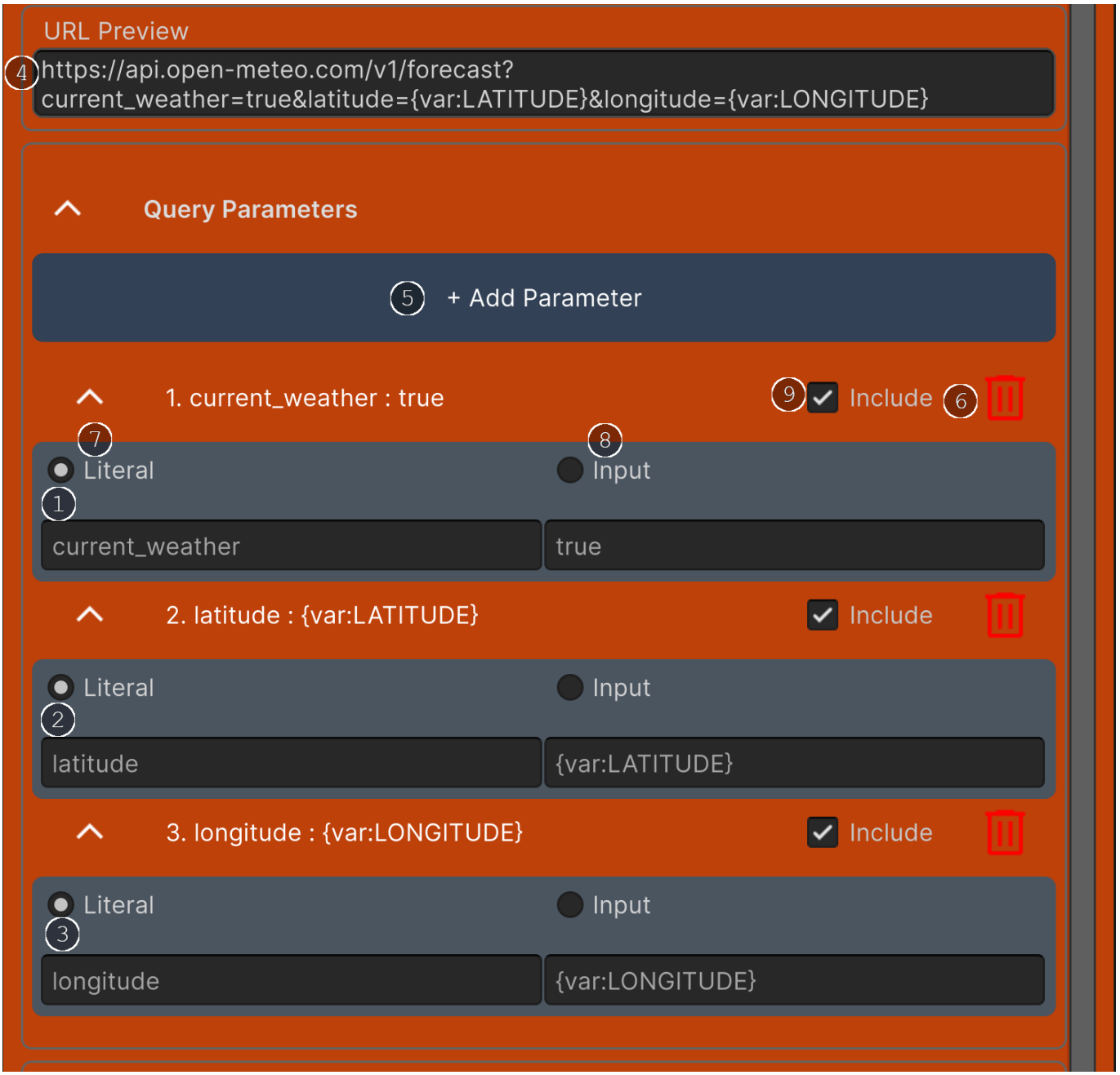
Query Parameters

These append parameters to your URL. You assign a key & value to each. For example, if we wanted to call the weather API to get the current weather around Mount St.Helens, USA it would look like this: [1]**ParamKey1:** `latitude` - **Param1Value:** `46.2` [2]**ParamKey2:** `longitude` - **Param2Value:** `122.18` [3]**ParamKey3:** `current_weather` - **Param3Value**: `true`

This will result in the full URL of [4]`https://api.open-meteo.com/v1/forecast?latitude=46.2&longitude=122.18&current_weather=true`.

You can add parameters using the [5]**+ Add Parameter** button, and remove them by selecting the [6]`Trash` button in the parameter header. Once added, you have to select whether the parameter is a Literal or Input by selecting either [7]**Literal** or [8]**Input**. You can also choose not to include a parameter by unchecking the

[9]**Include** checkbox. See Parameters for info on how to use literal & input parameters.



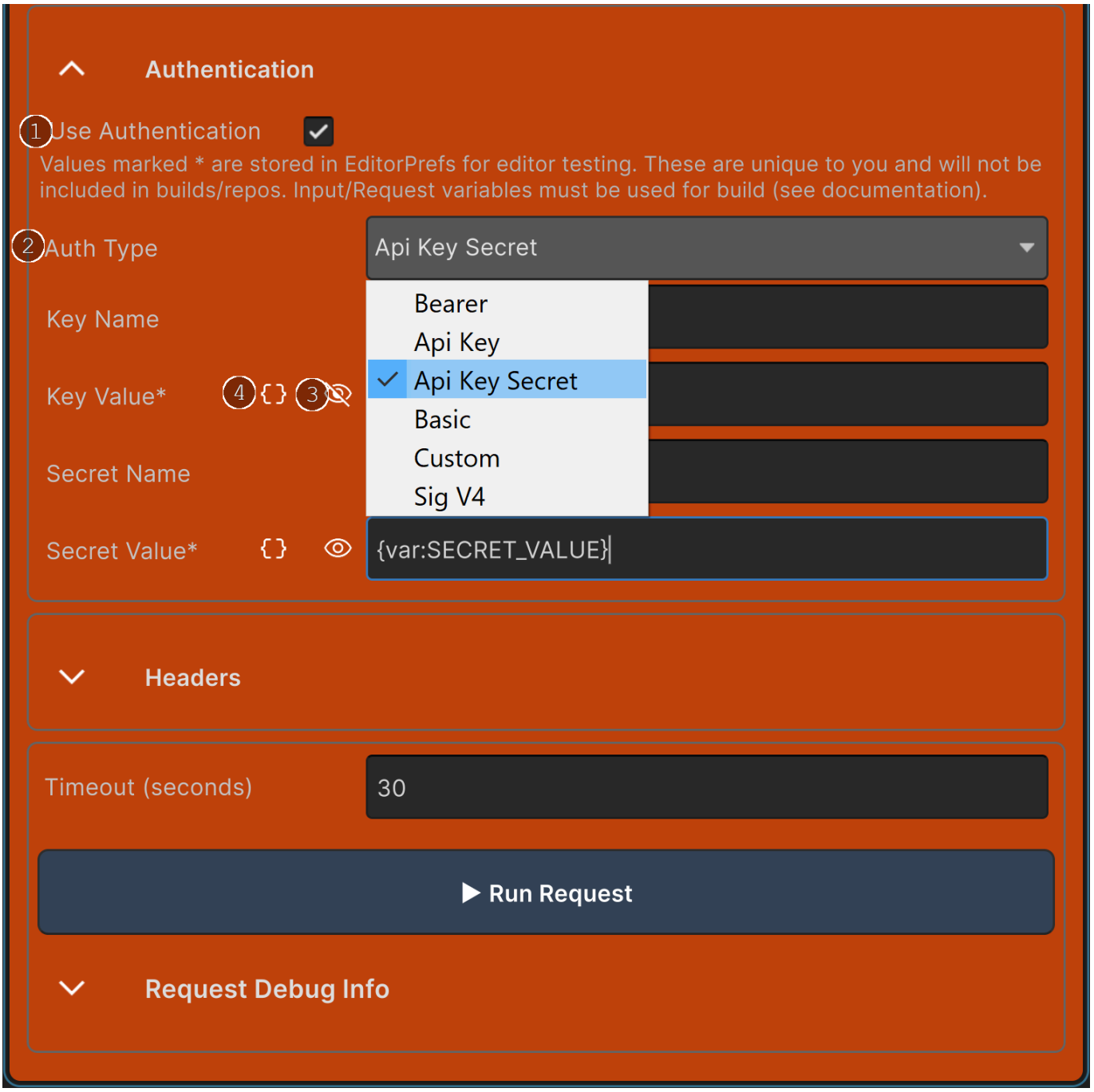
Authentication

As the title implies, this section is used for authentication/signing in to servers.

To use Authentication, ensure the [1]**Use Authentication** checkbox is checked and select the [2]**Auth Type** (see Authentication section for more info).

You can toggle the visibility of sensitive fields on with its corresponding [3]**Visibility Button**.

If there is a connected Parse node with JSON data you can also use the [4]**Insert Property Button** to easily set the field to use the input path of a property (see Property Insertion for more details).

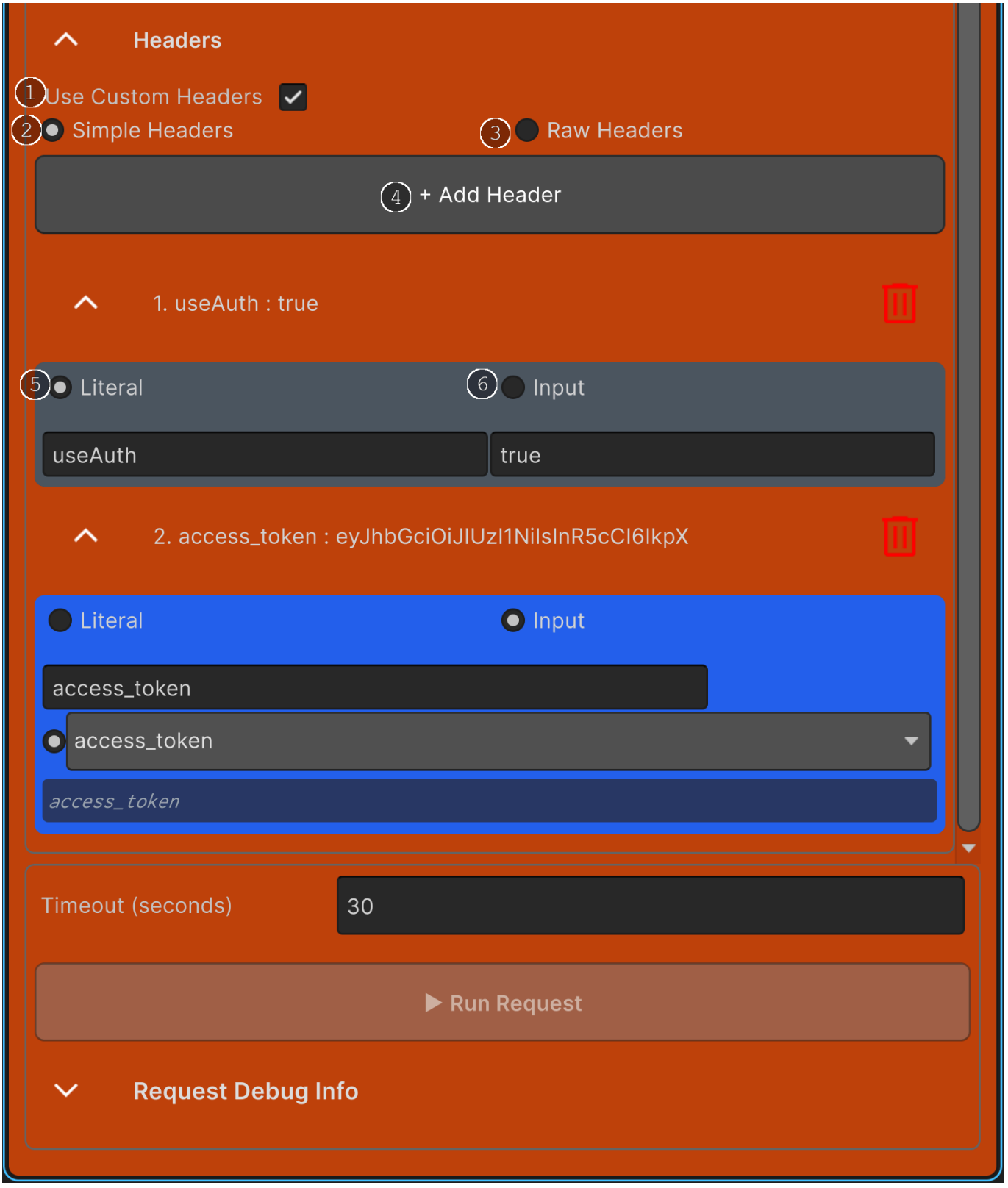


Headers

To add headers, ensure the [1]**Use Custom Headers** checkbox is checked, and select either [2]**Simple Headers** or [3]**Raw Headers**.

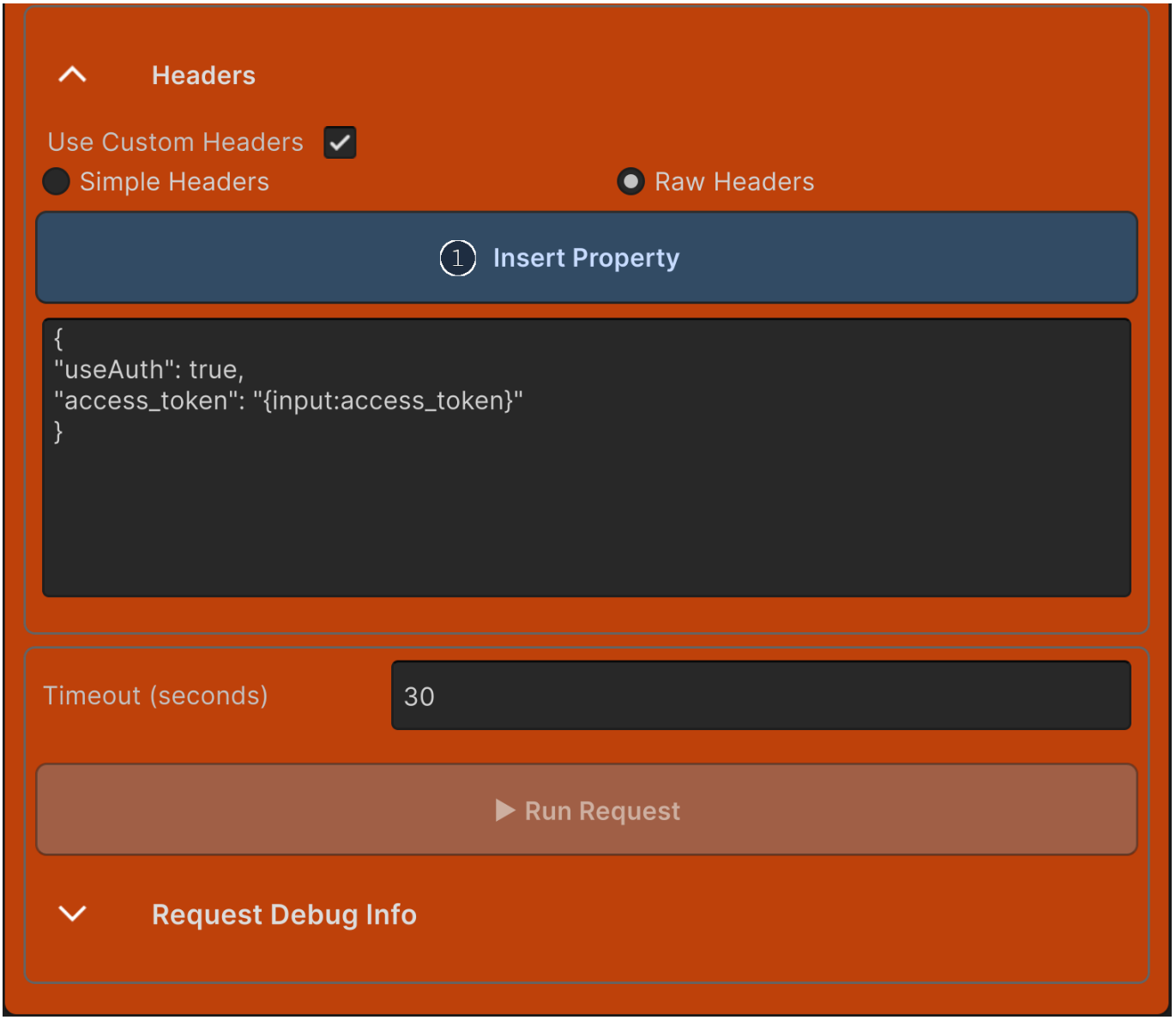
If using simple headers (recommended for most use cases), you can add headers using the [4]**+ Add Header** button. For each header, select either [5]**Literal** or [6]**Input**. See Parameters for info on how to use header

parameters.



If you select raw headers and have a Parse node with JSON content connected to the input port you can select the [1]**Insert Property** button to easily use properties from the connected JSON (see Property Insertion for

more details).

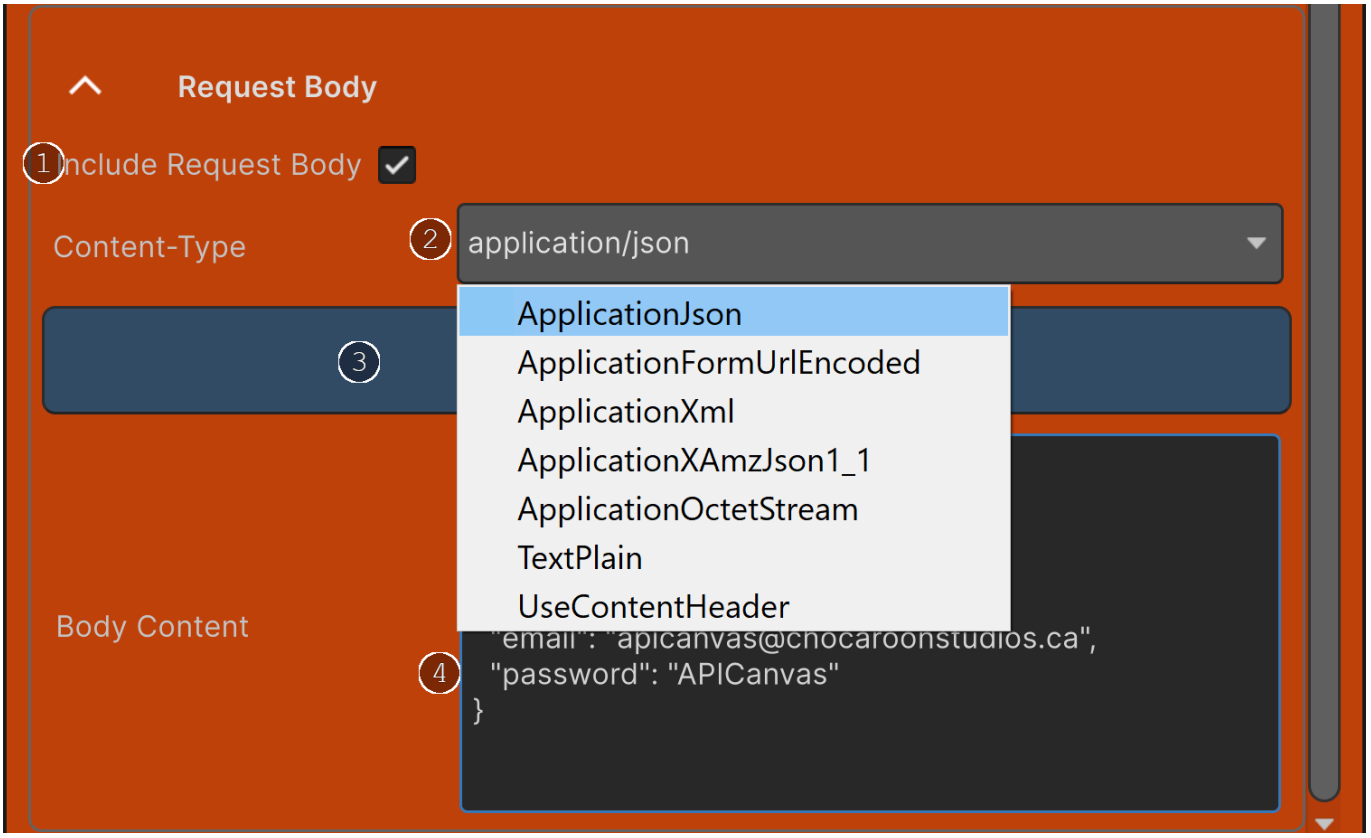


Request Body

To use a request body (this section will only show up when using `POST`, `PUT`, or `PATCH`) ensure the [1]**Include Request Body** checkbox is checked. Then select your [2]**Content-Type** (default is `application/json`). If your content type is not listed in the available types, you can select `UseContentHeader` and pass the content type in via a header (headers with the Content-Type key will be ignored unless `UseContentHeader` is selected here).

If you have a Parse node with JSON content connected to the input port you can select the [3]**Insert Property** button to easily use properties from the connected JSON (see Property Insertion for more details). The [4]**Body**

Content must be entered as raw JSON (a full, no typing JSON builder is planned in future updates).



Documentation

If you are unfamiliar with writing JSON here are a few good resources to get up to speed (you'll likely know enough after reading/watching one of these to use this tool):

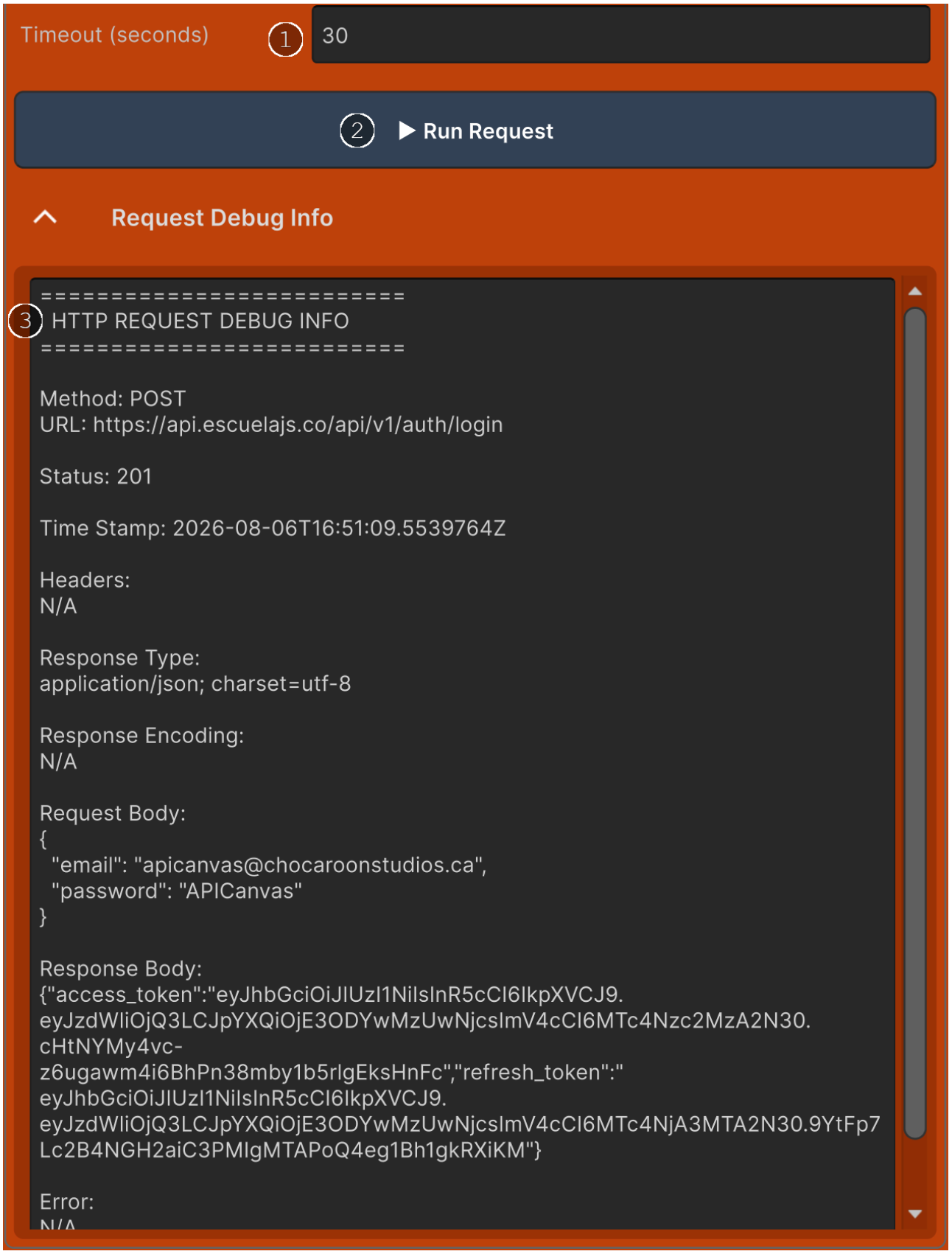
W3Schools ~10 min read - Should give you most of what you need. freeCodeCamp ~15 min read Microsoft Learn ~12 min read MDN ~20 min read

Videos:

Learn JSON in 10 Minutes 11m59s - Should give you all the info you need. If you learn better from video use this, otherwise the W3Schools page should be sufficient. Learn JSON Step-by-Step from Scratch 13m10s - Also a good, quick tutorial.

Footer

- [1]**Timeout:** Field where you can set how long the request will run before timing out.
- [2]**Run Request Button:** Use this to run the request. This will be disabled if a url has not been set and/or a Parse node has not been connected to the output port.
- [3]**Request Debug Info:** After a request is run you can check here to see detailed information (useful for ensuring everything was correctly sent and/or for finding out why a request failed).



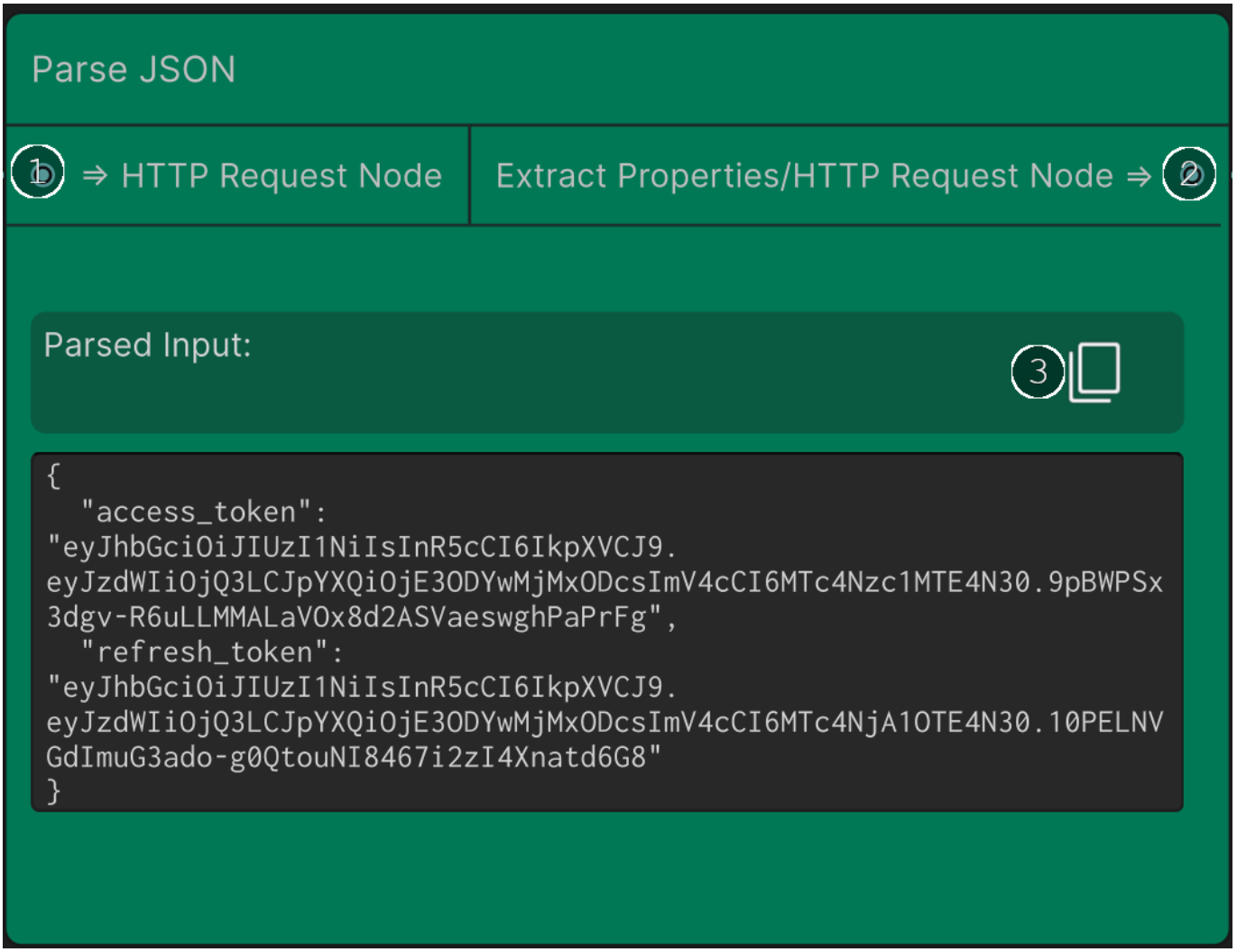
Parse Node

This is what takes the result from a request and formats it into a usable/readable format.

Ports

[1]**Input:** Can receive a single connection from an HTTP Request Node. [2]**Output:** Can send a single connection to an HTTP Request Node (for using Input variables), or an Extract Properties Node. [3]**Copy Button:** Copy the entire response. Sometimes the response will be truncated (long response) and/or broken up into multiple text fields (older Unity version limitations), but using this button will allow you to copy the response in full.

You can chain calls together (for when you need to use input from one call in the next one). For example: `Request > Parse > Request (you can now use the JSON input from the Parse node) > Parse > etc.`



Extract Properties Node

This is the endpoint of your request chain, where you set the names of your properties and their value (dot notation) path.

Port

[1]**Input:** Can receive a single connection from a Parse Node.

Fields

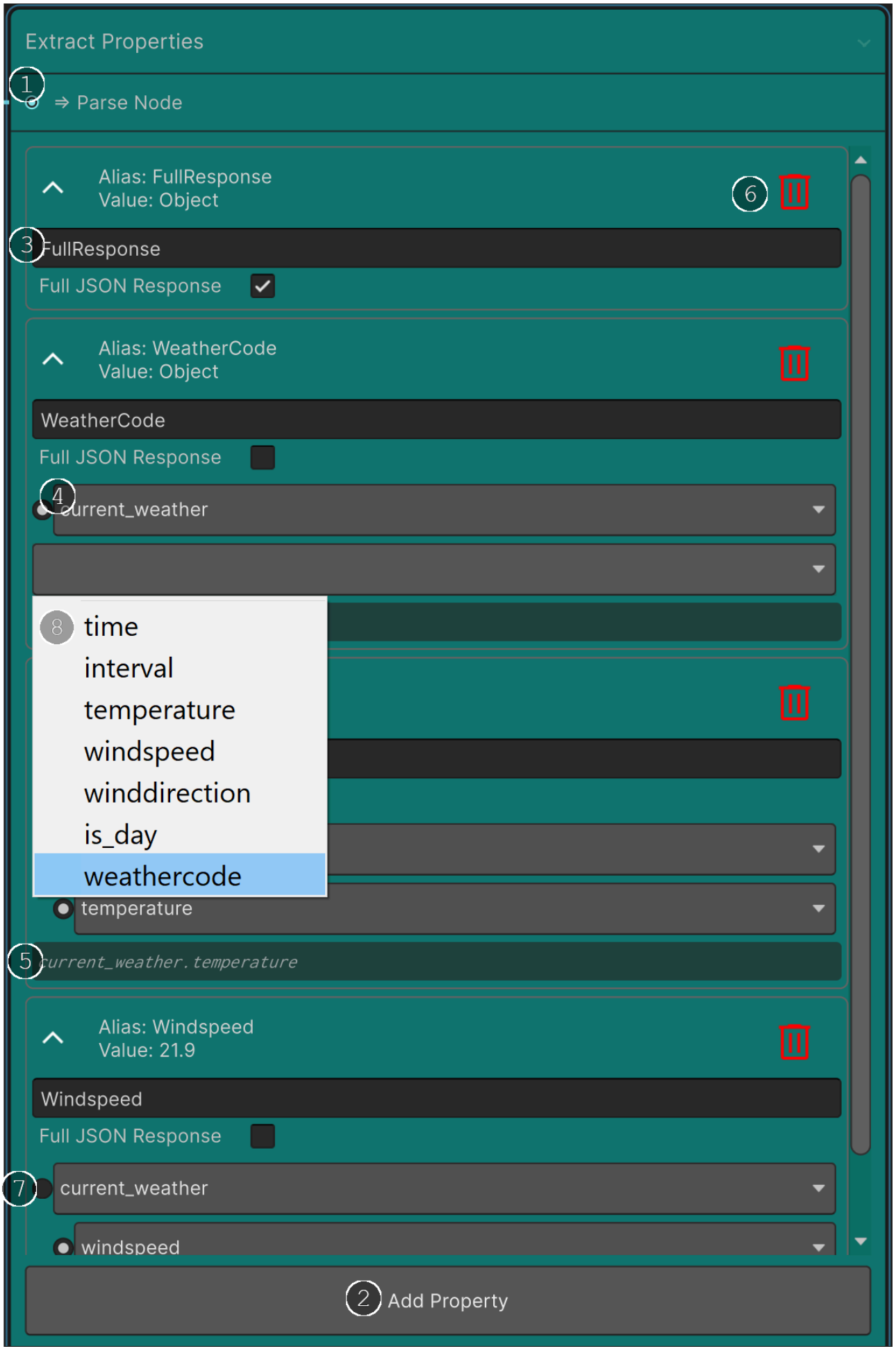
- [2]**Add Property Button:** Use this to add a property.
- [3]**Alias Field:** Enter the alias for the property. This is the name you will use to retrieve this property after running your request(s).

- [4]**Path Selection Field:** Select the path of the property via dropdowns (only visible when a Parse node with valid JSON is connected to the input port). For example, our earlier example with the weather API returned JSON with the object:


```
  "current_weather": {
  "time": "2026-07-07T16:00",
  "interval": 900,
  "temperature": 18.8,
  "windspeed": 9.9,
  "winddirection": 147,
  "is_day": 0,
  "weathercode": 95
}
```


If we wanted to get the `"temperature"` property, we would select `current_weather`, then `temperature` from our dropdowns (or type `current_weather.temperature` in the path field and connect our Parse node).

- [5]**Path Field:** This is the path to the selected property. It is only editable when there is no JSON data available (Parse node not connected or has no data). When a Parse node with data is connected, it attempts to parse the path, so disconnecting & reconnecting will not break your property path unless the path does not exist on the connected JSON. It is recommended to use the dropdowns exclusively.
- [6]**Trash Button:** Use this to delete the property.
- [7]**Radio Button:** If you want to select a different part of the currently selected path as the value source then select its radio button.
- [8]**Property Selection Popup:** This is what you select the path from when opening the path selection field.



Workflow Node

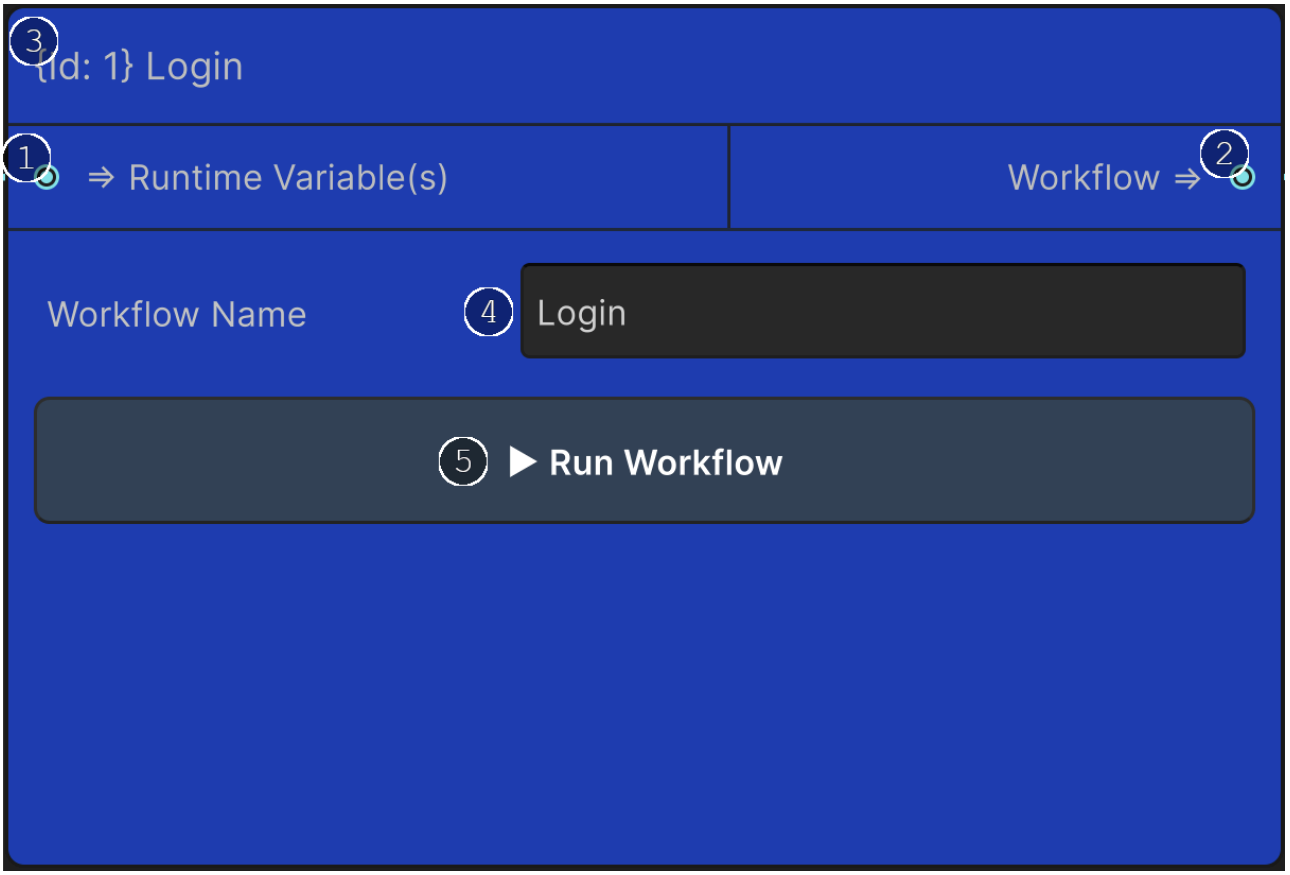
A simple, but important node. This is what you actually reference & call during runtime to run requests & retrieve properties.

Ports

[1]**Input:** (optional) Can receive multiple connections from Request Variable nodes. [2]**Output:** Can send a single connection to an HTTP Request node.

Fields/Properties

- [3]**Id:** An integer (whole number) that is used to reference this workflow during runtime. This is automatically assigned when creating Workflow nodes and cannot be changed.
- [4]**Workflow Name:** Used for you to identify the node (here, in the inspector, and in Visual Scripting Graph). It has no other purpose.
- [5]**Run Workflow Button:** Used to run the workflow to ensure the whole workflow chain is set up & working properly before using it during runtime.



Request Variable Node

This node is for using values in your requests that may not be available at the start of your game/app, and/or for sensitive data that you don't want hard coded. A good example of both would be a user sign in. When they enter their username & password into your UI field and submit it, you would assign the `Username` & `Password` values to your workflow's request variables via code/Visual Scripting. See How to Use Non-static Inputs/Variables for example on how to use these in your request fields.

Ports

[1]**Output:** Can send a single connection to a Workflow node.

Fields

- [2]**Input Key:** This is what will identify your variable (and the string you'll use to set it in code) so give it an appropriate name.

Parameters

This refers to both `Literal` & `Input` parameters used on HTTP Request nodes in their headers & query parameters.

- **Literal:** You can use this to manually enter values that will not change. For example, a Query Parameter with the key "page-count" and the value "10" for queries where you always want the response to give you a page count of 10. This is also where you would use request variables (see How to Use Non-static Inputs/Variables for details). Input variable use also works but there's already a dedicated option for that as you'll see below.
- **Input:** You can use this to indicate properties that should come from attached Parse node JSON (see How to Use Non-static Inputs/Variables for details). For example, if your workflow started with a login request that returned an authentication token, you could set the input path to that token and use it for a subsequent request that requires the token.

How to use Non-static Inputs/Variables

A lot of the time you will need to use property values that are not known/available immediately, or that you do not want to store as static values due to their sensitive nature. That's where you would use these. They can be used in all value fields (including Base URL & Path) of an HTTP Request node with the exception of the `Timeout` field.

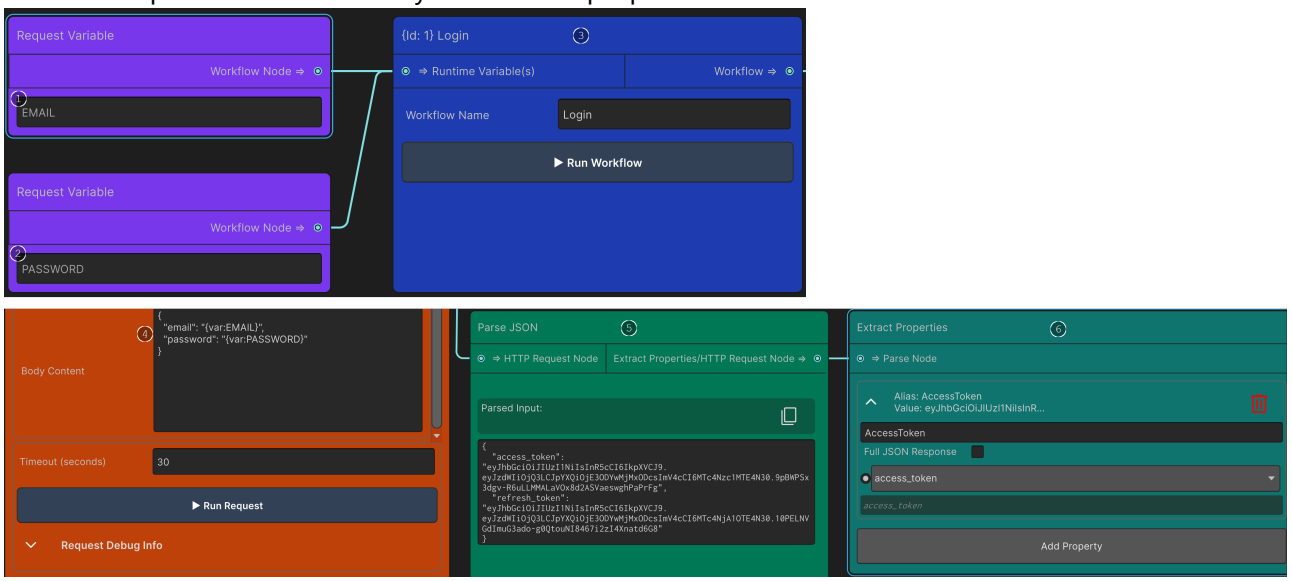
IMPORTANT: *Sensitive fields marked with* ***** *set with static values will not resolve in builds as they are stored in EditorPrefs for safety. Inputs/Variables must be used for these to work.*

Syntax

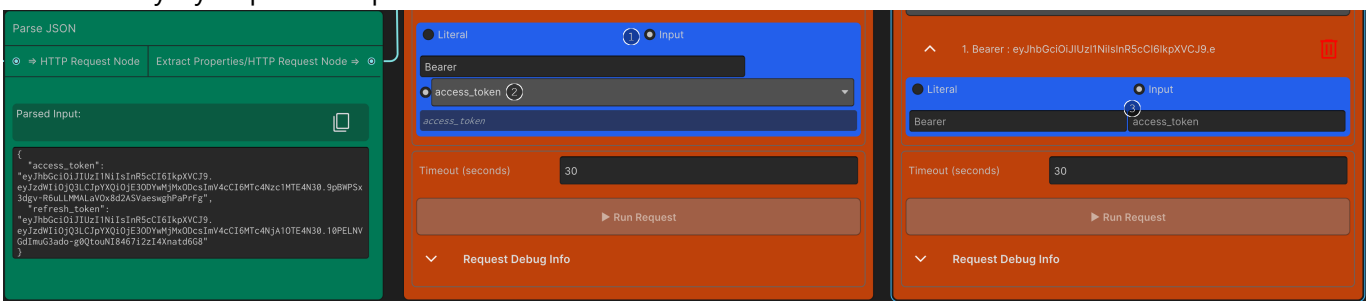
Request Variable: To use a request variable you would enter the following in your (non-input path) value field: `{var:KEYNAME}`, where `KEYNAME` is the key you assigned to the corresponding Request Variable node.

This is best used for values such as user entered data (usernames, etc.) which you would assign to the request variable before running the workflow that uses it. For example:

1. Program Starts
2. User logs in with Email & Password in you UI
3. You assign these values to their respective Request Variable nodes using either Visual Scripting or your own code ([1]`EMAIL` & [2]`PASSWORD`)
4. [3]You run the workflow they are attached to
5. A connected HTTP Request uses [4]`{var:EMAIL}` & `{var:PASSWORD}` to authenticate
6. `{var:EMAIL}` & `{var:PASSWORD}` are replaced with the values the user put in (that you assigned in step 3) and the request is sent
7. [5]Parse node parses the response
8. [6]Extract Properties node extracts your selected properties.



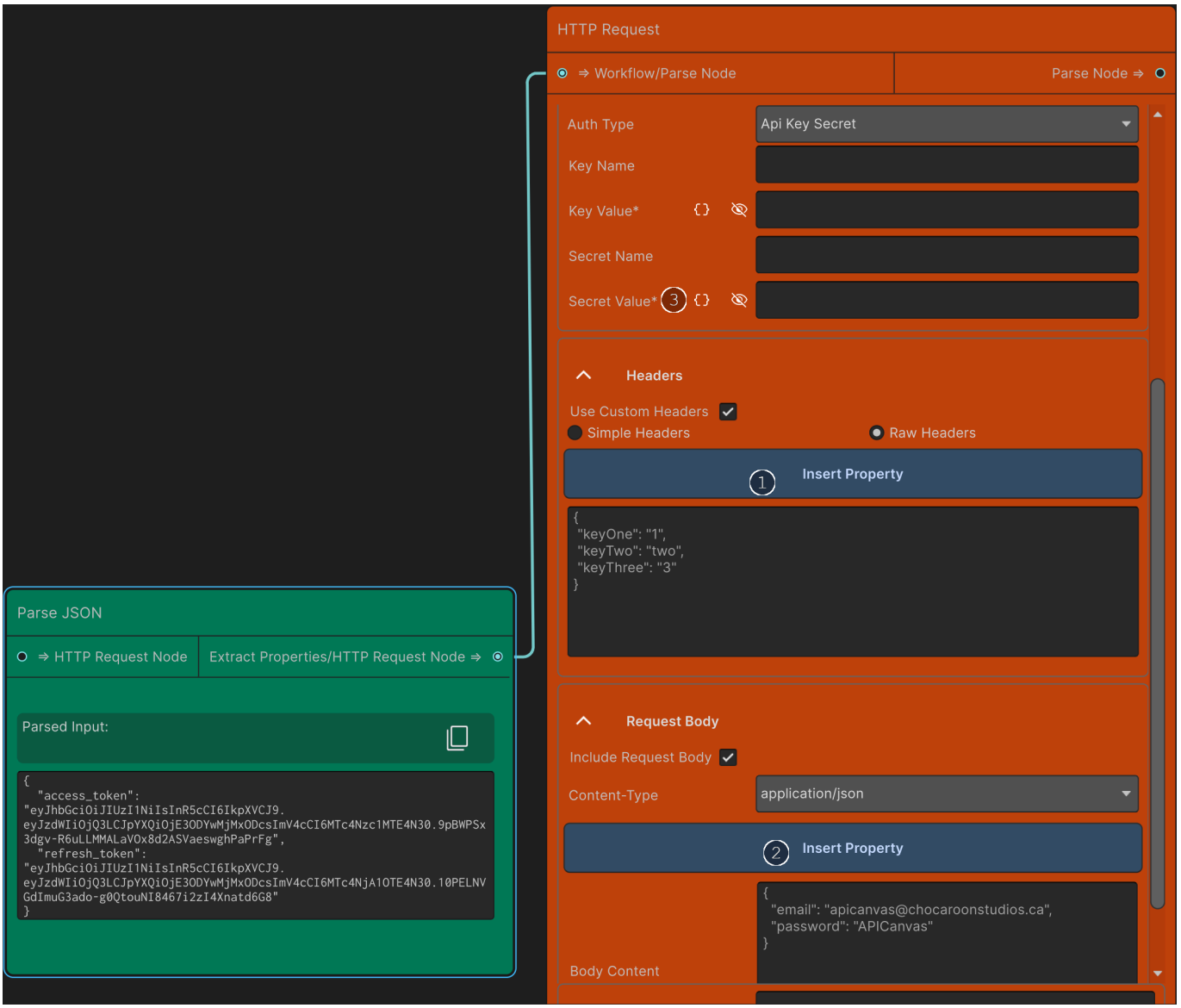
Input Variable: To use an input variable you would simple select [1]`Input` as your type. If a Parse node with JSON data is connected you can simply navigate the [2]dropdowns to set the desired property path (recommended), otherwise, you can [3]manually type in the path & when a Parse node is connected it will automatically try to parse the path.



Note: `{var:key}` & `{input:path}` can be used in all request value fields aside from Timeout, but insert buttons have only been placed where they would commonly be used.

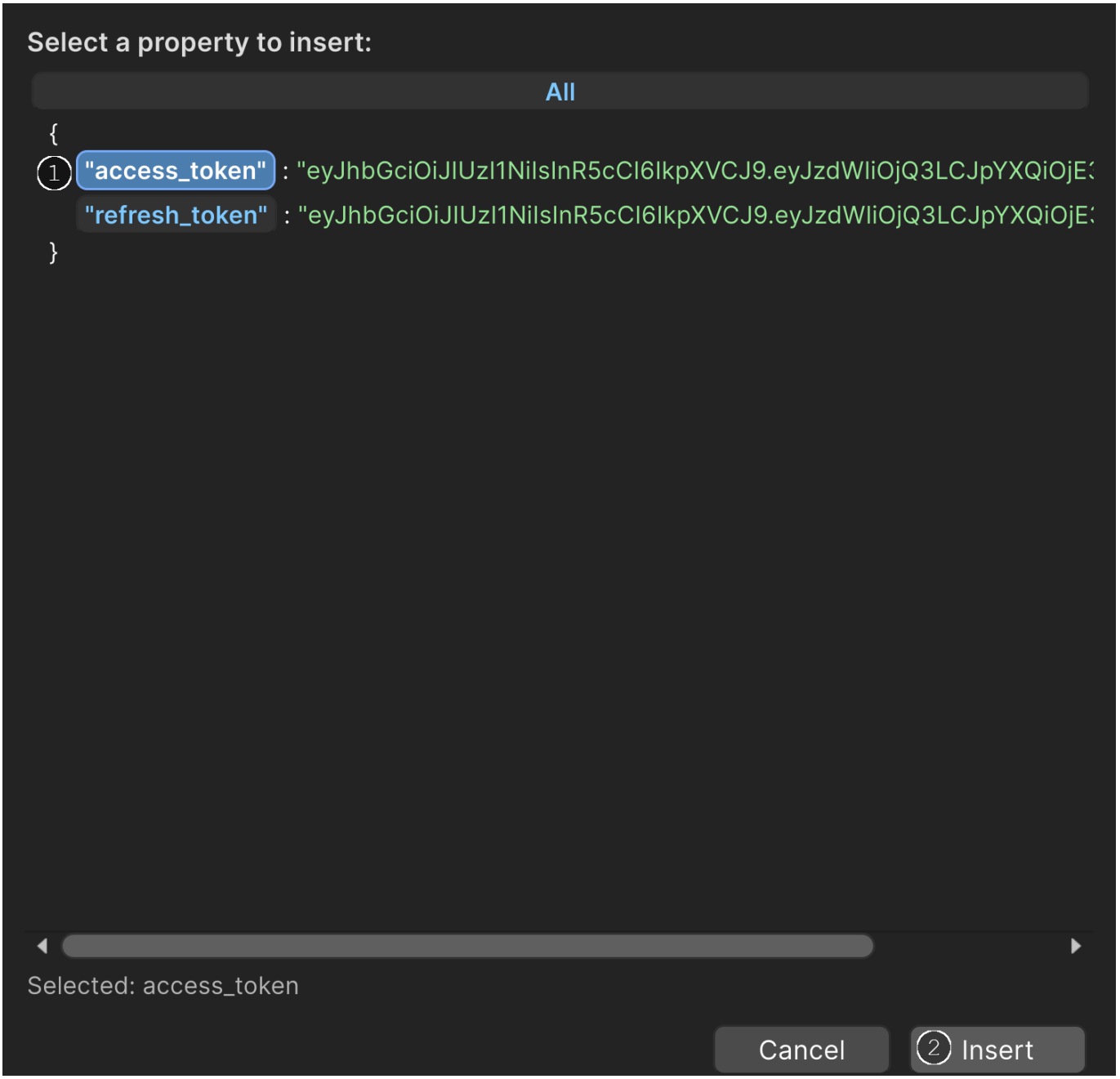
Property Insertion

In [1]**Raw Headers**, [2]**Request Body**, and in some [3]**Authentication fields**, you can choose to auto-insert the input path of a property from the JSON data of a connected Parse node.



Selecting the respective buttons will open the insert property window. From here, [1]select the property you want to insert, and select the [2]**Insert Button** to insert it. For the Raw Headers & Request Body fields it will be inserted at the last place you selected in the text field. For Authentication fields it will replace the entire value

with the property path.



Security/Authentication Handling & Need to Know

Please keep the following in mind when entering sensitive data:

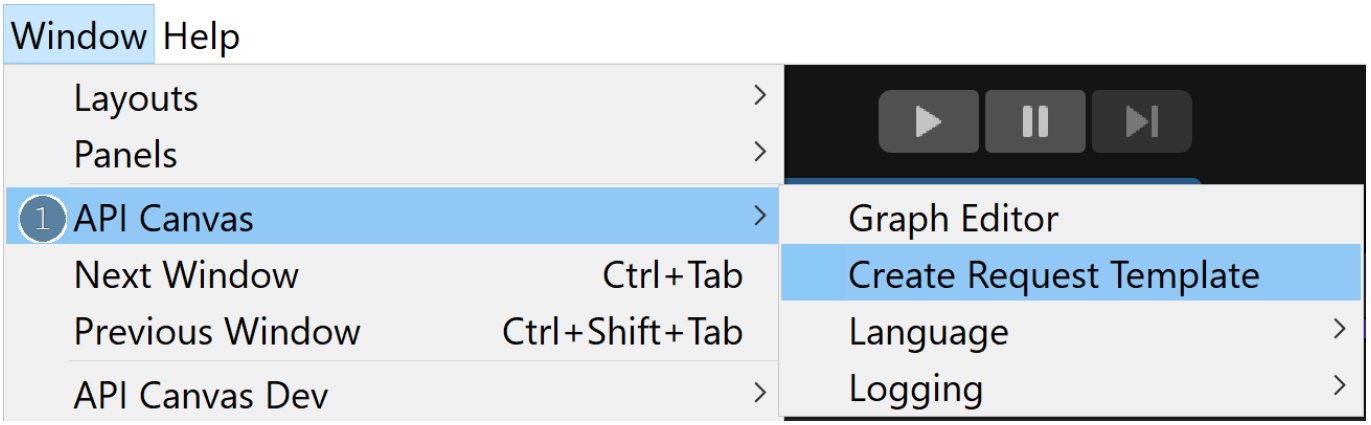
- Any field marked with a * in Authentication means the value is stored in your Editor Preferences and will not be included in your build/git repo/etc. These must be set to input or request variables when using in builds.
- Some platforms require authentication/sensitive data to be passed via request body/query parameters/etc. which are stored in the graph asset. It is strongly advised that you use Request Variables (make sure the value is not exposed by using environment variables, Player/Editor prefs, etc.) & Input Variables before publishing publically, and be careful who you share with if you're testing with hardcoded values.
- When creating templates, ensure you are not using any hardcoded sensitive data in the template if you plan on collaborating or sharing the codebase (or save them in a specific directory that is included in your .gitignore). Precautions are in place to prevent hardcoded values being used in known sensitive

fields (such as various authentication fields), but cannot identify them when used in request bodies/raw headers/etc.

Creating Request Templates

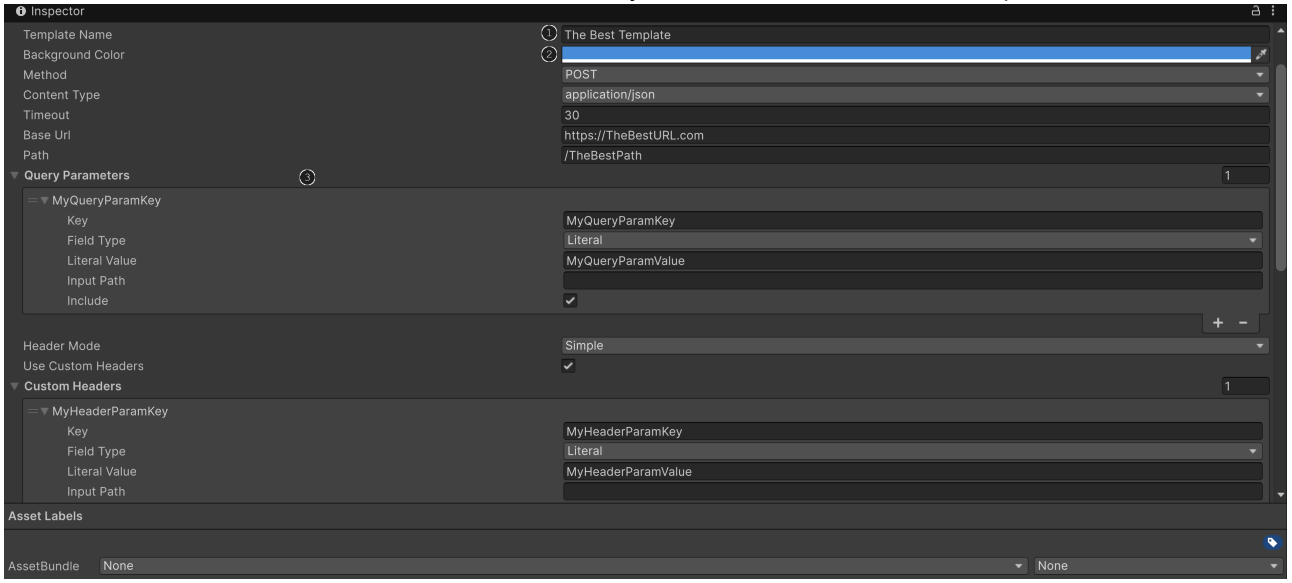
It can be a pain to fill in all the information for a request over and over for the same endpoints. If you find yourself targeting the same API several times, whether in one project, or several (the templates are saved as asset files & can be easily shared between projects), it might make sense to create a template for it. A few are included in this package, but creating your own is very simple.

[1]Simply open the **API Canvas** menu and select **Create Request Template**. Name your asset anything you want and place it in whatever directory you want.

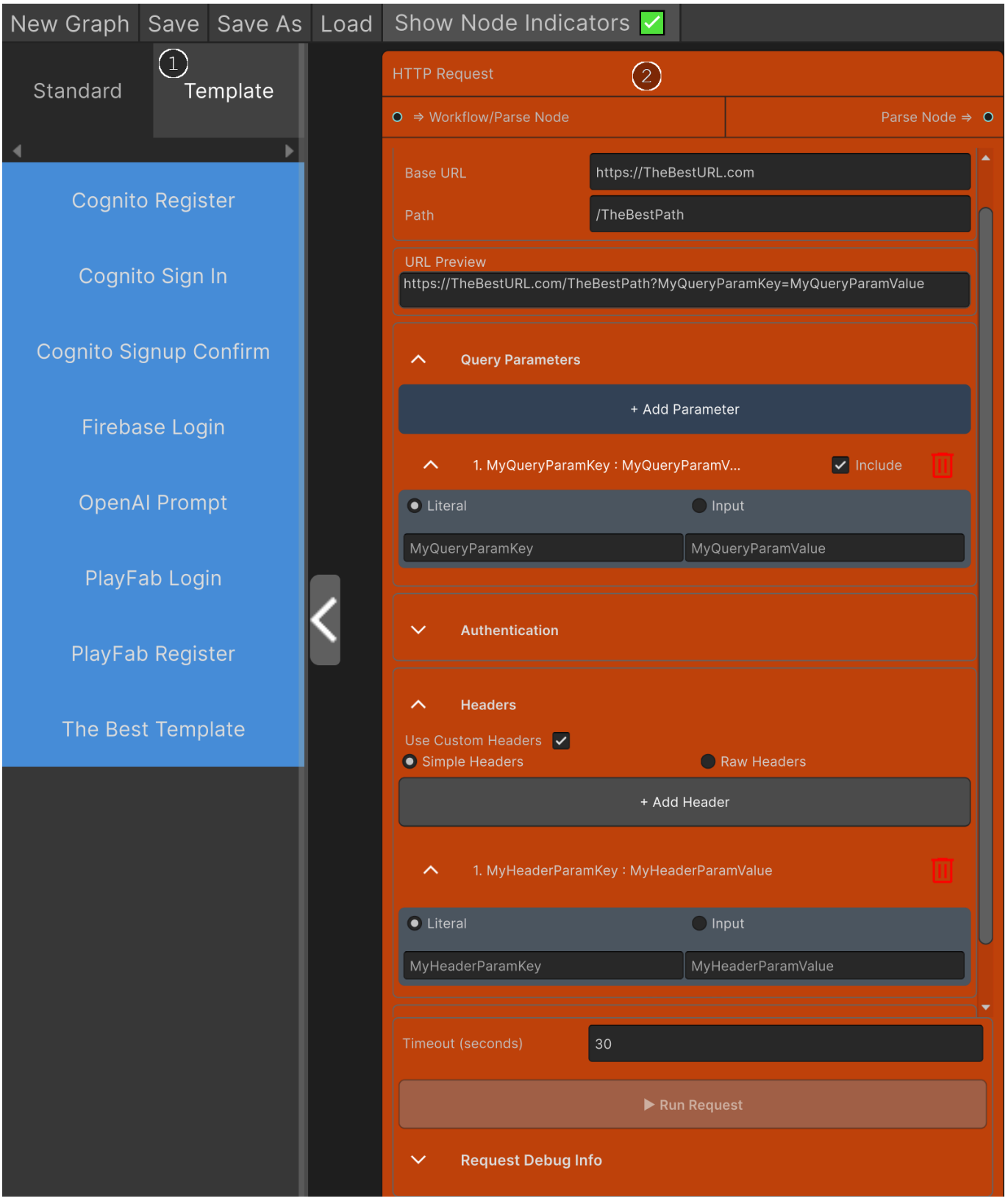


Fields

- [1]**Template Name:** This is what the template label will be in the graphs templates list (if this is empty the template will not be displayed).
- [2]**Background Color:** The color you want your template to have in the graphs templates list.
- [3]**Various:** The rest of the fields will be the same as you would set on an HTTP Request Node.



You can now see your template in the graphs templates list by selecting the [1]**Template Tab**. You can then drag & drop into the graph to create a new [2]instance.



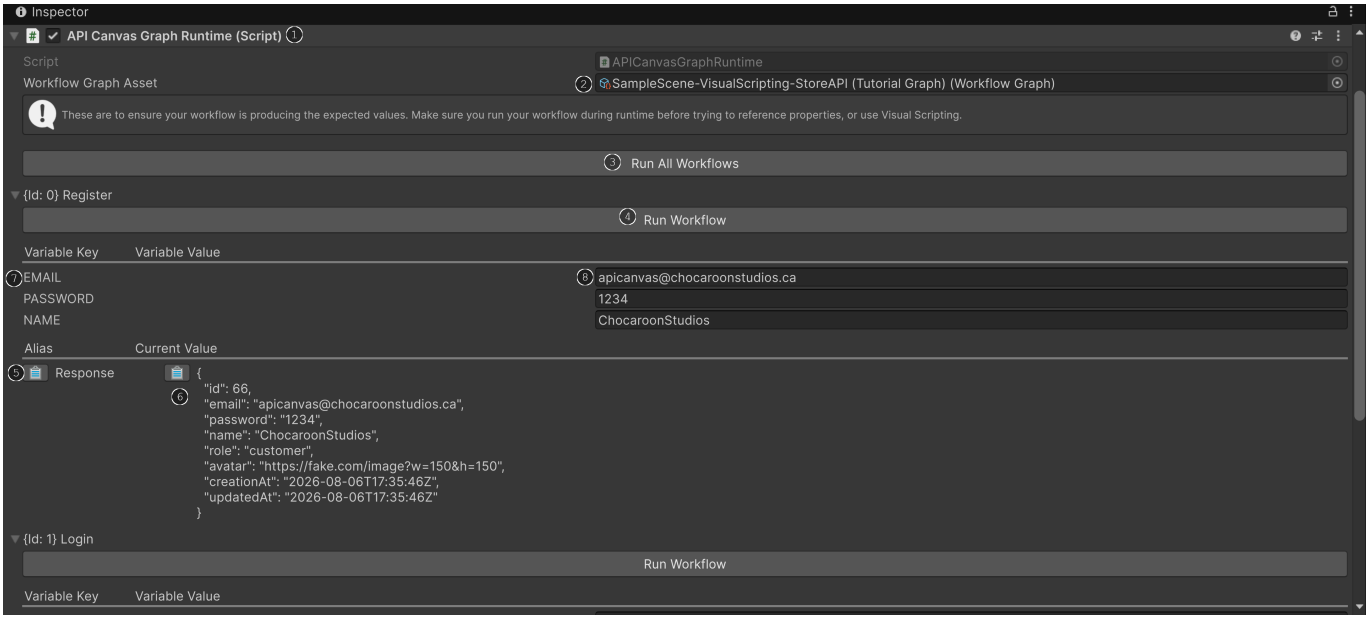
Scene Setup

To use a graph in a scene (one graph per scene), add an [1]**APICanavasGraphRuntime** component to a GameObject, and set its [2]**Workflow Graph Asset** to your saved graph asset. You can set request variable values & run workflows here to test them (values will not carry into runtime). During play mode these will update so you can see request variable values that have been set & properties that have been populated. You can also use the corresponding copy buttons to copy aliases & values.

Fields/Properties/etc.

- [3]**Run/Cancel All Workflows:** Runs/Cancels all the workflows present on the graph.
- [4]**Run/Cancel Workflow:** Run/Cancel the corresponding workflow.
- [5]**Alias:** The property alias.
- [6]**Current Value:** The current property value.
- [7]**Request Variable Key:** The request variable key.
- [8]**Request Variable Value:** The request variable value.

Important Note: The inspector view/functions are merely to for quickly testing your workflow to ensure they work & to debug in the editor during play mode. These values will not persist when transitioning to/from play mode.

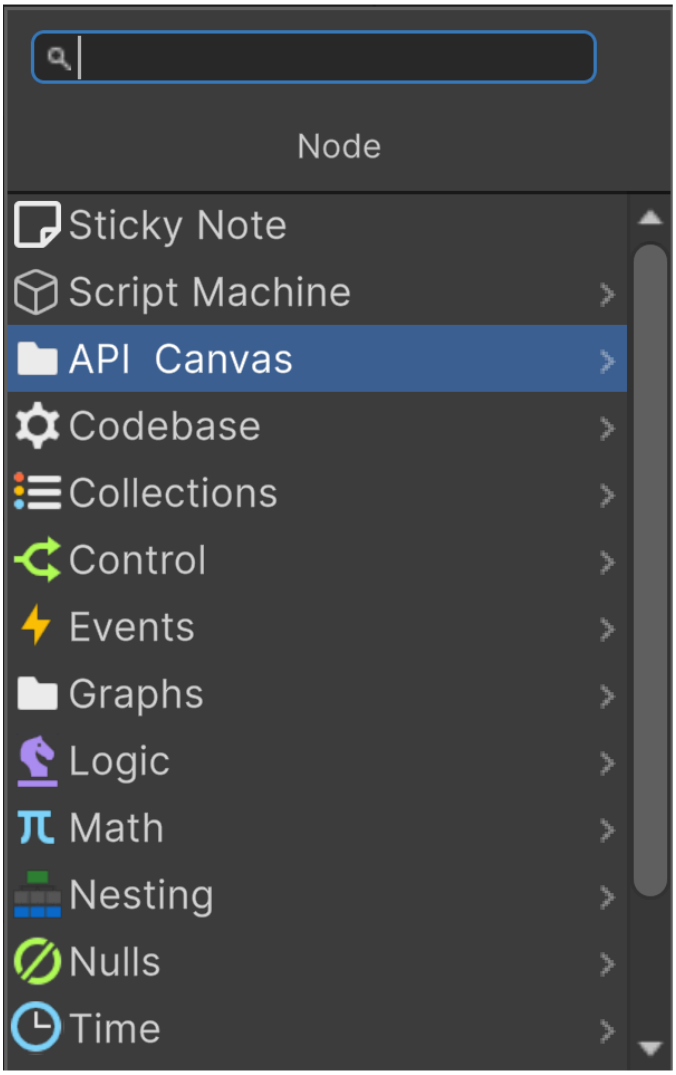


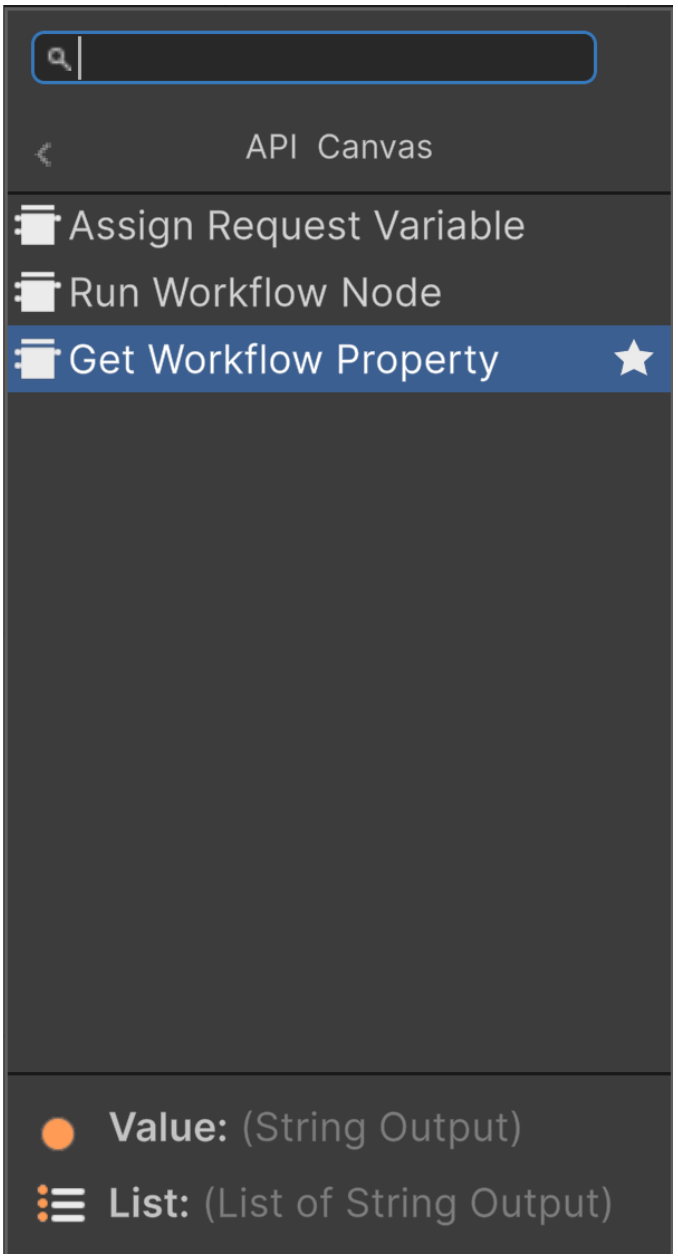
How to use in Visual Scripting

Once your graph is setup you can perform all needed functionality using the Visual Scripting graph using API Canvas' custom nodes. Make sure an instance of **APICanvasGraphRuntime** is present in your scene & its **WorkflowGraphAsset** is assigned with the graph asset you want to use. I would suggest creating a new Script Graph for every API Canvas graph you plan on using.

Create a Node

To create an API Canvas node, go into the add node menu (right click on graph) and you'll find them in the API Canvas section.



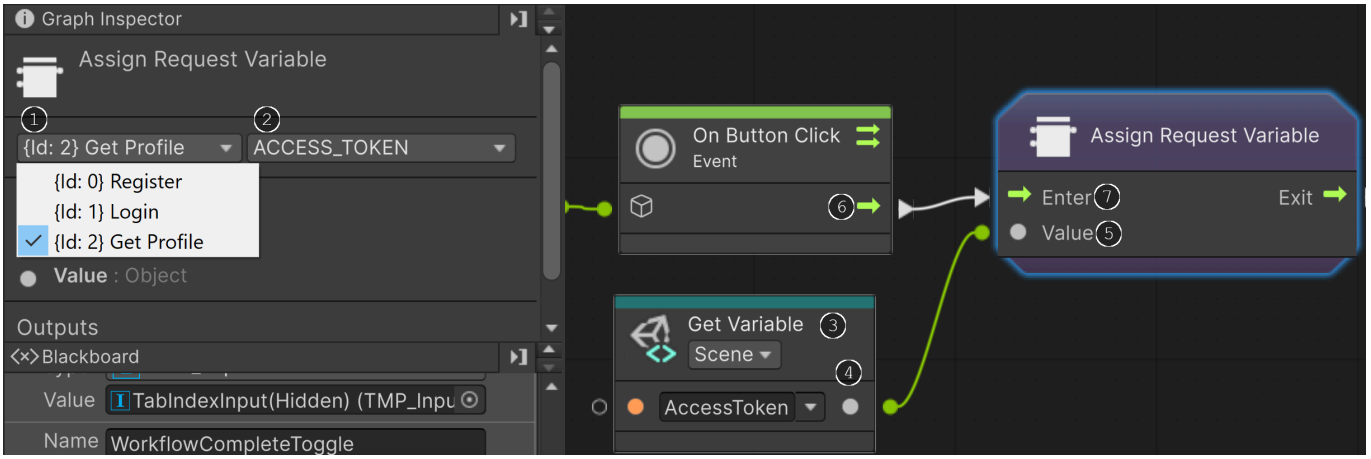


Assign Request Variable Node

Use this to assign a value to a request variable.

1. In the graph inspector, select the corresponding [1]workflow and choose the variable you want to assign to from the [2]list of request variables connected to that workflow.
2. Set up where your value is going to come from, such as an [3]Get Variable Node.
3. Connect the [4]**Value Output** to the Assign Request Variable input [5]**Value Input**.
4. Connect your event node's [6]**Exit Control Port** to the Assign Request Variable nodes [7]**Enter Port**.

The value is now assigned and will be used in your workflow.



Run Workflow Node

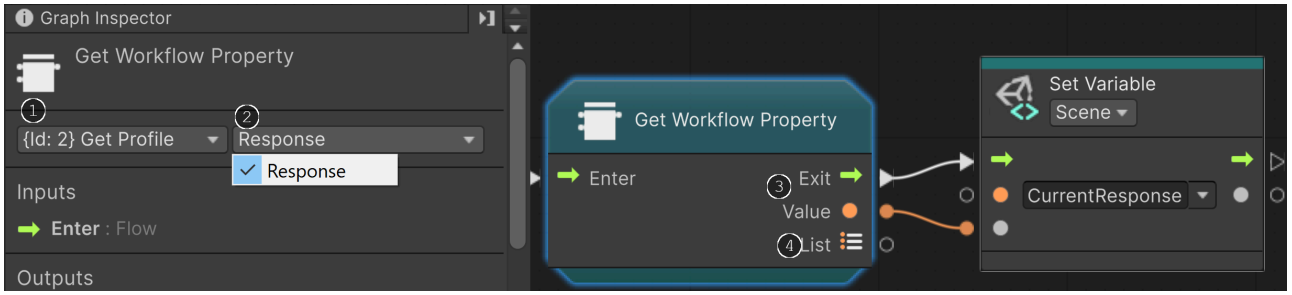
Run your workflow when ready using this node. Simply [1]select the `Workflow` in the graph inspector and trigger it as with any other node by [2]connecting to its **Enter Port** with an event.

Important Note: The event that triggers this must have its Coroutine toggle on as running a workflow is an async operation.

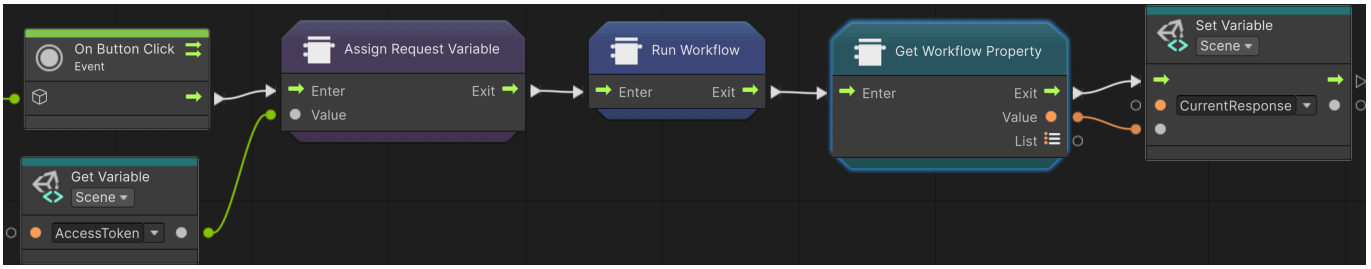
Get Workflow Property Node

Use these to actually get the properties you set up in your Extract Properties node on the graph.

1. Select the [1]`Workflow` & [2]`Alias` of the property.
2. Connect the Get Workflow Property Node(s) to the event chain (I'd suggest using a Sequence Node if retrieving multiple properties).
3. Use the output value as needed by connecting the [3]**Output Value Port** & **Exit Port** to any node that accepts them (if the property is an array you can use the [4]**List Output Port** and iterate through it as any other list).



***Note:** You do not have to use the properties immediately after running the workflow. Once the workflow runs the properties are ready to use when you need them.*



How to use via Code

Using this in your own code is simple. Make sure an instance of **APICanvasGraphRuntime** is present in your scene & its **WorkflowGraphAsset** is assigned with the graph asset you want to use. Below are the important methods you'll most likely be using.

Methods

public WorkflowRuntime GetWorkflow(int workflowId): Get a workflow.

```
     private void GetWorkflowExample(int workflowId)
    {
        APICanvasGraphRuntime graphRuntime = APICanvasGraphRuntime.Active;
        WorkflowRuntime workflow;

        if (graphRuntime == null)
        {
            Debug.LogError("Graph runtime is null.");

            return;
        }

        workflow = graphRuntime.GetWorkflow(workflowId);

        if (workflow == null)
        {
            Debug.LogError("Workflow runtime is null.");

            return;
        }

        // Do whatever you want with the workflow
    }
```

public async Task RunWorkflow(int workflowId): Run a specific workflow.

```
private async Task RunWorkflowExample(int workflowId)
    {
        APICanvasGraphRuntime graphRuntime = APICanvasGraphRuntime.Active;
```

```
        WorkflowRuntime workflow;
        bool workflowWasSuccessful;

        // Make sure the graph runtime instance is in the scene
        if (graphRuntime == null)
        {
            Debug.LogError("Graph Runtime instance does not exist!");

            return;
        }

        workflow = graphRuntime.GetWorkflow(workflowId);

        // Make sure the workflow exists
        if (workflow == null)
        {
            Debug.LogError($"Workflow with Id {workflowId} does not exist!");

            return;
        }

        // This would be where you show a loading/connecting/etc. screen
        workflowWasSuccessful = await graphRuntime.RunWorkflow(workflowId);
        // Or, since we have the actual workflow
        workflowWasSuccessful = await workflow.ExecuteWorkflow();

        if (workflowWasSuccessful)
        {
            // Get all the properties from the workflow
            foreach (ExtractedPropertyEntryData property in
workflow.ExtractPropertiesRuntime.CurrentProperties.Values)
            {
                // Do something with the property value
                Debug.Log($"Alias: {property.Alias} - Value:
{property.ExtractedValue}");
            }

            // Or we could get a specific property
            ExtractedPropertyEntryData specificProperty =
workflow.GetProperty("MyPropertyAlias");

            if (specificProperty != null)
            {
                Debug.Log($"Alias: {specificProperty.Alias} - Value:
{specificProperty.ExtractedValue}");
            }
        }
    }
```

public async Task RunAllWorkflows(): Run all the workflows on the graph at once.

```
        private async Task RunAllWorkflowsExample()
        {
            APICanvasGraphRuntime graphRuntime = APICanvasGraphRuntime.Active;

            // Make sure the graph runtime instance exists in the scene
            if (graphRuntime == null)
            {
                Debug.LogError("Graph Runtime instance does not exist!");

                return;
            }

            // This would be where you show a loading/connecting/etc. screen
            await graphRuntime.RunAllWorkflows();

            // Do something with them after they complete
            foreach (WorkflowRuntime workflow in
graphRuntime.WorkflowRuntimes.Values)
            {
                // Get all the properties from the workflow
                foreach (ExtractedPropertyEntryData property in
workflow.ExtractPropertiesRuntime.CurrentProperties.Values)
                {
                    // Do something with the property value
                    Debug.Log($"Alias: {property.Alias} - Value:
{property.ExtractedValue}");
                }
            }
        }
```

public void CancelWorkflow(int workflowId): Cancel a specific workflow. **public void CancelAllWorkflows():** Cancel all running workflows.

public ExtractedPropertyEntryData GetProperty(int workflowId, string alias): Get a property from a workflow. Alternately, if you have the workflow reference already you can call `workflow.GetProperty(string alias)`

```
private void GetPropertyExample(int workflowId, string alias)
        {
            APICanvasGraphRuntime graphRuntime = APICanvasGraphRuntime.Active;
            ExtractedPropertyEntryData property;

            // Make sure the graph runtime instance is in the scene
            if (graphRuntime == null)
            {
                Debug.LogError("Graph Runtime instance does not exist!");

                return;
            }
```

```
            property = graphRuntime.GetProperty(workflowId, alias);
            // If the workflow or property was not found (make sure you ran the
workflow first) the above will log errors in the console

            if (property != null)
            {
                Debug.Log(property.ExtractedValue); // or whatever you want to do
with it
            }
        }
```

Example Scenes

There are several example scenes included in APICanvas/Samples/Scenes.

- **Open-Meteo Weather API** - Calls the Open-Meteo Weather API & gets back forecasts for current, or future weather up to 16 days in advance. There is an example using only the Visual Script Graph, as well as one using code.
- **Platzi Fake Store API** - Calls the Platzi Fake Store API, mainly to demonstrate registration & login. The registration email can be fake (must be formatted correctly) and the email/password is actually required to login, obtain an access token, and use other endpoints. There is an example using only the Visual Script Graph, as well as one using code.

Links

Chocaroon Studios Homepage API Canvas Product Page API Canvas Latest Documentation YouTube Page (Tutorials) Unity Publisher Page (Coming Soon) Unity Asset Store (Coming Soon)

FAQ

Q: My request is failing. How do I figure out why? A: If the error log in the console is not specific enough, look at the debug info field at the bottom of the request node and it will give detailed information on the request, including what was sent & was was returned.

Q: My request template is not showing up in the API Canvas graph editor template list. A: Make sure you have entered a name for your template. If the template name field is empty it will not show up.

Q: I don't see the dropdowns for my workflows/properties on the visual scripting nodes. A: Make sure you have gone into visual scripting in project settings and generated custom inspector properties after importing the package.

Q: I'm getting visual scripting errors after importing the package. A: This usually only happens if you've imported a different version of this package before. To fix the issue, go into visual scripting in project settings & regenerate the nodes.

Future Plans/Roadmap

- Full JSON builder: No entering raw JSON if you don't want to. Similar to the insert property behavior but for the entire JSON structure.

- Convert to C# (Standard)/C# (Unity Specific)/JavaScript/etc. code: Create .cs/.js/etc. files based on a graph/workflow setup. This would allow you to easily transfer it to different codebases outside of Unity.
- Selectable dropdown for inserting request variables (similar to how Input variables are used).
- Auto-template creation: One click solution to automatically create a template from one of your exisiting HTTP Request Nodes.
- Auto-creation of data transfer models: Instead of creating one manually in order to transfer JSON structure into a data class, this will automatically generate the class for you based on Parse Node data.
- Auto-conversion of image endpoints to Unity friendly types (possibly videos as well)
- Auto-conversion of audio endpoints to Unity friendly types
- Automatic save toggle: Auto-save graph on edit.
- Optional UniTask support
- Add more templates

Feedback & Contact

If you have any feedback, questions, bug reports, or anything else to do with the tool, please contact me and I will get back to you as soon as I can. For more details on this and future projects, please visit my official site.