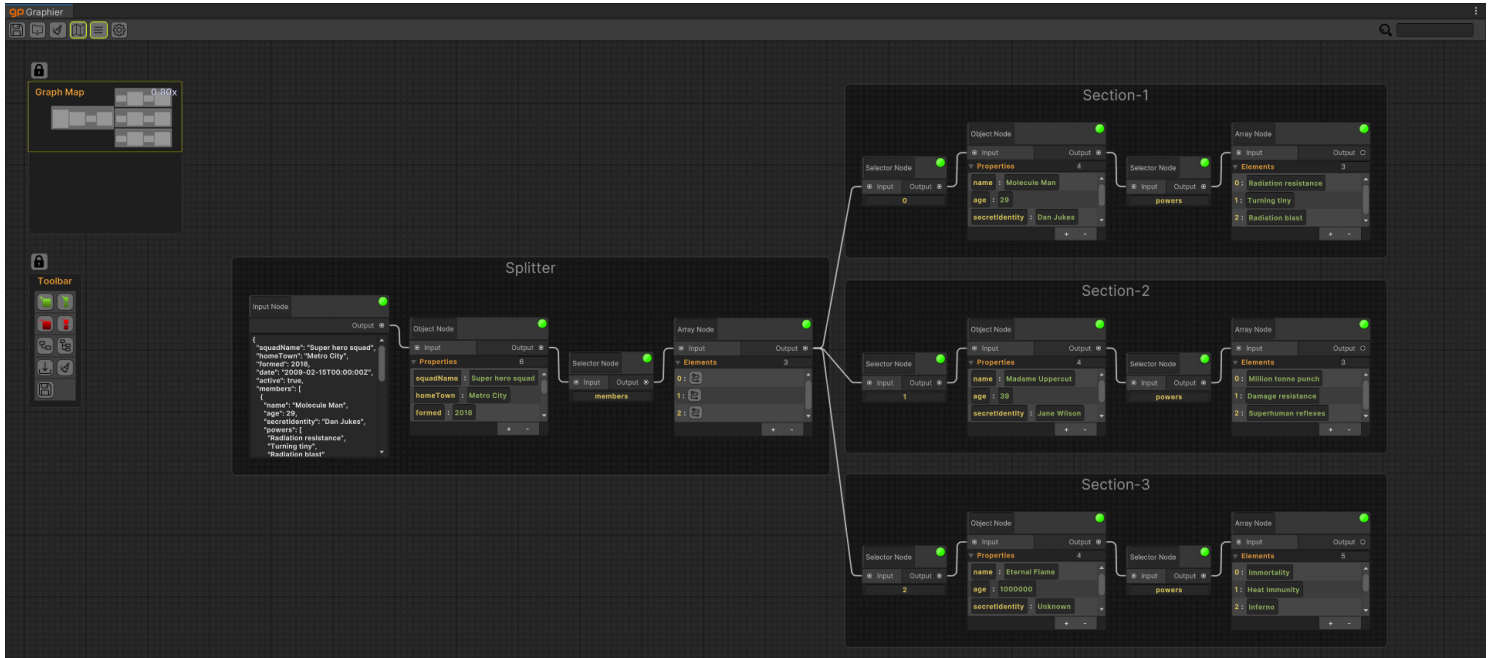


Getting Started

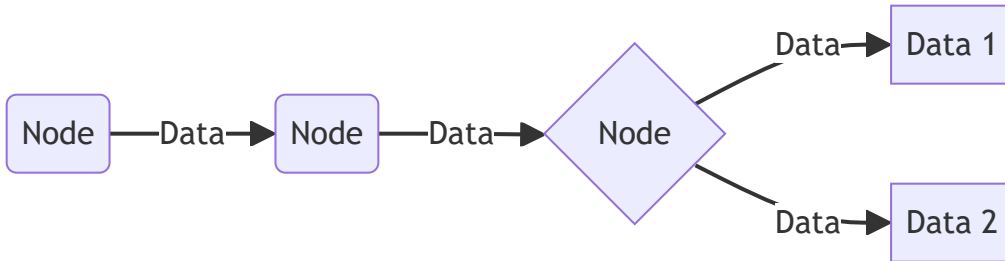
This part of the documentation shows the properties of the elements contained in the editor and simply their content.



Node

Node element is simply the element responsible for data transfer in Graphier.

It contains data, this data is transferred to other nodes through the ports that the node has.



Node

The elements at the base of all nodes are shown here.

They simply have two elements in common.




Title

It exists to distinguish and name nodes from each other.

The data of this element is also reflected on the saved scriptable object.

State

The node state element exists so that we can easily get information about the data that the node contains.

-  When red, it indicates that the data inside is null or null due to corrupt data that cannot be processed.
-  When white, it indicates that it contains data but the data is empty , an empty JObject or JArray object...
-  When green, it indicates that it contains data and is being processed properly.



Input Field

Input fields are generally elements containing text created inside various objects such as

- Input Node - Input Field
- Object Node - Property Element (Key&Value) Section
- And other elements etc.

All input fields are derived from GUITextField class and they all work in the same way.

WARNING

Input fields have a general limitation regarding the number of characters, for a detailed explanation of this, please refer to the explanations in the section below.

Character Count Limitation

In general, the number of characters is important in the following sense.

Unity uses a specific vertex and tris for each character when rendering text field objects.

At the same time, a visual element, which is a restriction of unity, can contain a maximum of 65k vertex, which restricts us at this point.

There is a way to overcome this.

The maximum character limit that an Input Field, ie text field, can contain is 12k characters.

But if you paste any text longer than 12k characters into this field, even if this text appears halfway in the render, we continue to keep the text you throw into it on the backside.

In this way, even if the sample data you want to examine inside exceeds the 12k character limit, a certain part is rendered and the entire text is properly contained inside.

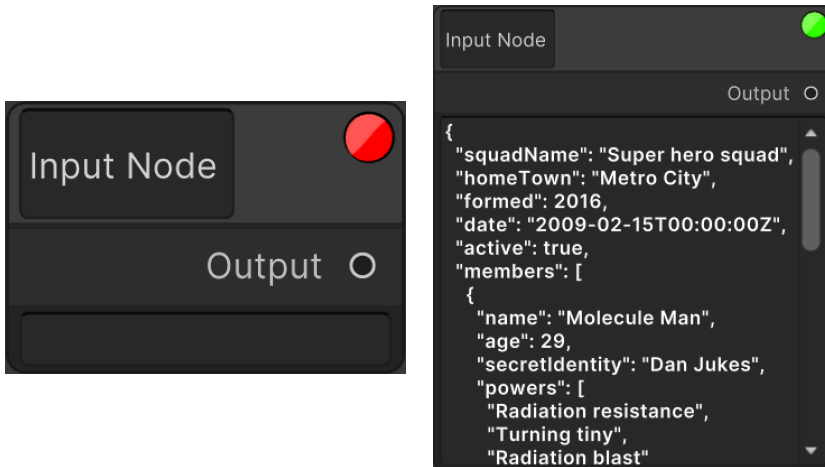
If you want to retrieve and copy this text losslessly, when you select the entire text by mouse or with a key combination such as CTRL-A and copy or cut the whole text, you will obtain the text losslessly.

In future versions, we will improve this more and eliminate this problem.

Input Node

Input node is generally the main data source of the graph.

When the data to be given as input JSON is written to the input field in this node, the data is automatically processed and transferred to the nodes to which it is connected.



Input Field

It is the input field element where data is given as raw, i.e. text.

Player Prefs

There are some extra features specific to the node.

The transfer and saving of PlayerPrefs data in Unity is also done through this node.

After selecting any Input Node, you can use the Node Toolbar

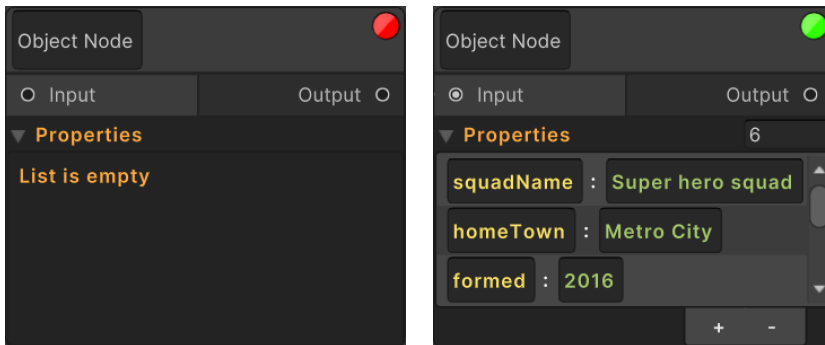
- Import PlayerPrefs
- Save PlayerPrefs
- Clear PlayerPrefs

methods for this node.

Object Node

Object node is a node that contains and manipulates JObject data.

It contains an element called Property List View and handles the operations from there.



Property List View

Property List View parses the JObject object held by the Object node and displays the JProperty objects in it in the list view.

It creates a Property block for each JProperty.

There are two different types of Property blocks.

JObject Property

A visual element that contains 2 combinations of elements:

- Key Input Field & Value Input Field (Value Property)
- Key Input Field & Focus Button (Node Property)

The key input field is responsible for changing the key of the JProperty.

The value input field is responsible for changing the value of the JProperty.

The focus button is responsible for frame and shows you where the related node.

Value Property

If the value in the JProperty is not JObject or JArray, but is an editable value type (float, string, int, etc.), it opens a value input field for this.

The value is manipulated with this input field.

If you want to convert the values of JProperties to JObject or JArray type, it will be converted automatically when you write [], {} in the value input field.

Node Property

If the value in the JProperty is JObject or JArray, the value field is not created and since this property will contain children within itself, it is determined as the next node and only a focus button is placed next to the key field.

If the focus button is pressed, it frames and select the next node and shows you where the node is.

Collection Size

You can increase or decrease the number of elements in the data with the collection size section on the right of the list view.

Add Remove Buttons

When you press the Add button, i.e. " + ", a new data is created and added with a unique key and value.

When you press the Remove button, i.e. " - ", the selected elements are deleted from the data.

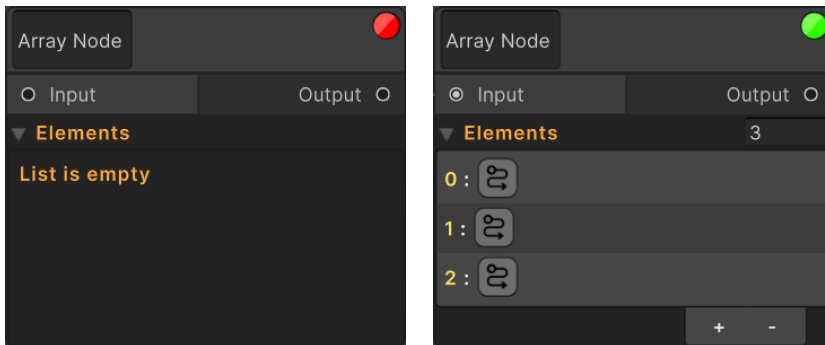
Index Change

You can change the index in the data by dragging the selected element in the list view to the desired index with the CTRL + Left Mouse Button Drag combination.

Array Node

Array node is a node that contains and manipulates JArray data.

It contains an element called Property List View and handles the operations from there.



Element List View

Element List View parses the JArray object held by the Array node and displays the JToken objects in it in the list view.

It creates a Property block for each JToken.

There are two different types of Property blocks.

JToken Property

A visual element that contains 2 combinations of elements:

- Index Label & Value Input Field (Value Property)
- Index Label & Focus Button (Node Property)

The index label is responsible for show the index of the JToken.

The value input field is responsible for changing the value of the JToken.

Value Property

If the value in the JProperty is not JObject or JArray, but is an editable value type (float, string, int, etc.), it opens a value input field for this.

The value is manipulated with this input field.

If you want to convert the values of JProperties to JObject or JArray type, it will be converted automatically when you write [], {} in the value input field.

Node Property

If the value in the JProperty is JObject or JArray, the value field is not created and since this property will contain children within itself, it is determined as the next node and only a focus button is placed next to the key field.

If the focus button is pressed, it frames and select the next node and shows you where the node is.



Collection Size

You can increase or decrease the number of elements in the data with the collection size section on the right of the list view.



Add Remove Buttons

When you press the Add button, i.e. " + ", a new data is created and added with a unique key and value.

When you press the Remove button, i.e. " - ", the selected elements are deleted from the data.

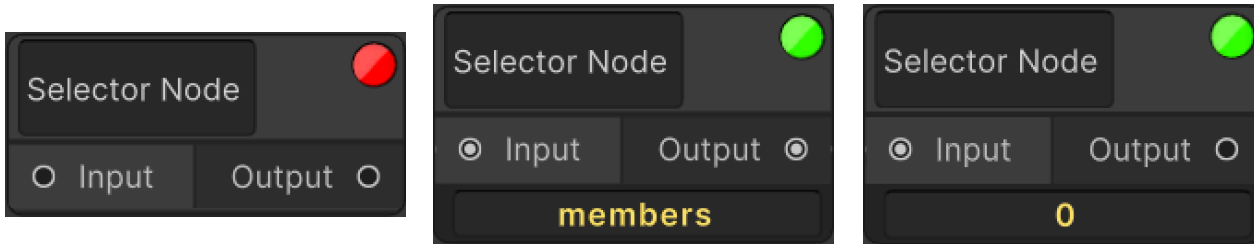


Index Change

You can change the index in the data by dragging the selected element in the list view to the desired index with the CTRL + Left Mouse Button Drag combination.

Selector Node

Selector node is the node we use to get a certain part of the data coming to it on the graph.



Input Field

It is the input field element where selection data is given, i.e key names, element indexes.

There are two different modes in it.

{ } Connected To A Object Node

When connected to the Object node, the input field part inside it expects any key in the JObject as a parser.

When we give this key, it forwards the data by taking only the JToken that has that key.

[] Connected To A Array Node

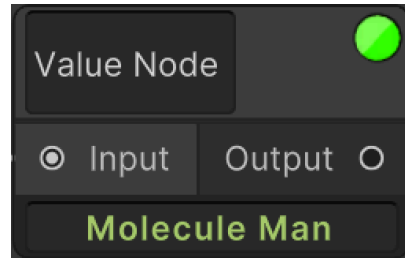
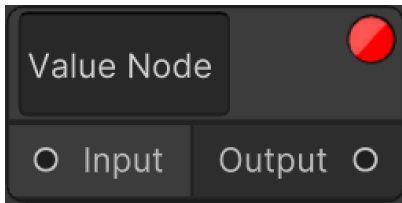
When connected to the Array node, the input field part inside it expects any index in the JArray as a parser.

When we give this index, it forwards the data by taking only the JToken in its index.

Value Node

Value node is a node that does not have much functionality compared to other nodes and can only be used to show and transmit a certain JToken.

Generally, if you want to take a certain JToken in the nodes with selector node and show it at a point, you can use this node.



Input Field

It is the input field element where data is shown and edited.

The data type it expects is all types that are JValue, for example string, int, float, datetime etc.

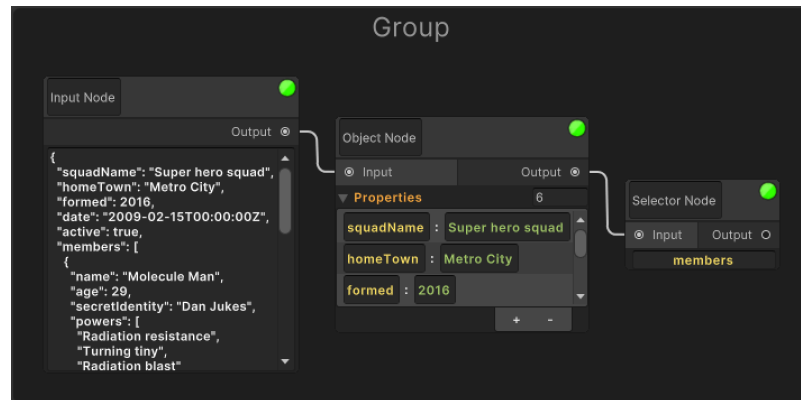
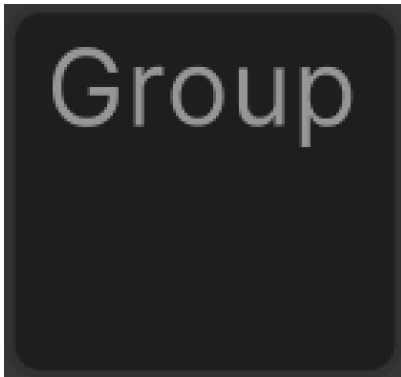
Group

The Group element helps you to easily drag and categorize certain nodes under a single group.

If you want to create it, you can click on group from the Search Window and create it.

Or you can select the nodes you want to group and right click and use the group action from the Contextual Menu to group them under a group.

They simply have one elements in common.



ID Title

It exists to distinguish and name groups from each other.

The data of this element is also reflected on the saved scriptable object.

Manually Grouping Nodes

After creating the Group element, drag and drop the nodes to the empty area within the group with Left Click and they will be included in the group element.

Manually Removing Nodes from the Group

After selecting the node within the group element, you can drag and drop the nodes from the group with the Shift + Left Click combination.

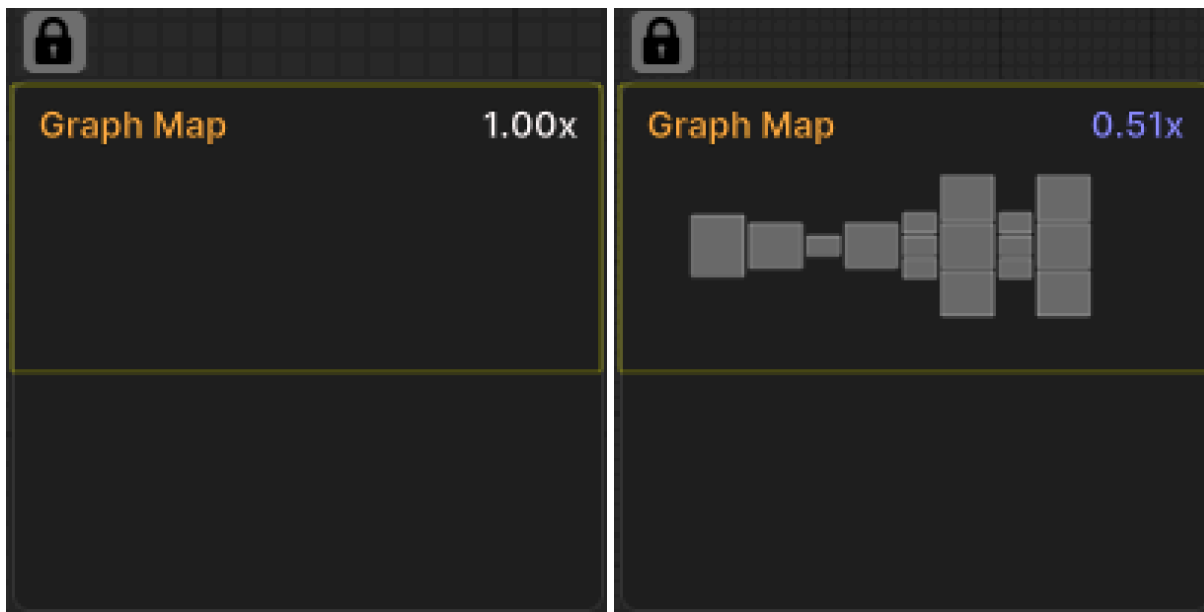
Map

It is a mini-map element that shows all nodes and groups in the editor.

You can activate or deactivate the map from the map icon in the Editor Toolbar above.

You can easily focus on the locations of the elements on the map by clicking on them with the left click.

You can bring the map to the desired location and lock it with the anchor button on the map.



Node Toolbar

It is a node-specific toolbar element that shows all node operations in it. You can bring the Node Toolbar to the desired location and lock it with the anchor button on the Node Toolbar.



Creation Operations

These are operations in which other nodes are automatically created or deleted according to the data in the nodes.

Creation

There are two different operations in creation operations, both work with the same logic.

When you select any node and click on Create Next Node on the toolbar, if there is a data that can be created in the node, for example JObject, JArray, these nodes are automatically created and connected to the selected node.

When you select any node and click on Create All Nodes on the toolbar, Create Next Node runs and if it contains data that can be created in the next node, it is run recursively until this process is completed and all nodes are created.

Deletion

There are two different operations in deletion operations, both work with the same logic.

When you select any node and click Destroy Next Node on the toolbar, if there is a suitable connection in the node, the next node is automatically deleted.

When you select any node and click Destroy All Nodes on the toolbar, Destroy Next Node runs and if the next node contains suitable connections, it is run recursively until this process is completed and all connected nodes are deleted.

Organization Operations

There are two different operations in organization operations, both work with the same logic.

When you select any node and click Organize Next Node on the toolbar, if there is a suitable connection in the node, the next node is automatically organized and its position is updated.

When you select any node and click Organize All Nodes on the toolbar, Organize Next Node runs and if the next node contains suitable connections, it is run recursively until this process is completed and all connected are organized and their positions are updated.

PlayerPrefs Operations

PlayerPrefs operations include three different operations.

If there is an input node among the selected nodes, you can use these three operations, they are three operations specific to the input node.

Import

The Import operation collects all PlayerPrefs data stored on the platform and manipulates it by placing it as raw text in the input field of the selected Input Nodes.

Windows

All PlayerPrefs data is read and processed from the relevant Regedit block values.

If the read values match the company name and product name specific to the project, data is transmitted.

Linux

All PlayerPrefs data is read and processed from the relevant Pref file.

If the read values match the company name and product name specific to the project, data is transmitted.

Mac OS

All PlayerPrefs data is read and processed from the relevant Plist file.

If the read values match the company name and product name specific to the project, data is transmitted.

Save

The Save operation collects the JTokens with key value pair relationships in all selected Input Nodes, namely their JProperties.

Then, if these JTokens contain data suitable for PlayerPrefs, namely if they are of type (string, float, int), they are categorized and recorded in the platform according to the keys they are determined.

Clear

The Clear operation clears and saves all PlayerPrefs data on the platform.

Toolbar

It is a editor specific toolbar element that shows editor operations in it.



Save Graph

This is the operation button used to save the graph.

Load Graph

This is the operation button used to load the Graph via ScriptableObjects.

Clear Graph

It is the operation button used to clear the current graph.

Graph Map

It is the operation button that is used to open and close the Map element in the Graph.

Node Toolbar

It is the operation button that is used to open and close the Node Toolbar element in the Graph.




Graph Settings

It is the operation button that is used to open and close the Graph Settings element in the Graph.

Settings




The graph editor settings is also the element where custom settings are found. It currently consists of three different sections.

Setting






Use Animation ☒

Setting



Include Input Node ☐

Setting



Live Editing

Key ☒

Value ☒

Selector ☒

Input ☒

Performance

This section contains the performance settings found on the editor.

Animation Settings

This section allows nodes to be animatedly moved while being organized, but therefore manipulates one tween for each node being moved.

This can cause performance degradation if used for a large number of nodes.

When you turn off animation, the organizing methods will instantly move nodes to where they need to be, cheaply.

Editing

This section contains the data editing settings found on the editor.

Live Editing Settings

If you want to see the effect instantly when you make any changes in the editing of input fields in the editor, you can open the live editing section of the sections you want.

Otherwise, if live editing is off, the changes you make in the input fields will not be reflected instantly, whenever you click on a different point from the input field, it will be changed at that moment with the out of focus event and its effects will be applied.

Points that support this option:

- Input Node
 - Input Field
- List View Property
 - Key Field
 - Value Field
- Selector Node
 - Input Field

Search

This section contains the search settings found on the editor.

Search Settings

This section allows you to select whether the content of searches should include input nodes when searching.

Because if the data you are looking for exists in the nodes, it must also exist in the input node, so you will always get double results if the data you are looking for is found.



Contextual Menu

It is a menu window element that opens when you right-click on any point in the graph view.

There are buttons for certain operations in this window.

Create Node

It is the operation button that creates a node creation request and then opens the search window.

Clipboard Operations Cut | Copy | Paste | Duplicate

These are operations that enable copying and pasting of elements within the graph.

Delete

It is the operation that enables the deletion of selected elements.

Disconnect All

This is the operation that deletes all connections of all selected nodes.

Group Operations Group | Ungroup | Clear Group

These are operations that allow the selected nodes to be included or removed from the group.

Clear group is an operation that allows the nodes in the selected groups to be removed from the group.

Collapse | Expand

These are operations that change the visibility status of the selected nodes to minimized or maximized.

Search Window

After clicking on the create node from the contextual menu, it is a search window that meets the node creation request.

The elements that can be created are shown here.

It currently has three categories. These are Data, Operation, Utility.

Nodes that carry data on the Graph are included in Data.

There is a Group element under Utility.

Operation Nodes are still under development for some special operations between data with the next version.

At the same time, if you create a connection from the ports of the nodes to any point in the graph and leave it empty, this window will open again and this time if you select the node as the example you want to create and if this starting node is of a type that can be connected to this node, an automatic connection will be created between them.

Search Bar

It is a search bar element that you can find in the upper right corner of the editor.

According to the input you enter into it, it compares the data in the elements or by looking at the titles of the elements.

If it finds a result, it selects the result and frames the camera towards that element.



Graph Logger

These are the elements that enable the actions or warnings to be displayed on the screen within the editor.

They are displayed on the left side of the editor, scrolling from top to bottom

- Red logs represent errors.
- White logs represent normal logs.
- Green logs represent successful points.