Visual Scripting for UI Toolkit

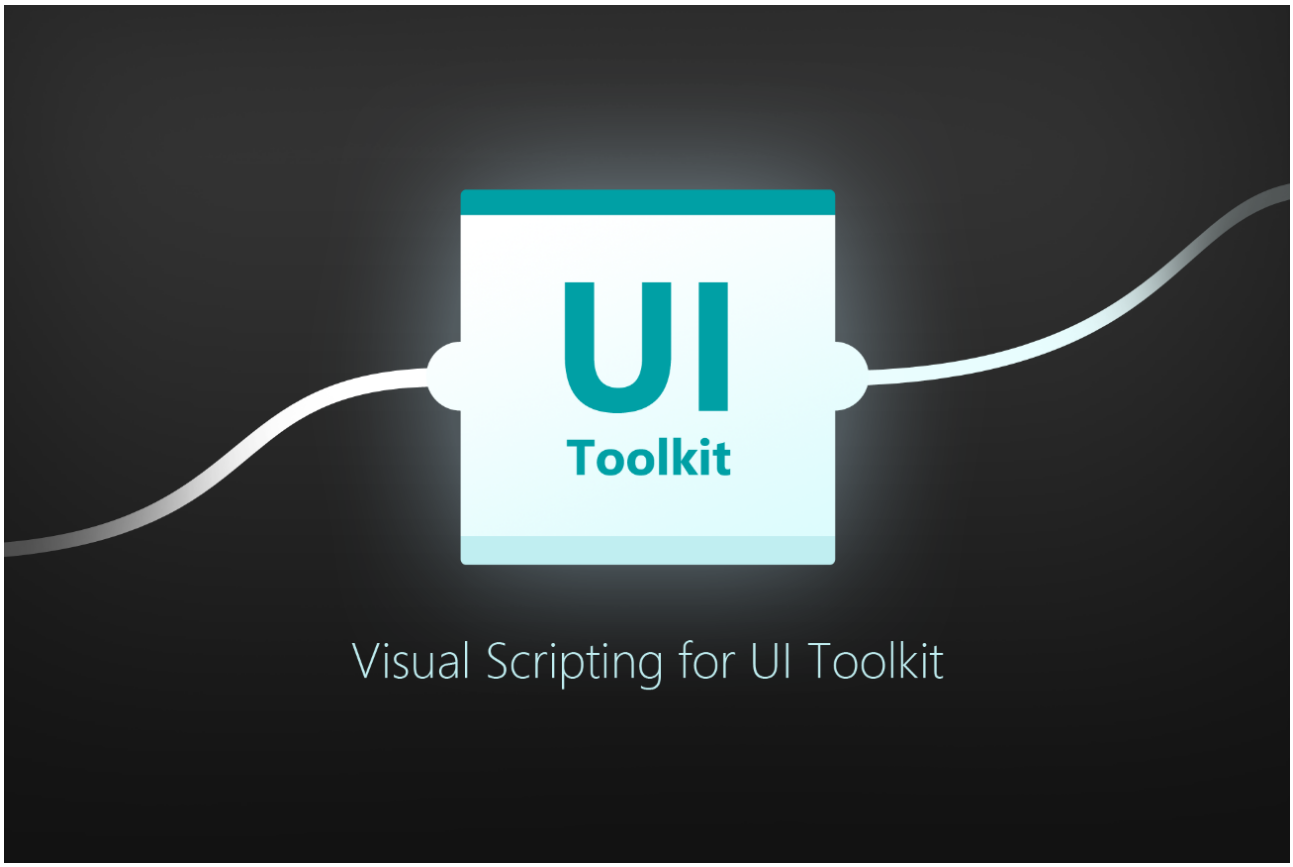


Table of contents

Requirements & Setup

Requirements

Unity 2021.2 or higher is required since that is when Unity added the UI Toolkit Module for runtime use. If you can, please upgrade to the highest LTS version of Unity. The newer the version the less „glitches" the UI Toolkit has.

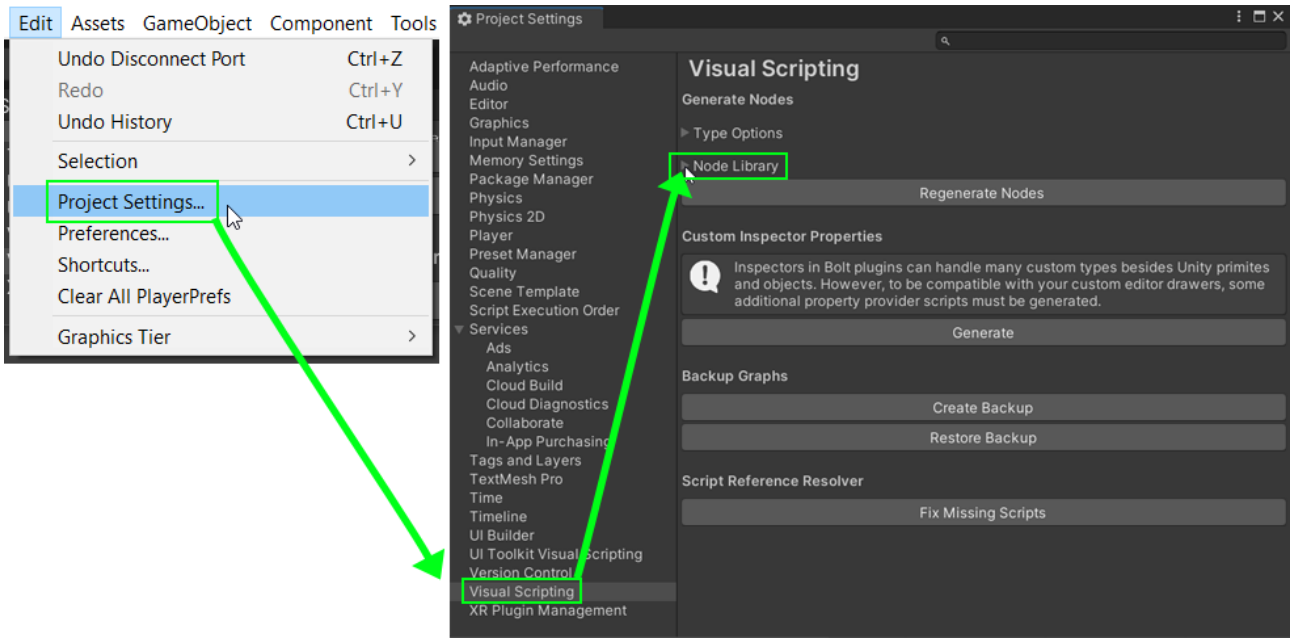
Keep in mind, UI Toolkit as a whole is still a work in progress and not quite ready for prime time. Unity itself still recommends using UGUI instead of UI Toolkit for runtime applications (source).

Visual Scripting 1.7.2 or higher is required.

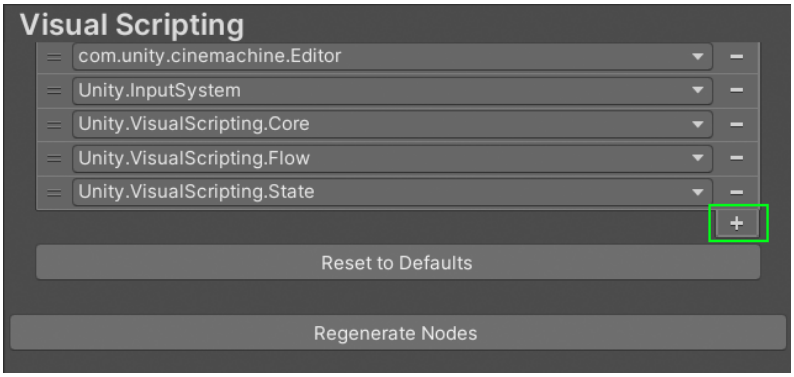
Manual Setup

Usually the automatic setup adds the Kamgam.UIToolkitVisualScripting assembly to your Node Library and rebuilds the nodes. In case this fails then here is the manual way to do it.

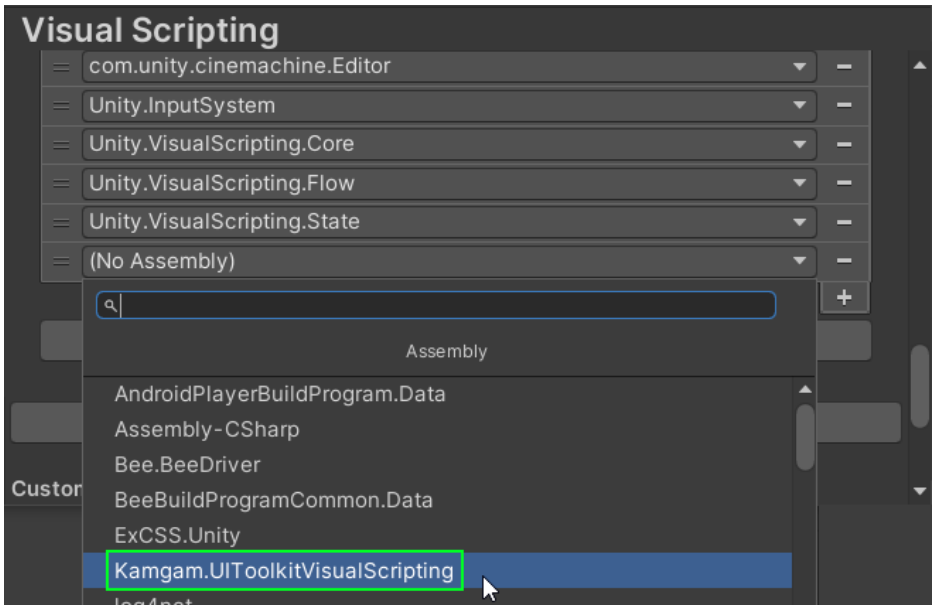
1) Open your project settings via **Edit** > **Project Settings** > **Visual Scripting** > **Node Library** (click the arrow left to „Node Library").



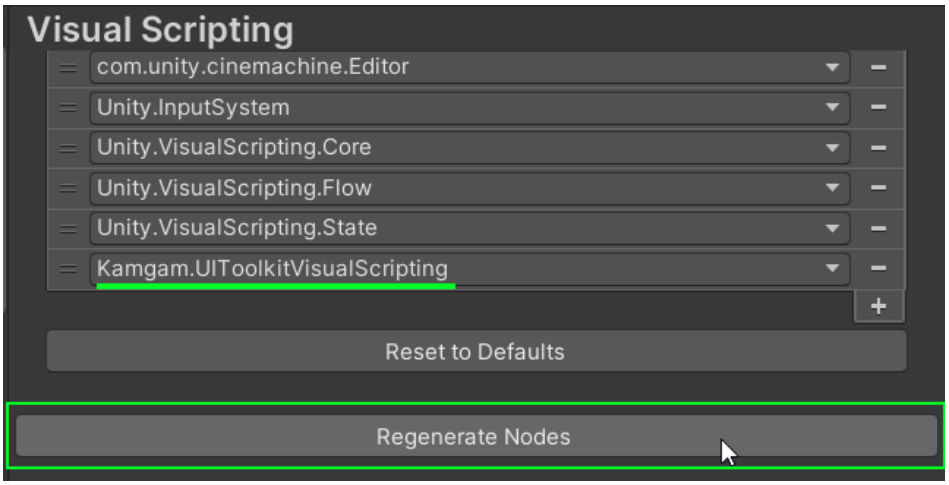
2) Scroll way down in the Node Library and click the [+] button on the right.



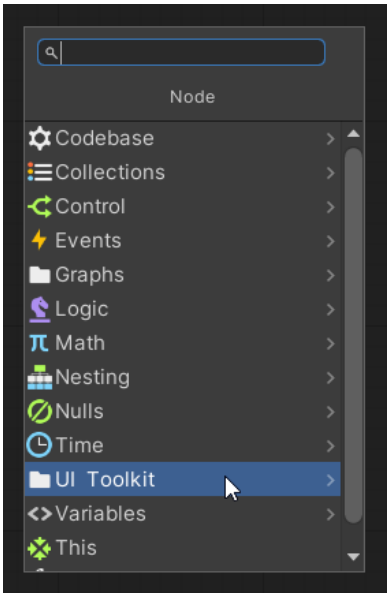
3) Select **Kamgam.UIToolkitVisualScripting** from the list.



4) Hit the „Regenerate Nodes" button (this will take a while).



5) That's it, you are done. If the fuzzy finder now shows a „UI Toolkit" section then it worked.

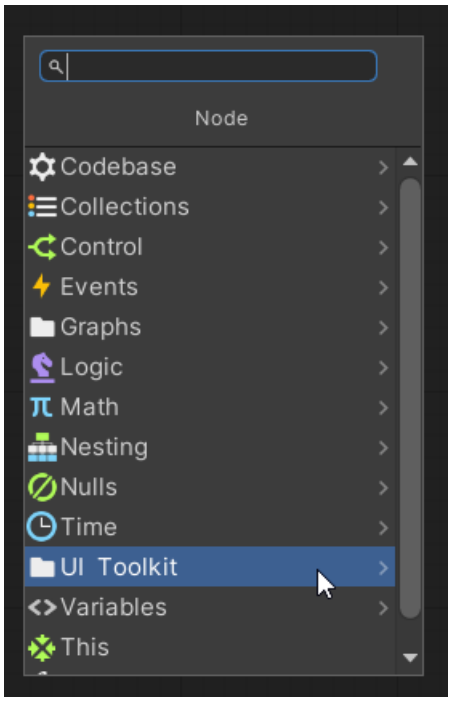


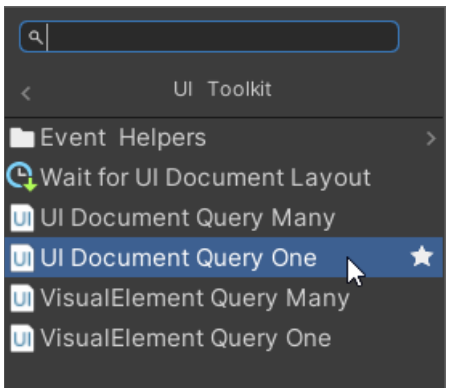
First steps (Please read this before you start)

There are three ways to add UI Toolkit nodes to your graph (A, B, C).

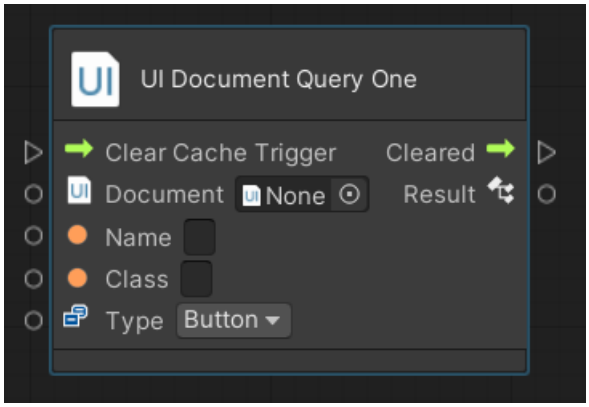
A) Right-Click > **UI Toolkit** > **...**

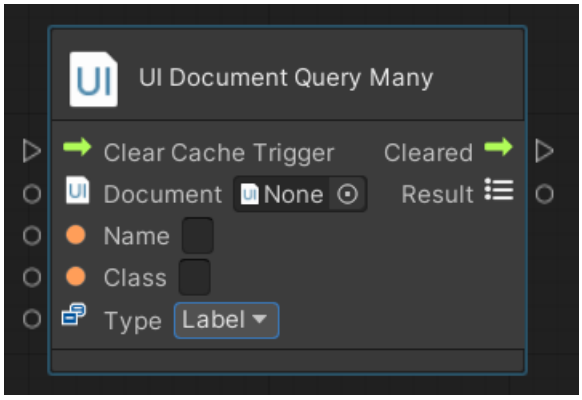
In this menu you will find the basic nodes needed for accessing the UI Hierarchy.





The nodes that you will probably use the most are the query nodes. These are the entry points for fetching ui elements from the UI Toolkit visual hierarchy. They are used to query the document for elements.

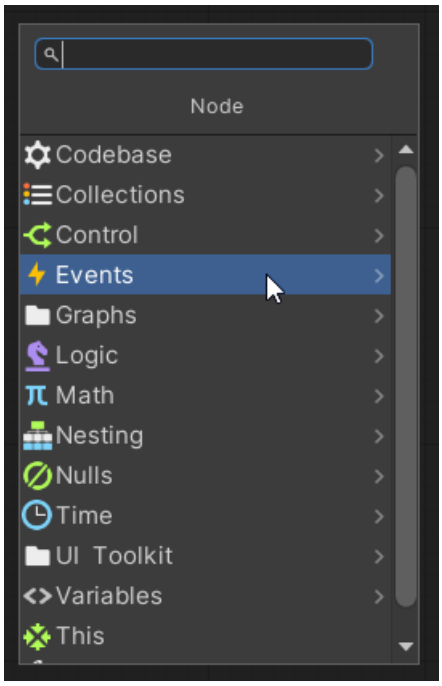


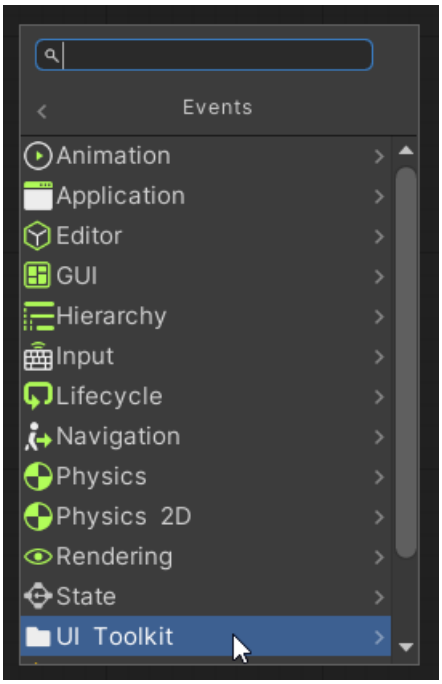


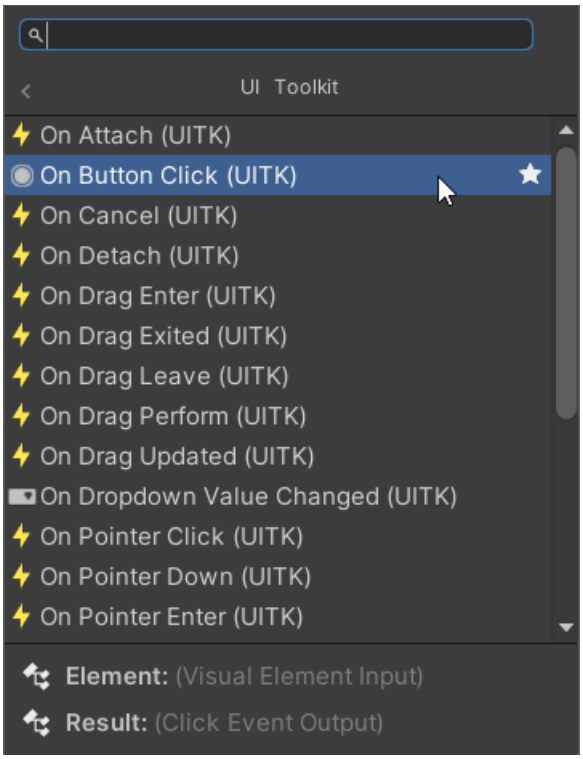
The „Results“ of the query nodes are VisualElements which you can then add events to (see B) or change via extension nodes (see C).

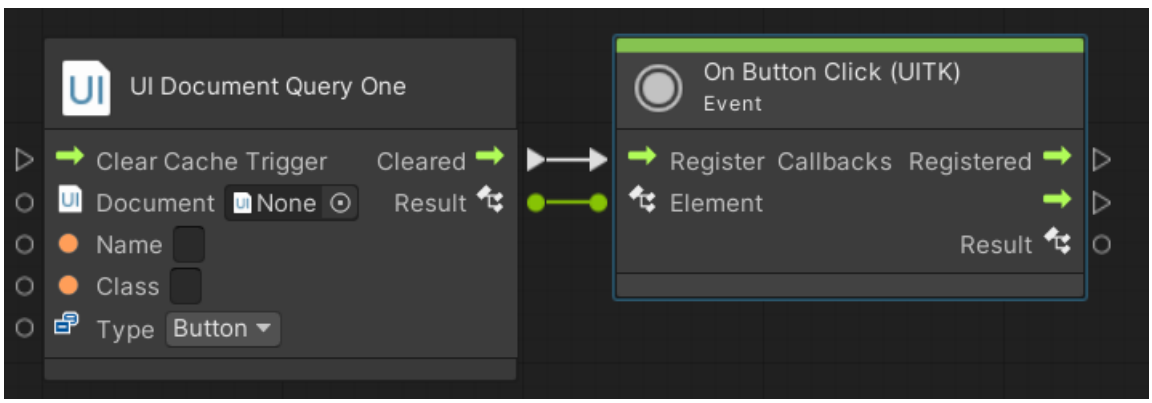
B) Right-Click > **Events > UI Toolkit**

Use these event nodes to register events to the VisualElements returned by the query nodes.



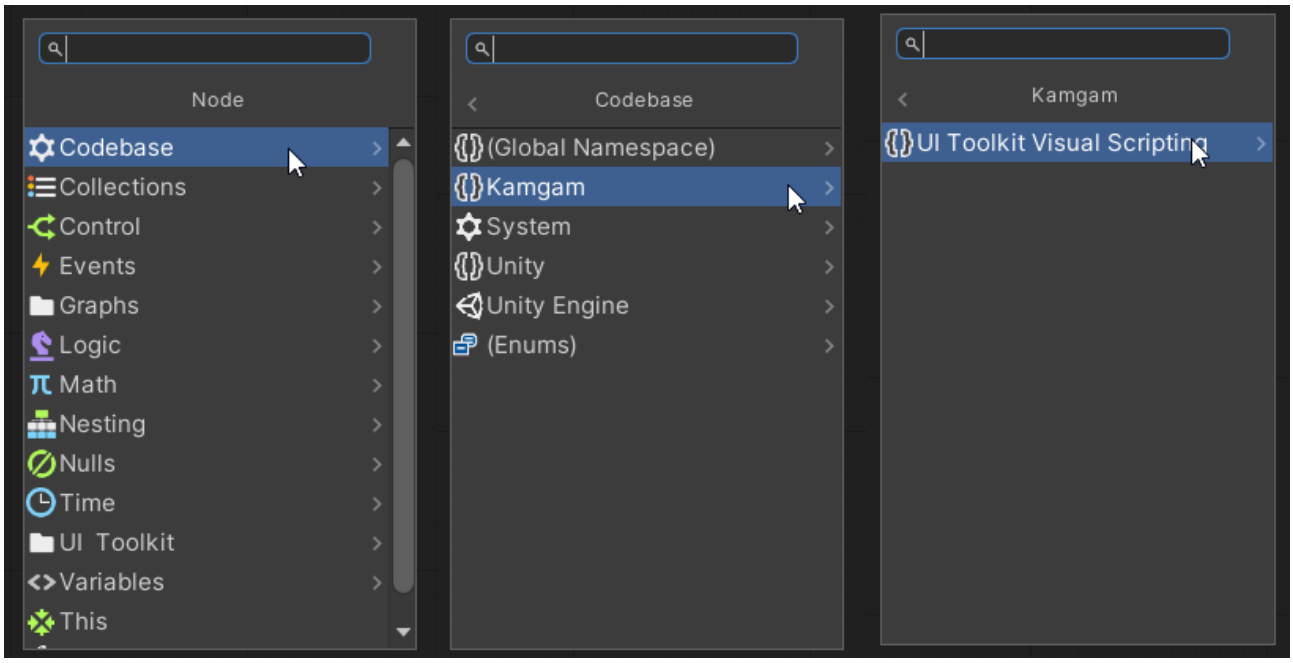




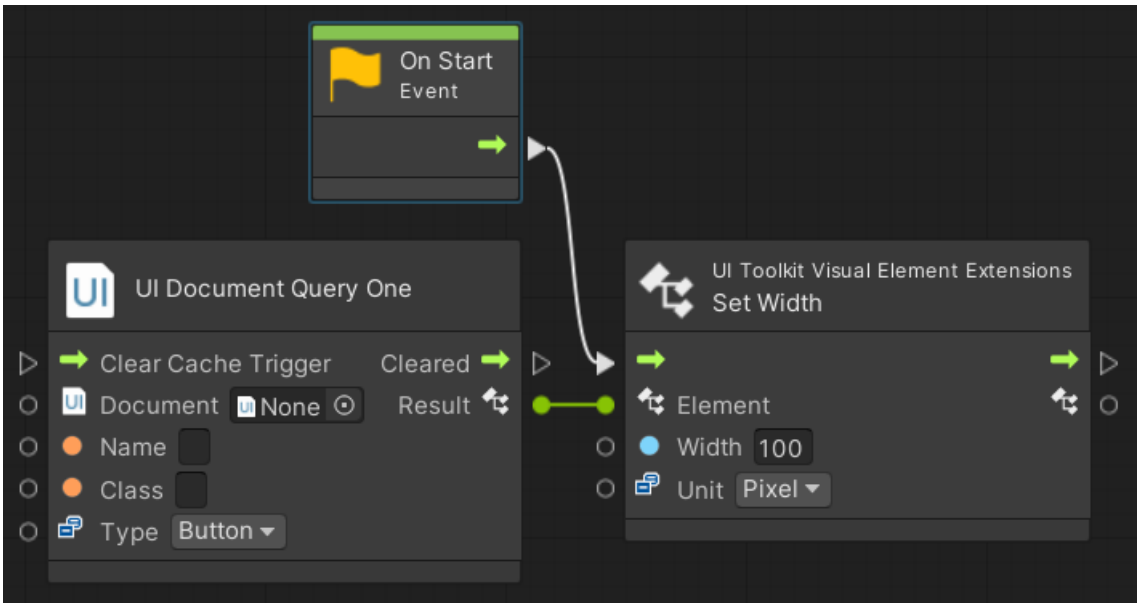


C) Right-Click > **Codebase** > **Kamgam** > **UI Toolkit Visual Scripting** > ...

The extension nodes provide you with a lot of shortcuts to access and modify VisualElements.



In the example below we set the width of a Button to 100 px at the start.

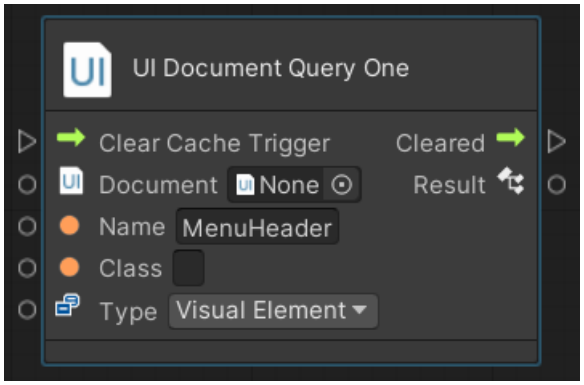


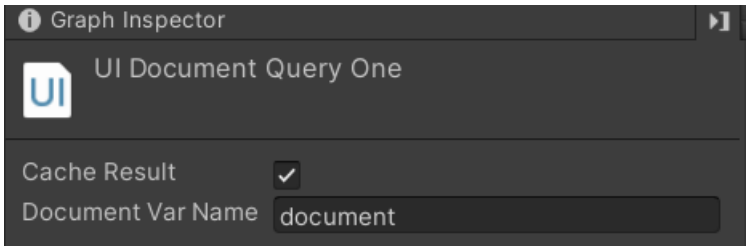
Query Nodes

The process of getting hold of a visual element in UI Toolkit is a bit different than the usual Unity way. To fetch an element from the visual hierarchy you need to query the document for it. Here is a list of query nodes that will help you with this.

UI Document Query One

Queries one element from a UI Document. The result is a single visual element.





Inputs:

Clear Cache Trigger: (optional) Use to clear the cache and re-run the query.

Document: Takes a UI Document as input. This is the document on which the query will be executed. You can leave this empty.

If it is left empty then it will try to search for a UIDocument in this order (the first found will be used):

1. It looks for a UIDocument on „this" game object (this = the game object which the graph was added to)
2. It looks for a variable named „document" on „this" game object
3. It looks for a scene variable named „document".

The variable name that is used for searching can be configured in the inspector via „Document Var Name".

Name: The element name to use for the query. Leave empty if you do not care about the name. More infos on queries can be found in the Unity Manual.

Class: The class name to use for the query. Leave empty if you do not care about the class name. More infos on queries can be found in the Unity Manual.

Type: The element type to use for the query. Leave empty if you do not care about the type. More infos on queries can be found in the Unity Manual.

Cache Results (Inspector): If the query is used very often (in an update loop for example) then it would be a waste of resources to repeatedly query the UI. Instead we can cache the result of the last query and use that. Uncheck this option if you want to submit a new query every time.

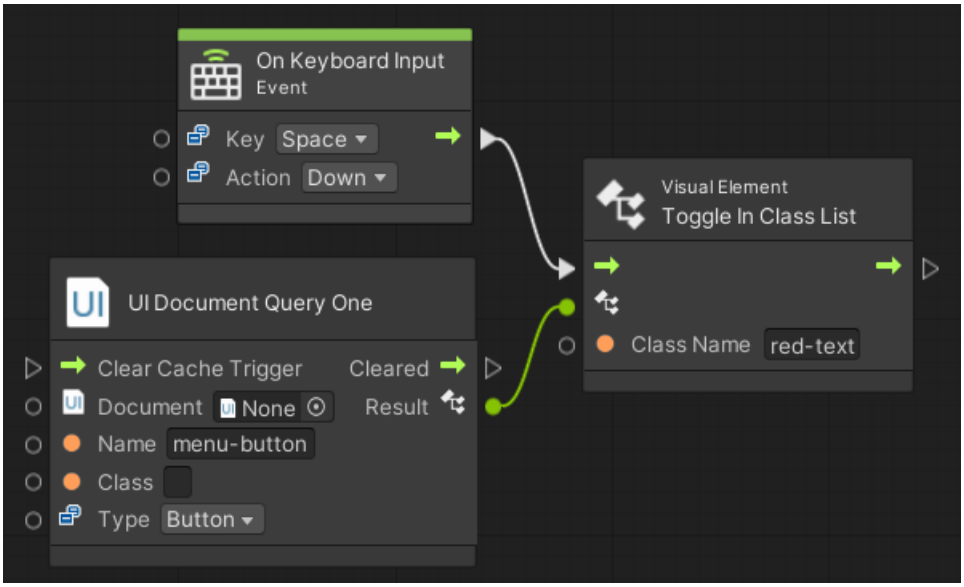
Document Var Name (Inspector): If no document is specified then the node will try to find one in the local- or scene variables. To do that it uses this name as the variable name.

Outputs:

Cleared: Called automatically after the „Clear Cache Trigger" was triggered.

Result: A single visual element (or Null). If multiple elements match the query then the first one found is used.

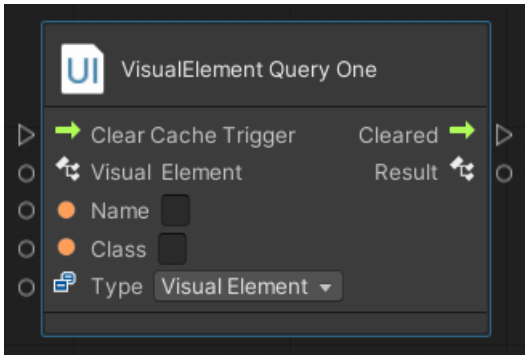
Usage:



This example toggles a class „red-text" on the „menu-button" button.

Visual Element Query One

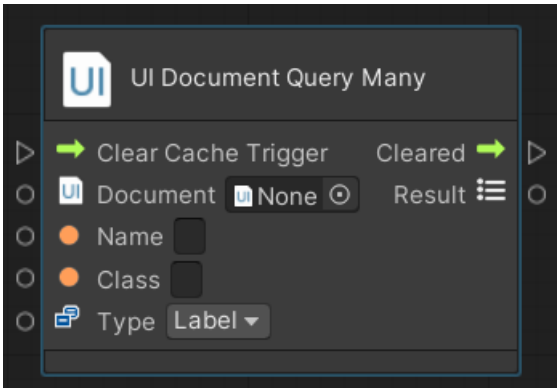
It's basically the same as UI Document Query One only that the starting element is not a UI Document but another Visual Element.

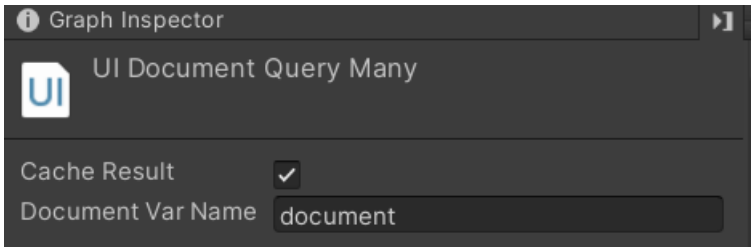


Please read the „UI Document Query One" for details on this node.

UI Document Query Many

Query for multiple elements in a UI Document. The result is a list of visual elements.





Inputs:

Clear Cache Trigger: (optional) Use to clear the cache and re-run the query.

Document: Takes a UI Document as input. This is the document on which the query will be executed. You can leave this empty.

If it is left empty then it will try to search for a UIDocument in this order (the first found will be used):

4. It looks for a UIDocument on „this" game object.
5. It looks for a variable named „document" on „this" game object
6. It looks for a scene variable named „document".

The variable name that is used for searching can be configured in the inspector via „Document Var Name".

Name: The element name to use for the query. Leave empty if you do not care about the name. More infos on queries can be found in the Unity Manual.

Class: The class name to use for the query. Leave empty if you do not care about the class name. More infos on queries can be found in the Unity Manual.

Type: The element type to use for the query. Leave empty if you do not care about the type. More infos on queries can be found in the Unity Manual.

Cache Results (Inspector): If the query is used very often (in an update loop for example) then it would be a waste of resources to repeatedly query the UI. Instead we can cache the result of the last query and use that. Uncheck this option if you want to submit a new query every time.

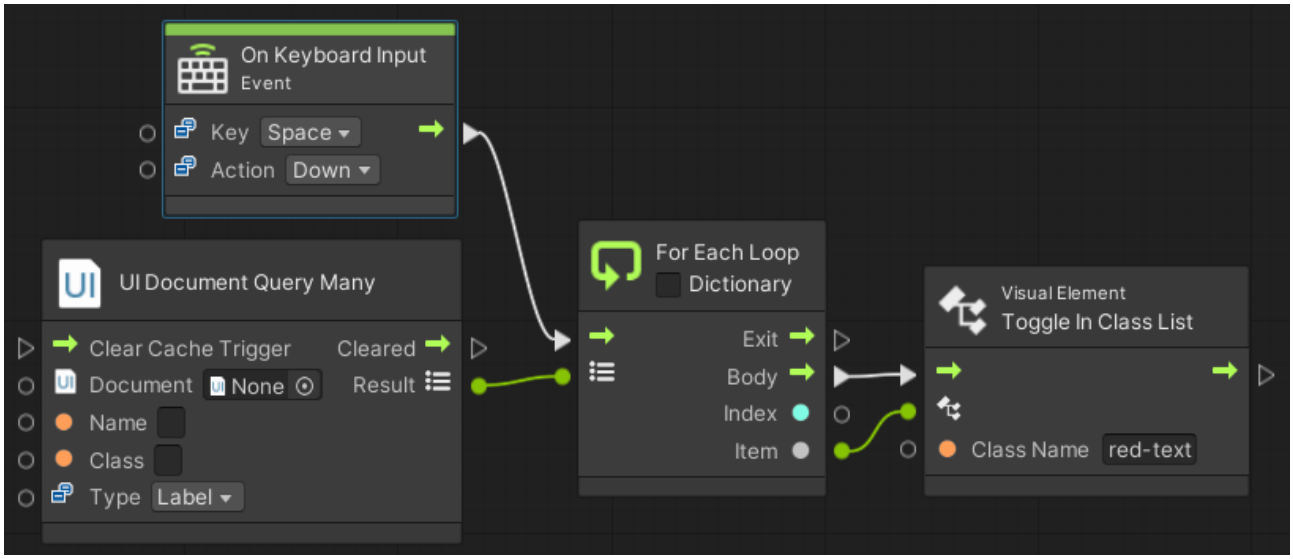
Document Var Name (Inspector): If no document is specified then the node will try to find one in the local- or scene variables. To do that it uses this name as the variable name.

Outputs:

Cleared Trigger: Called automatically after the „Clear Cache Trigger" was triggered.

Result: A list of visual elements (List<VisualElement>). Use a for each loop the extract them.

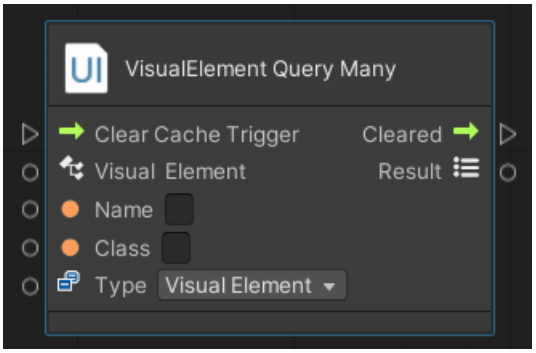
Usage:



This one toggles a class „red-text" on all labels in the document.

Visual Element Query Many

It's basically the same as UI Document Query Many only that the starting element is not a UI Document but another Visual Element.



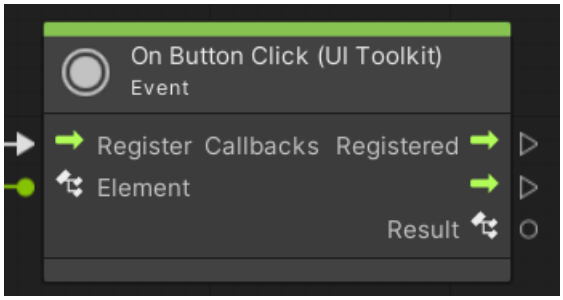
Please read the „UI Document Query Many" for details on this node.

Event Nodes

Here is a list of all the event nodes for UIToolkit. Whenever possible the nodes try to use the same name as the normal UGUI event nodes to make the transition easier.

On Button Click

An event that reacts to a button click.



Inputs:

Register Callbacks: (optional, use if the Element has changed to reregister the event callbacks)

Element: Takes a Visual Element as input (usually from a query)

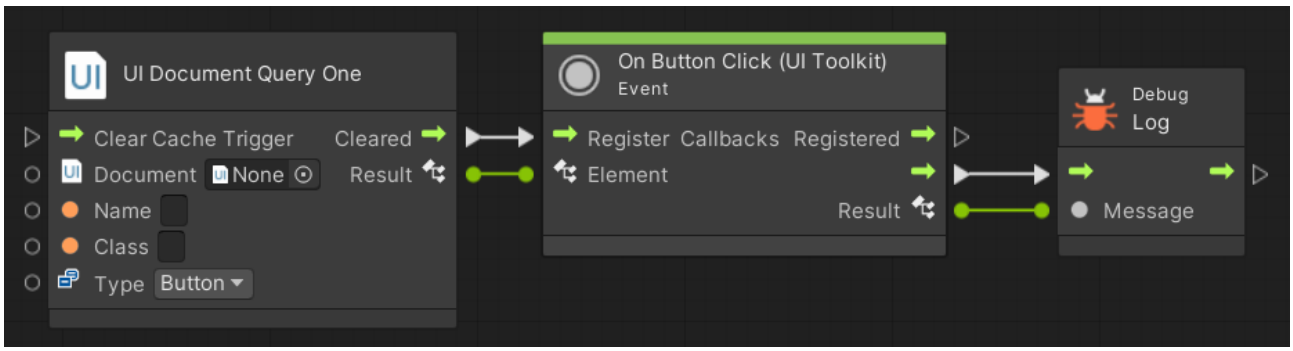
Outputs:

Registered: called automatically after the „Register Callbacks" was triggered.

Output #2 Trigger: the event trigger (usually you will use only this)

Result: a UI Toolkit ClickEvent object (see Unity Manual).

Usage:

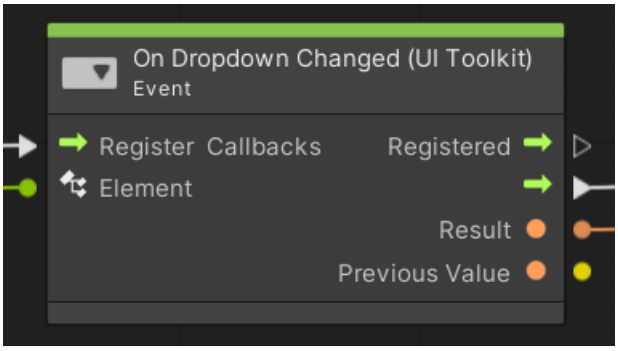


Notice how the „Cleared" output of the query is connected to the „Update Callback". While this is not strictly necessary it is a good habit. Because of this the event will automatically re-register the event listeners on the new query results whenever the cache on the query is cleared.

On Dropdown Changed

An event for dropdown changes.

Notice: The value returned is the STRING of the options, not the index.



Inputs:

Register Callbacks: (optional) Use if the Element has changed to reregister the event callbacks.

Element: Takes a Visual Element as input (usually from a query)

Outputs:

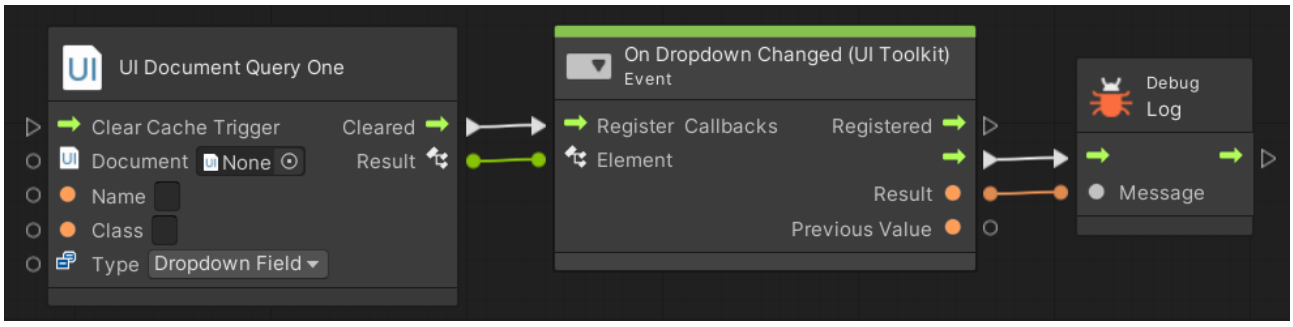
Registered: Called automatically after the „Register Callbacks" was triggered.

Output #2 Trigger: The event trigger (usually you will use only this trigger).

Result: The selected option (string)

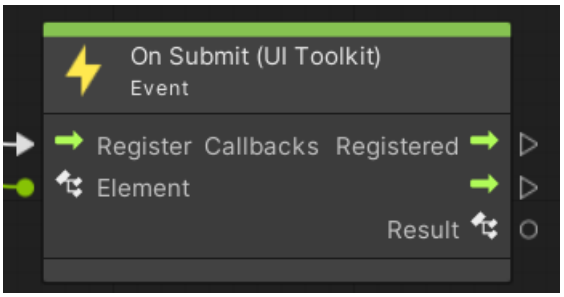
Previous Value: The previously selected option (string)

Usage:



On Submit

An event that reacts to submit and click events on any element.



Inputs:

Register Callbacks: (optional, use if the Element has changed to reregister the event callbacks)

Element: Takes a Visual Element as input (usually from a query)

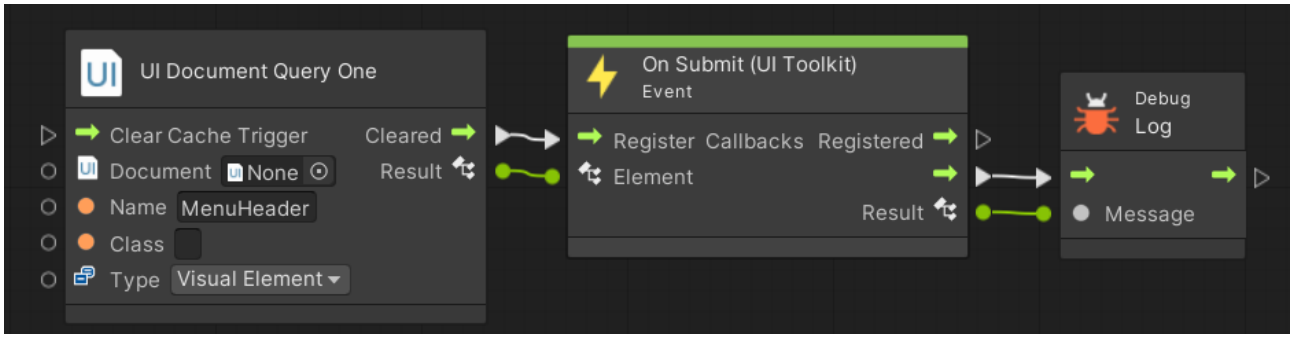
Outputs:

Registered: called automatically after the „Register Callbacks" was triggered.

Output #2 Trigger: the event trigger (usually you will use only this)

Result: a SubmitEvent object. Use its targetElement property to find out what visual element did trigger the event.

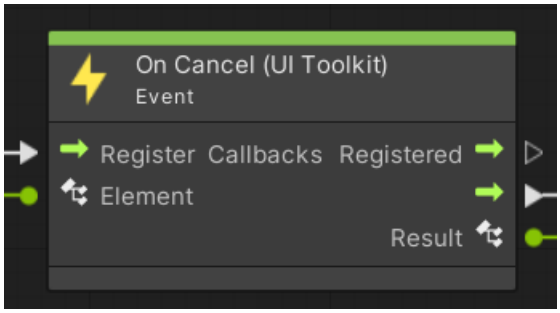
Usage:



Notice how the „Cleared" output of the query is connected to the „Update Callback". While this is not strictly necessary it is a good habit. Because of this the event will automatically re-register the event listeners on the new query results whenever the cache on the query is cleared.

On Cancel

This event is fired if an action is cancelled. For example if a text input field is exited with Esc.



Inputs:

Register Callbacks: (optional) Use if the Element has changed to reregister the event callbacks.

Element: Takes a Visual Element as input (usually from a query)

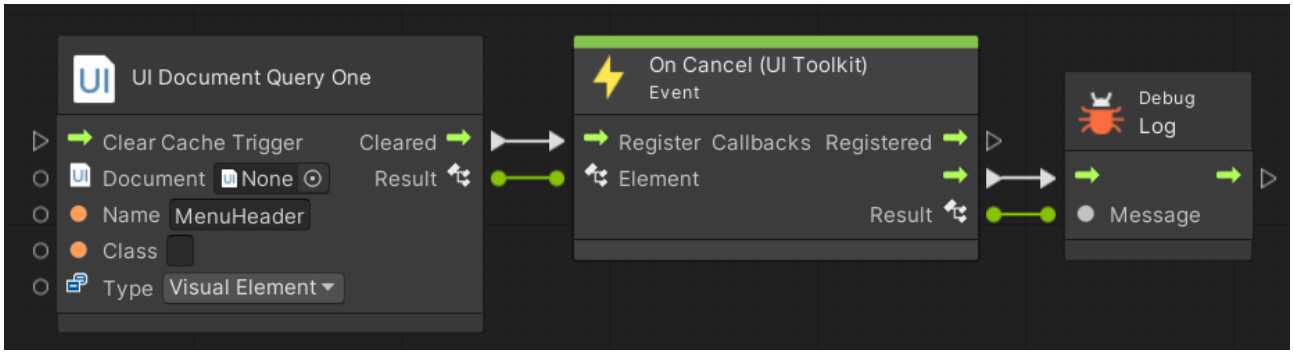
Outputs:

Registered: Called automatically after the „Register Callbacks“ was triggered.

Output #2 Trigger: The event trigger (usually you will use only this trigger).

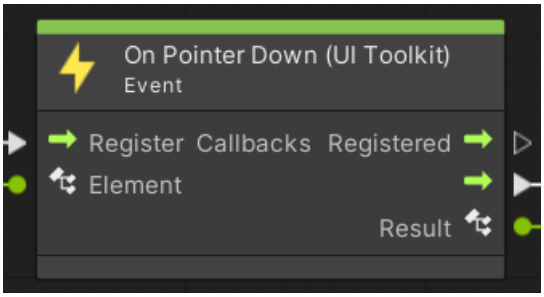
Result: The target (VisualElement) that triggered this event. It is identical with the Element input.

Usage:



On Pointer Down

Triggered when the pointer is pressed inside the visual element.



Inputs:

Register Callbacks: (optional) Use if the Element has changed to reregister the event callbacks.

Element: Takes a Visual Element as input (usually from a query)

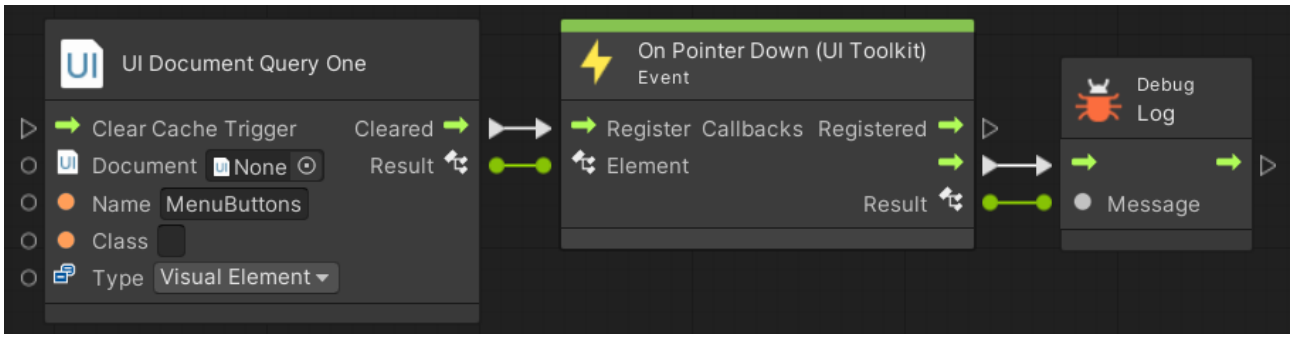
Outputs:

Registered: Called automatically after the „Register Callbacks“ was triggered.

Output #2 Trigger: The event trigger (usually you will use only this trigger).

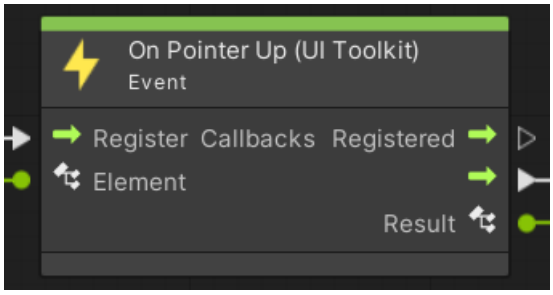
Result: The new value (boolen)

Usage:



On Pointer Up

Triggered when the pointer is released inside the visual element.



Inputs:

Register Callbacks: (optional) Use if the Element has changed to reregister the event callbacks.

Element: Takes a Visual Element as input (usually from a query)

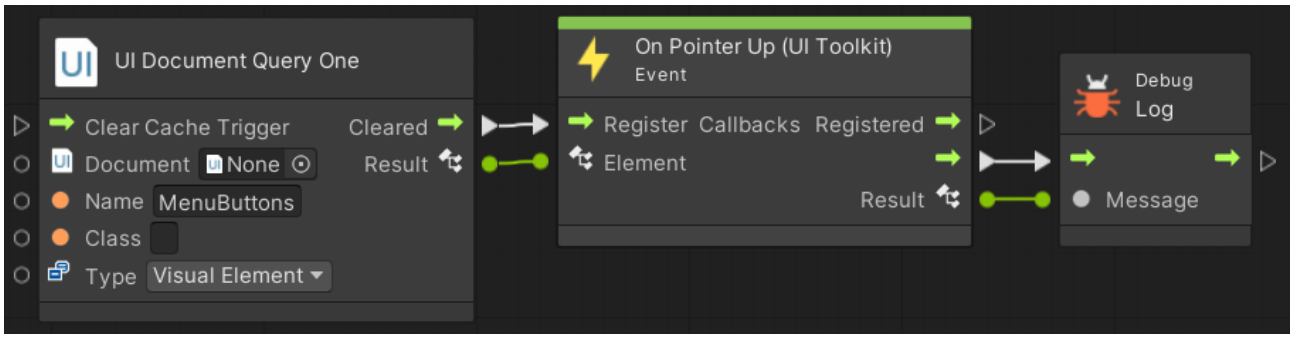
Outputs:

Registered: Called automatically after the „Register Callbacks“ was triggered.

Output #2 Trigger: The event trigger (usually you will use only this trigger).

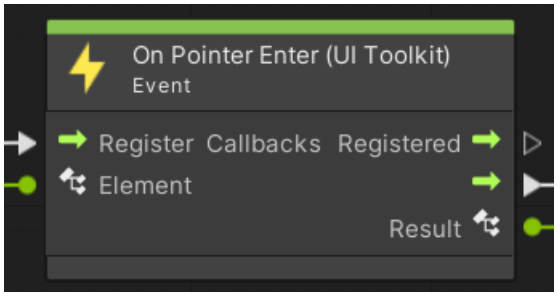
Result: The new value (boolen)

Usage:



On Pointer Enter

Triggered when the pointer enters the visual element.



Inputs:

Register Callbacks: (optional) Use if the Element has changed to reregister the event callbacks.

Element: Takes a Visual Element as input (usually from a query)

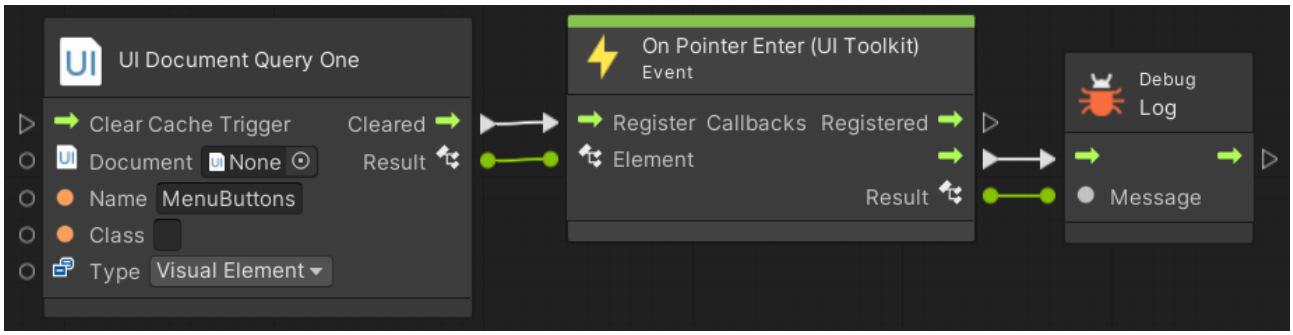
Outputs:

Registered: Called automatically after the „Register Callbacks“ was triggered.

Output #2 Trigger: The event trigger (usually you will use only this trigger).

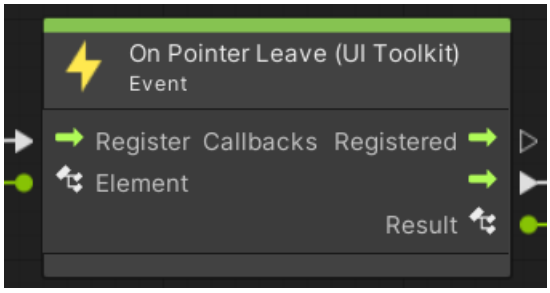
Result: The new value (boolen)

Usage:



On Pointer Leave

Triggered when the pointer leaves the visual element.



Inputs:

Register Callbacks: (optional) Use if the Element has changed to reregister the event callbacks.

Element: Takes a Visual Element as input (usually from a query)

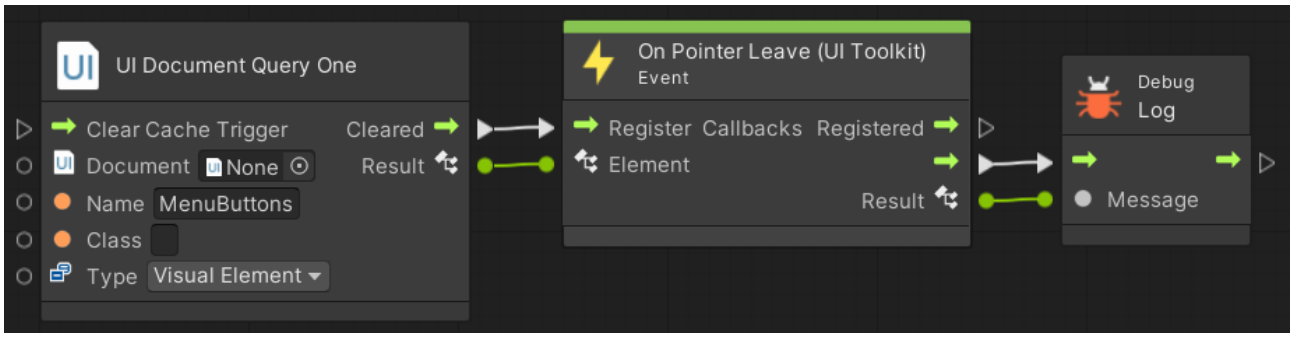
Outputs:

Registered: Called automatically after the „Register Callbacks“ was triggered.

Output #2 Trigger: The event trigger (usually you will use only this trigger).

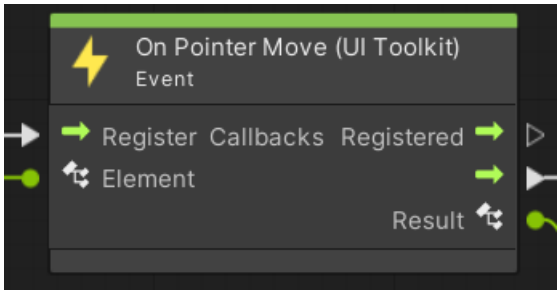
Result: The new value (boolen)

Usage:



On Pointer Move

Triggered when the pointer is moved inside the visual element.



Inputs:

Register Callbacks: (optional) Use if the Element has changed to reregister the event callbacks.

Element: Takes a Visual Element as input (usually from a query)

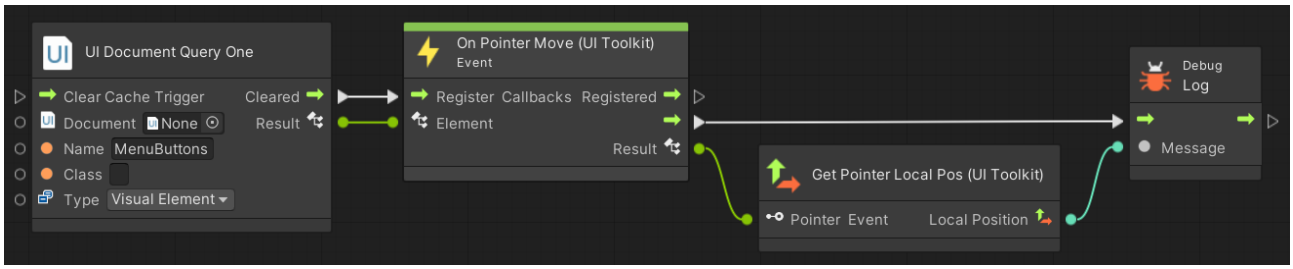
Outputs:

Registered: Called automatically after the „Register Callbacks“ was triggered.

Output #2 Trigger: The event trigger (usually you will use only this trigger).

Result: The new value (boolen)

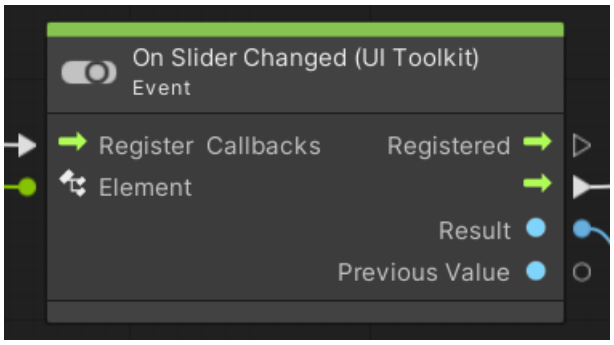
Usage:



On Slider Value Changed

Events for sliders. Result is a float.

Notice: IntSliders and MinMaxSlider have their own dedicated nodes.



Inputs:

Register Callbacks: (optional) Use if the Element has changed to reregister the event callbacks.

Element: Takes a Visual Element as input (usually from a query)

Outputs:

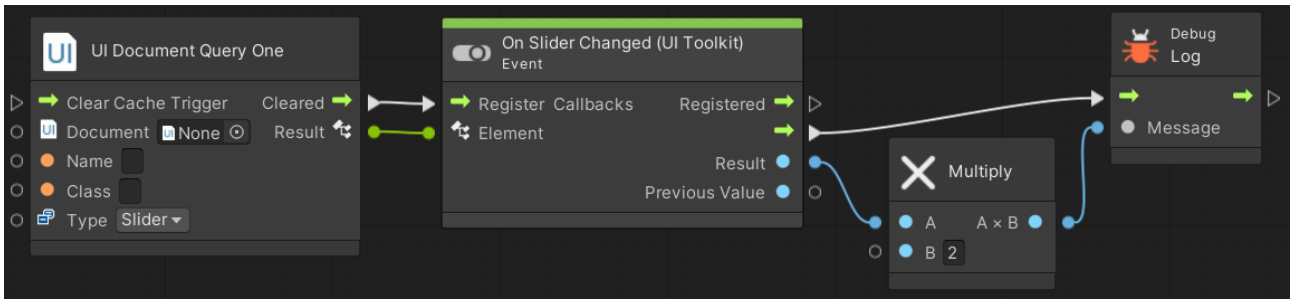
Registered: Called automatically after the „Register Callbacks" was triggered.

Output #2 Trigger: The event trigger (usually you will use only this).

Result: The new float value of the slider.

Previous Value: The old float value of the slider.

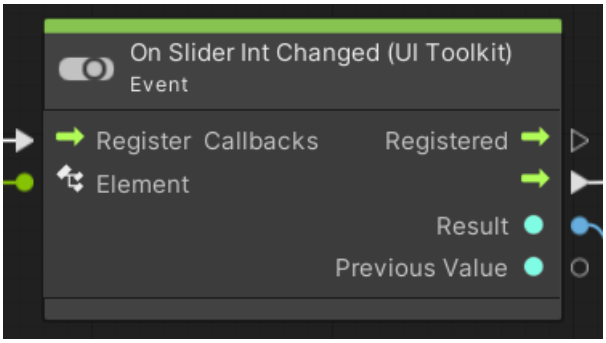
Usage:



On Slider Int Value Changed

Events for sliders. Result is an integer.

Notice: Sliders and MinMaxSlider have their own dedicated nodes.



Inputs:

Register Callbacks: (optional) Use if the Element has changed to reregister the event callbacks.

Element: Takes a Visual Element as input (usually from a query)

Outputs:

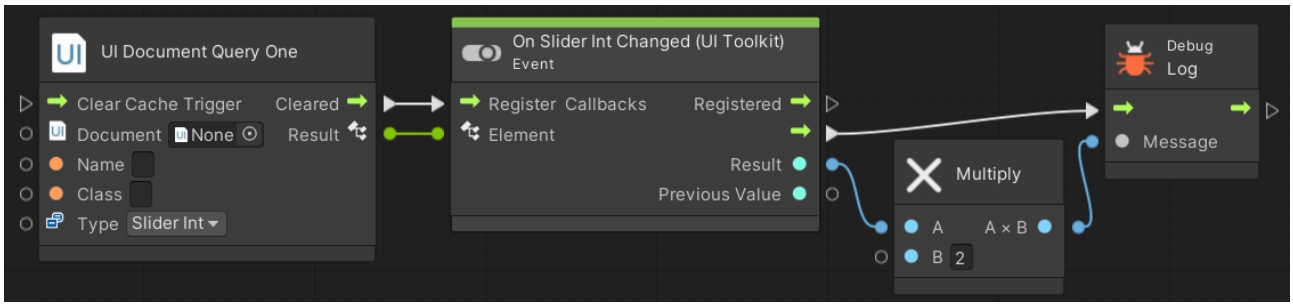
Registered: Called automatically after the „Register Callbacks" was triggered.

Output #2 Trigger: The event trigger (usually you will use only this).

Result: The new int value of the slider.

Previous Value: The old int value of the slider.

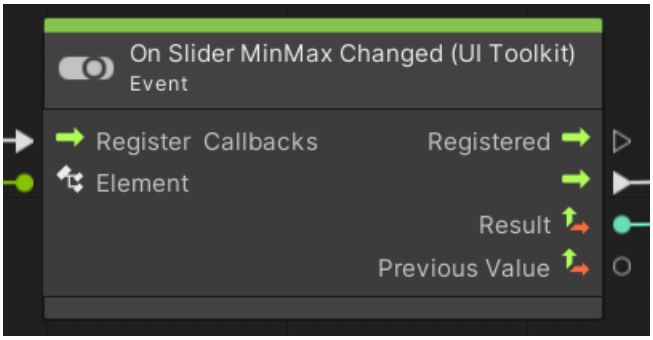
Usage:



On Slider MinMax Value Changed

Events for sliders. Result is an Vector2.

Notice: Sliders and IntSliders have their own dedicated nodes.



Inputs:

Register Callbacks: (optional) Use if the Element has changed to reregister the event callbacks.

Element: Takes a Visual Element as input (usually from a query)

Outputs:

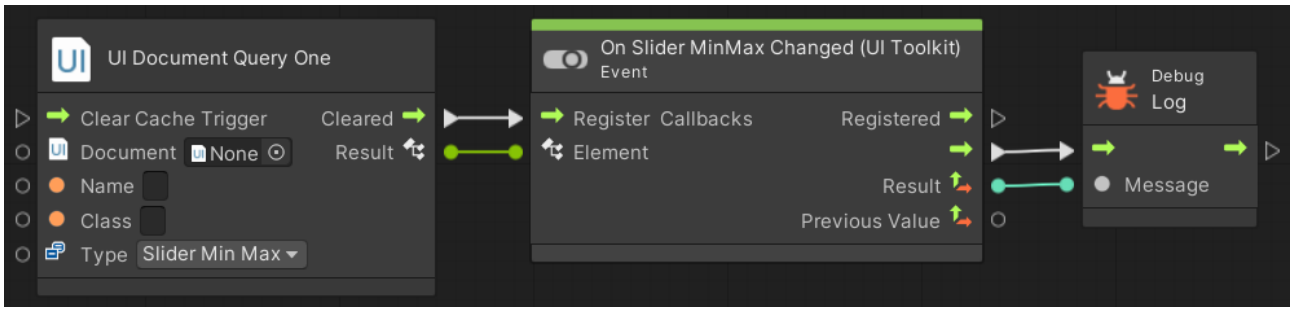
Registered: Called automatically after the „Register Callbacks" was triggered.

Output #2 Trigger: The event trigger (usually you will use only this).

Result: The new Vector2 value of the slider.

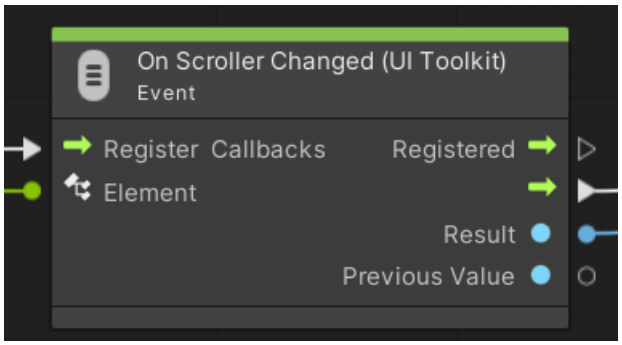
Previous Value: The old Vector2 value of the slider.

Usage:



On Scroller Value Changed

Events for scrollbars (in UI Toolkit they are called scrollers).



Inputs:

Register Callbacks: (optional) Use if the Element has changed to reregister the event callbacks.

Element: Takes a Visual Element as input (usually from a query)

Outputs:

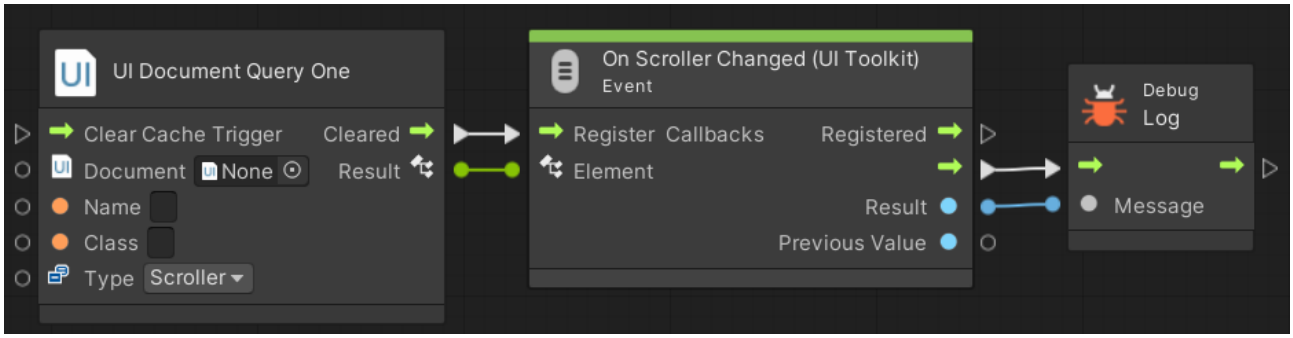
Registered: Called automatically after the „Register Callbacks“ was triggered.

Output #2 Trigger: The event trigger (usually you will use only this trigger).

Result: The new value (float).
HINT: ScrollViews consist of two Scrollers. In a ScrollView the values are the pixels the scroller is scrolling the CONTENT, starting with 0.

Previous Value: The old value (float)

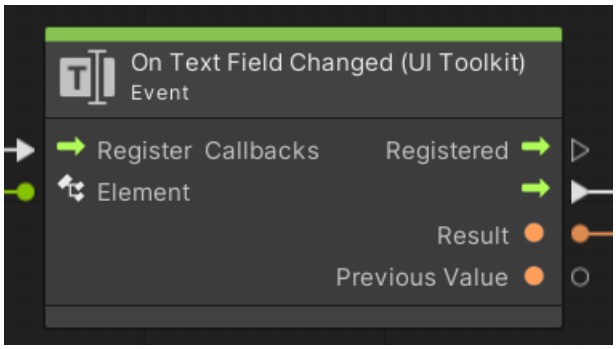
Usage:



On Text Field Value Changed

Events for changes in a text field.

Notice: Check the „isDelayed“ box on the textfield or use the OnTextFieldEditEnd event if you do not want this to be triggered with every key press.



Inputs:

Register Callbacks: (optional) Use if the Element has changed to reregister the event callbacks.

Element: Takes a Visual Element as input (usually from a query)

Outputs:

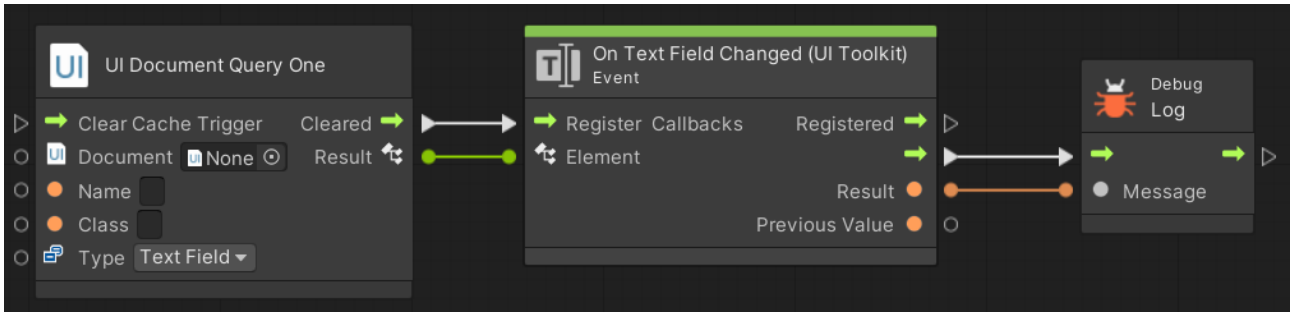
Registered: Called automatically after the „Register Callbacks“ was triggered.

Output #2 Trigger: The event trigger (usually you will use only this trigger).

Result: The new text (string)

Previous Value: The old text (string)

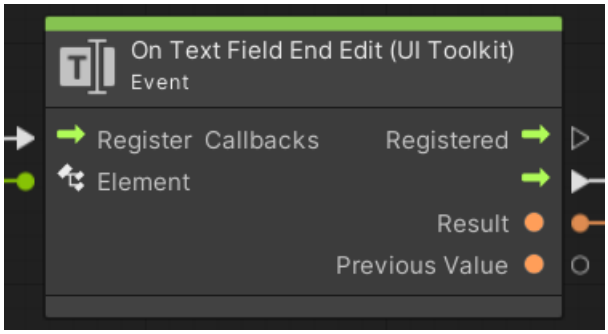
Usage:



On Text Field End Edit

Events for when the editing of a text field ended.

Notice: Use the OnTextFieldChanged event if you need an event for every key press.



Inputs:

Register Callbacks: (optional) Use if the Element has changed to reregister the event callbacks.

Element: Takes a Visual Element as input (usually from a query)

Outputs:

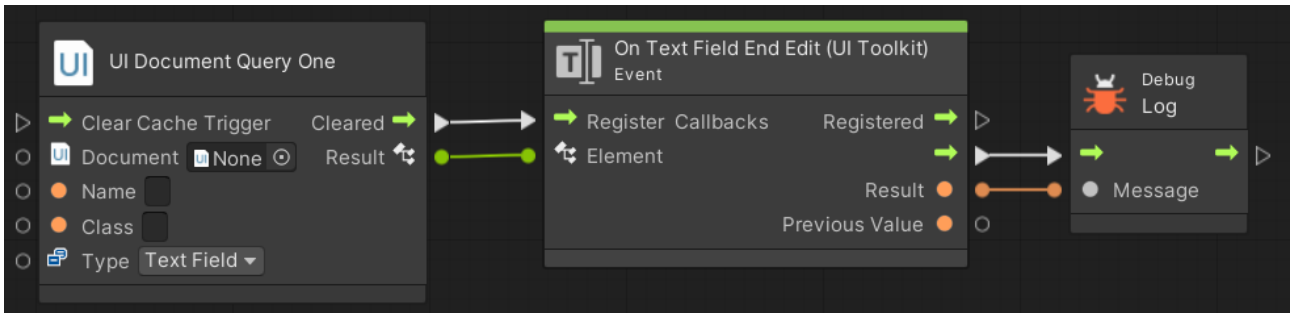
Registered: Called automatically after the „Register Callbacks" was triggered.

Output #2 Trigger: The event trigger (usually you will use only this trigger).

Result: The new text (string)

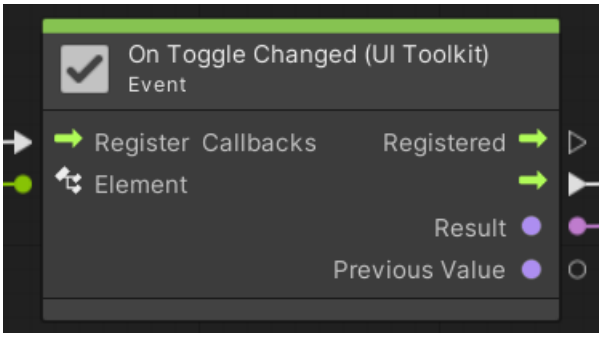
Previous Value: The old text (string)

Usage:



On Toggle Value Changed

Events for toggles.



Inputs:

Register Callbacks: (optional) Use if the Element has changed to reregister the event callbacks.

Element: Takes a Visual Element as input (usually from a query)

Outputs:

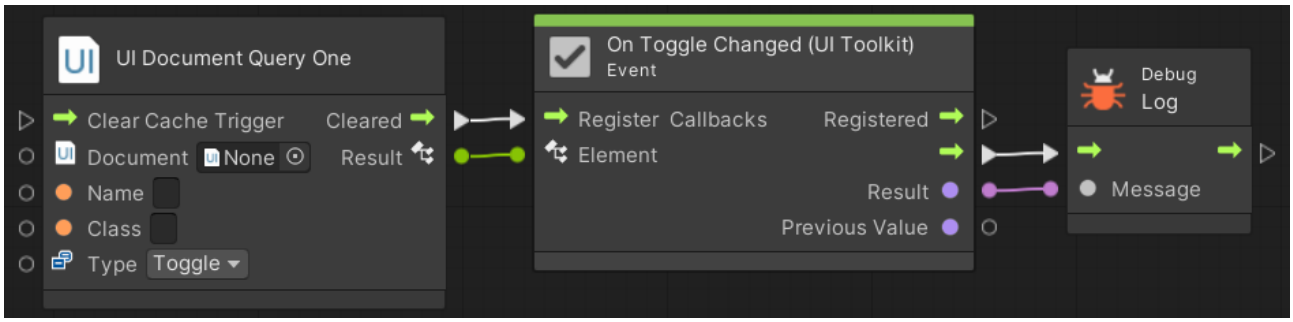
Registered: Called automatically after the „Register Callbacks“ was triggered.

Output #2 Trigger: The event trigger (usually you will use only this trigger).

Result: The new value (boolen)

Previous Value: The old value (boolean)

Usage:

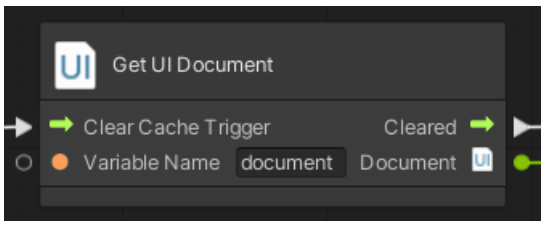


Helper Nodes

Here are some handy nodes to deal with events and other data types.

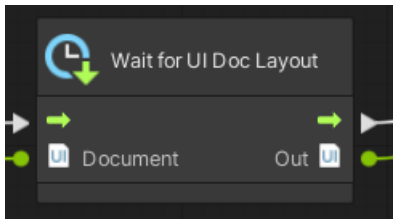
Get UI Document

Useful if you need to fetch th UI document from the gameObject or a variable (local/scene).



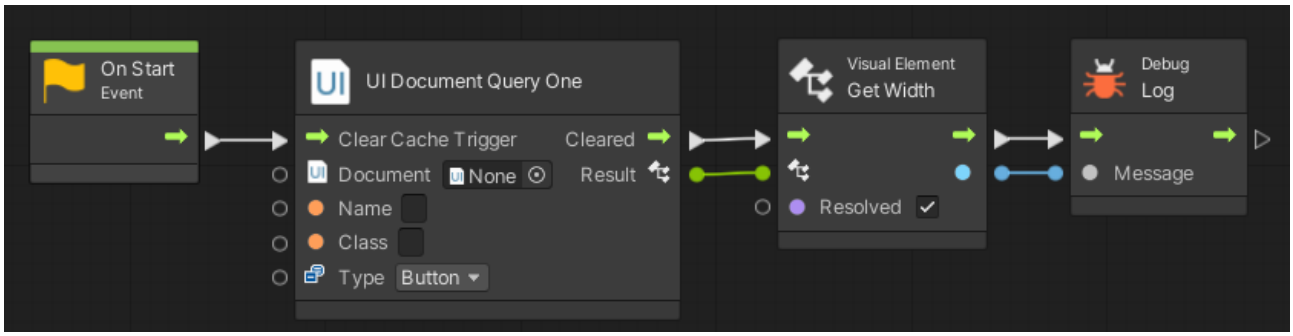
The „Variable Name“ is the name of the local or scene variable (whichever is found first is used). For more details on how it works please refer to the „UI Document Query“ nodes. The logic is the same.

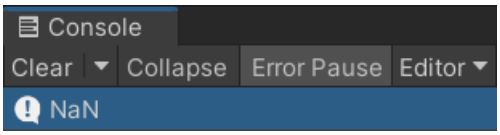
Wait For UI Document Layout



Once a UI Document is loaded it will need some time to do the layouting (usually done within the first frame after activation). This means that the dimensions may return NaN as value if you try to read them in the very first frame. This is a Unity limitation. UI Elements are loaded and layouted via the normal event message queue in Start().

So, if you do this (see below) you will get a NaN response:

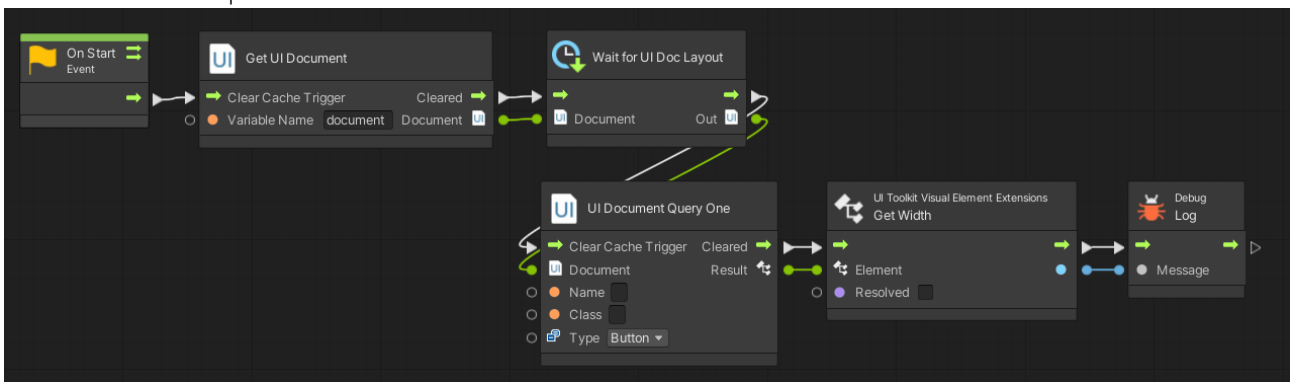




NOTICE: This is only relevant if you want to READ these values in the first frame. You can set them here without any problems.

The solution is to wait for the layouting to be done. The „Wait For UI Document Layout“ does exactly that.

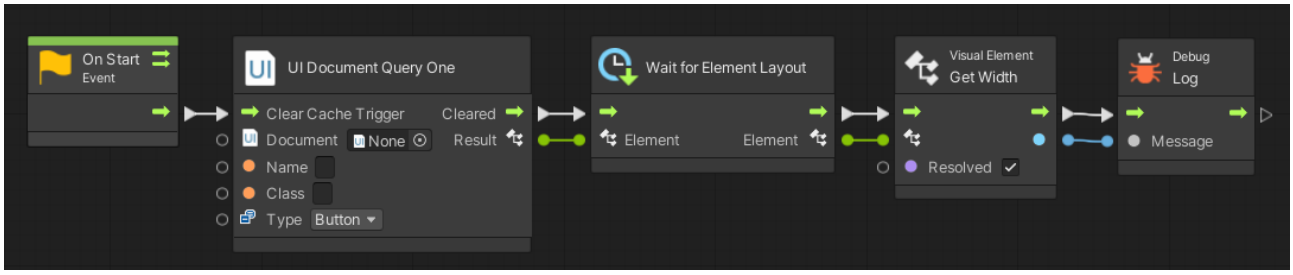
Here is the example from above with the wait node added:



This looks a bit cumbersome (get the document, feed it into the wait node, then into the query). An alternative to this is to use a „Wait for Element Layout“ node. It works the same, only that it takes an element instead of a document as input.

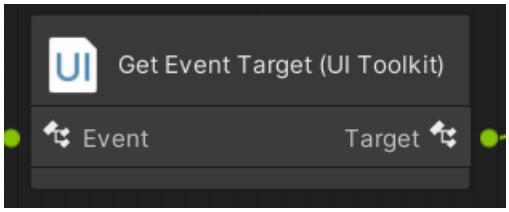
Wait For UI Element Layout

It does the same as the „Wait for UI Document Layout“ node (see above).



Event Target

Useful for extracting the target (visual element) from an event.



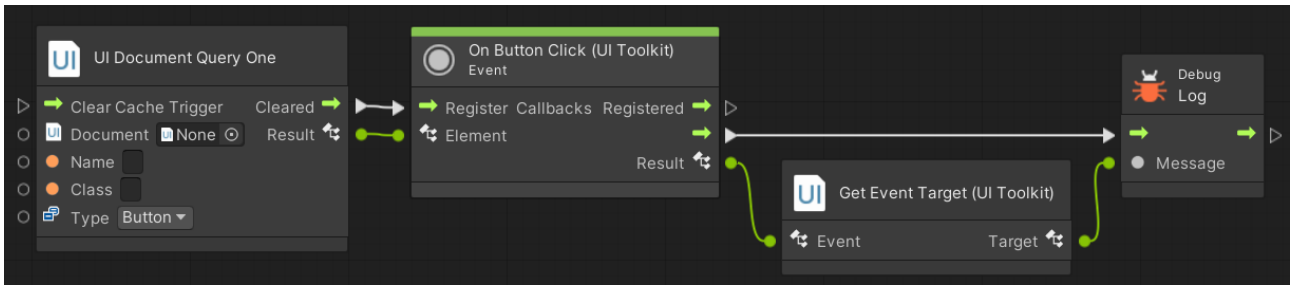
Inputs:

Event: The event object

Outputs:

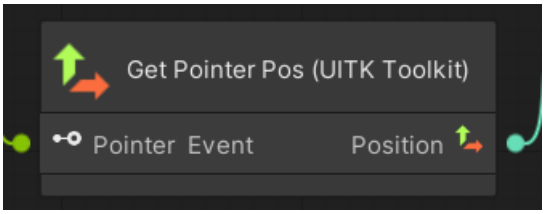
Target: The visual element stored in the EventBase.target property.

Usage:



Get Pointer Event Pos

Useful for extracting the global position of the pointer. Can be used on pointer events.



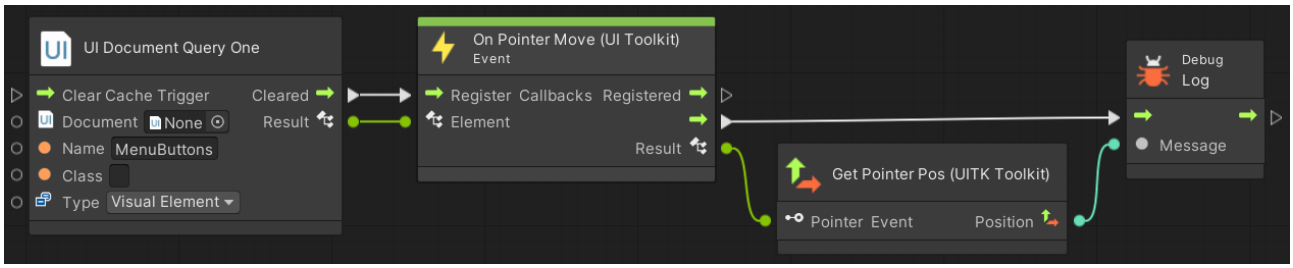
Inputs:

Event: The event object

Outputs:

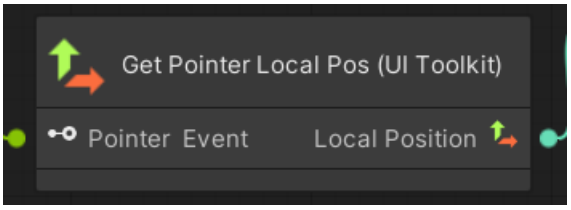
Position: The global pointer position.

Usage:



Get Local Pointer Event Pos

Useful for extracting the local position of the pointer within the visual element. Can be used on pointer events.



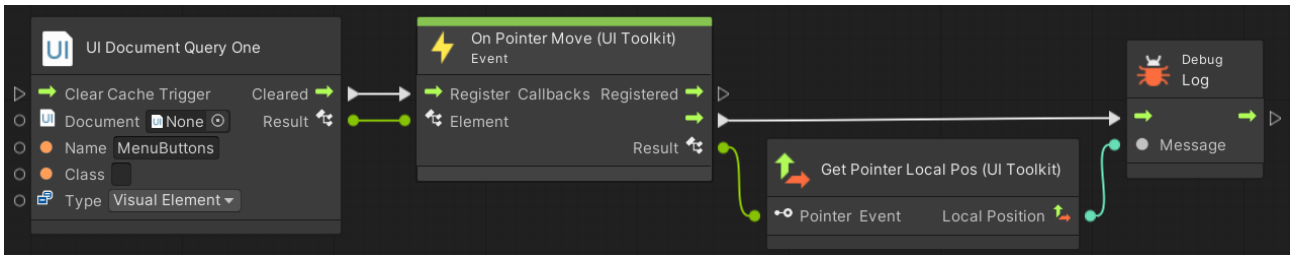
Inputs:

Event: The event object

Outputs:

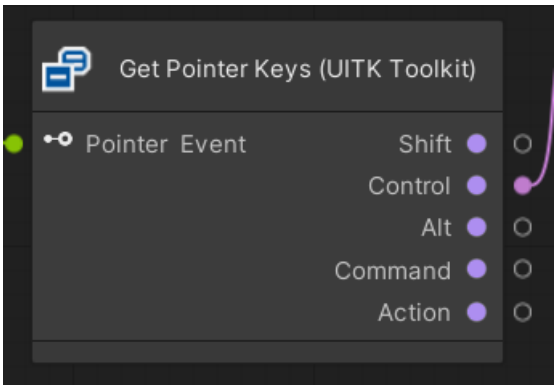
Local Position: The local pointer position.

Usage:



Get Pointer Keys

Useful to check if a key was pressed when the event fired.



Inputs:

Pointer Event: The pointer event object

Outputs:

Shift: Was the [Shift] key pressed (bool)

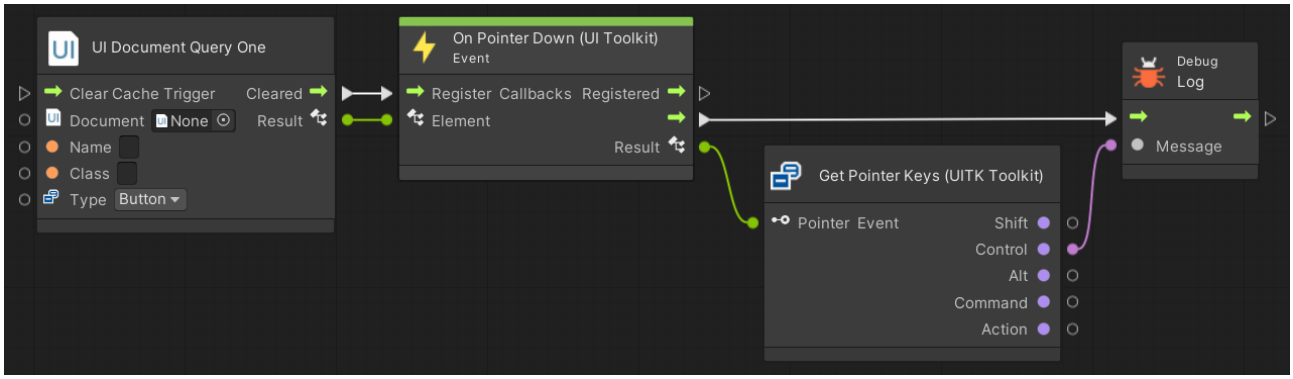
Control: Was the [Control] key pressed (bool)

Alt: Was the [Alt] key pressed (bool)

Command: Was the [Command] key pressed (bool)

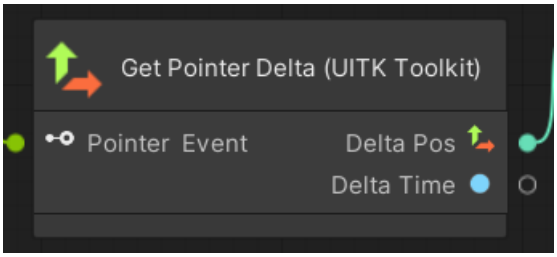
Action: Was the [Action] key pressed (bool)

Usage:



Get Pointer Delta

Gets the delta from the last pointer move.



Inputs:

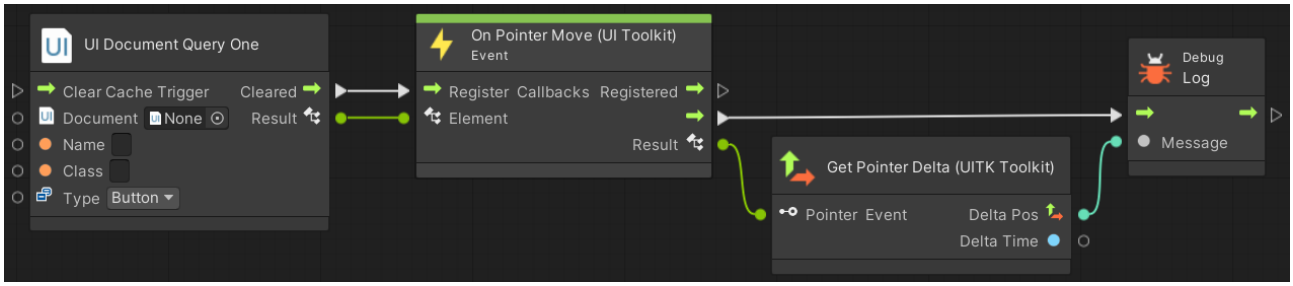
Pointer Event: The pointer event object

Outputs:

Delta Pos: The position delta

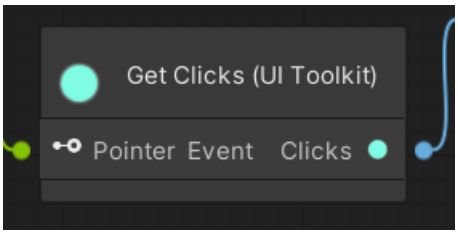
Delta Time: The time delta

Usage:



Get Pointer Clicks

Gets the number of clicks on the element. Useful for implementing double-click buttons.



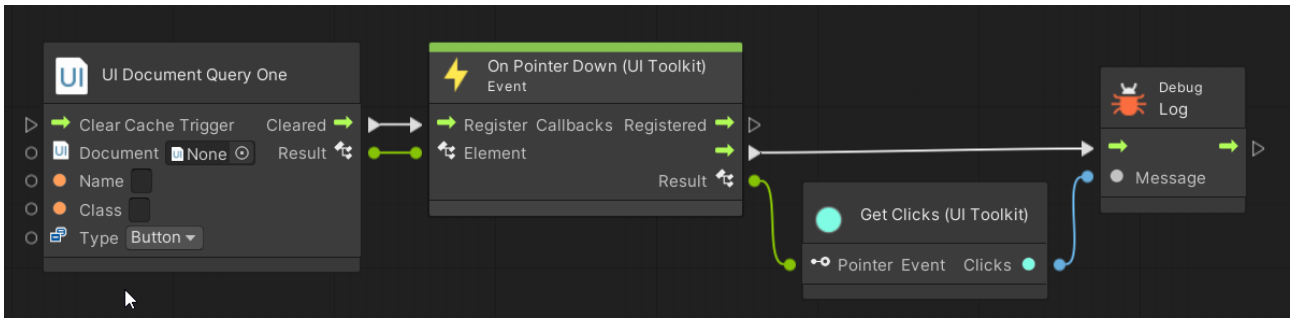
Inputs:

Pointer Event: The pointer event object

Outputs:

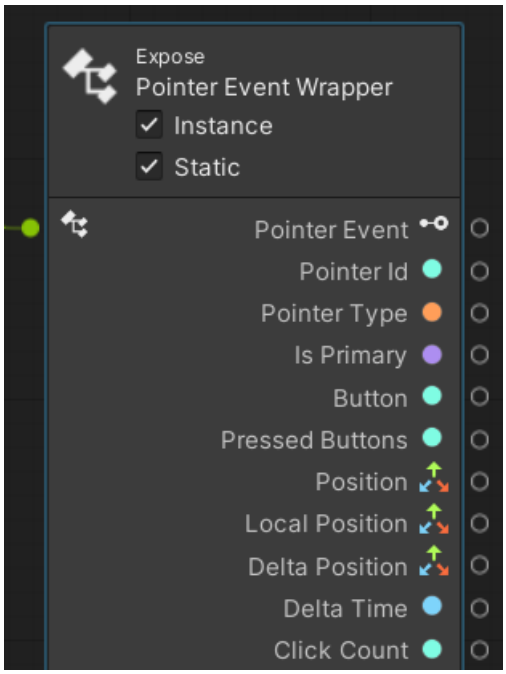
Clicks: The number of clicks counted within a small time window.

Usage:



Pointer Event Wrapper

If the „Get Pointer ..“ nodes are not enough then this will help you to access any data you want.



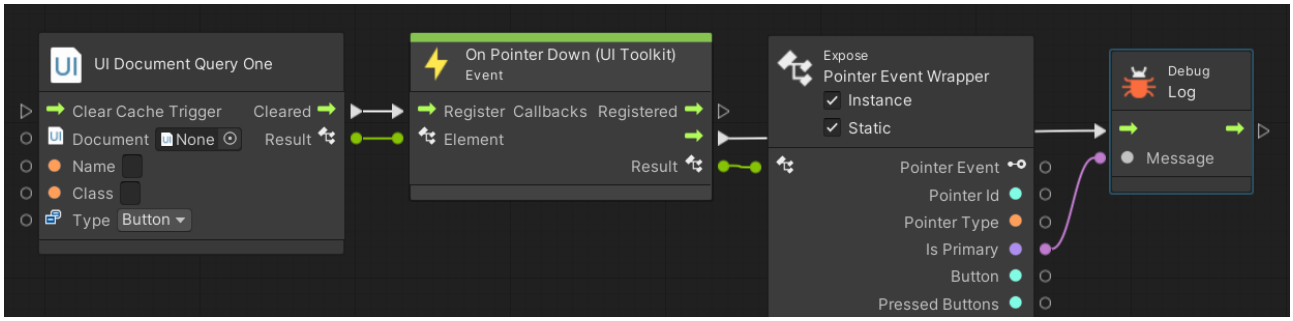
Inputs:

Pointer Event: A pointer event object

Outputs:

List: If used with „expose“ it will list all public properties of the pointer event.

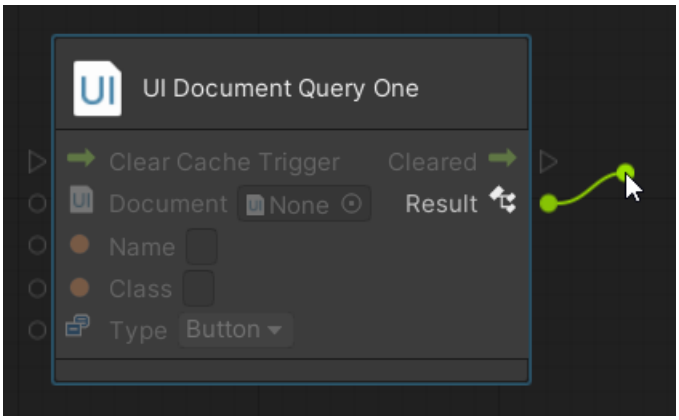
Usage:



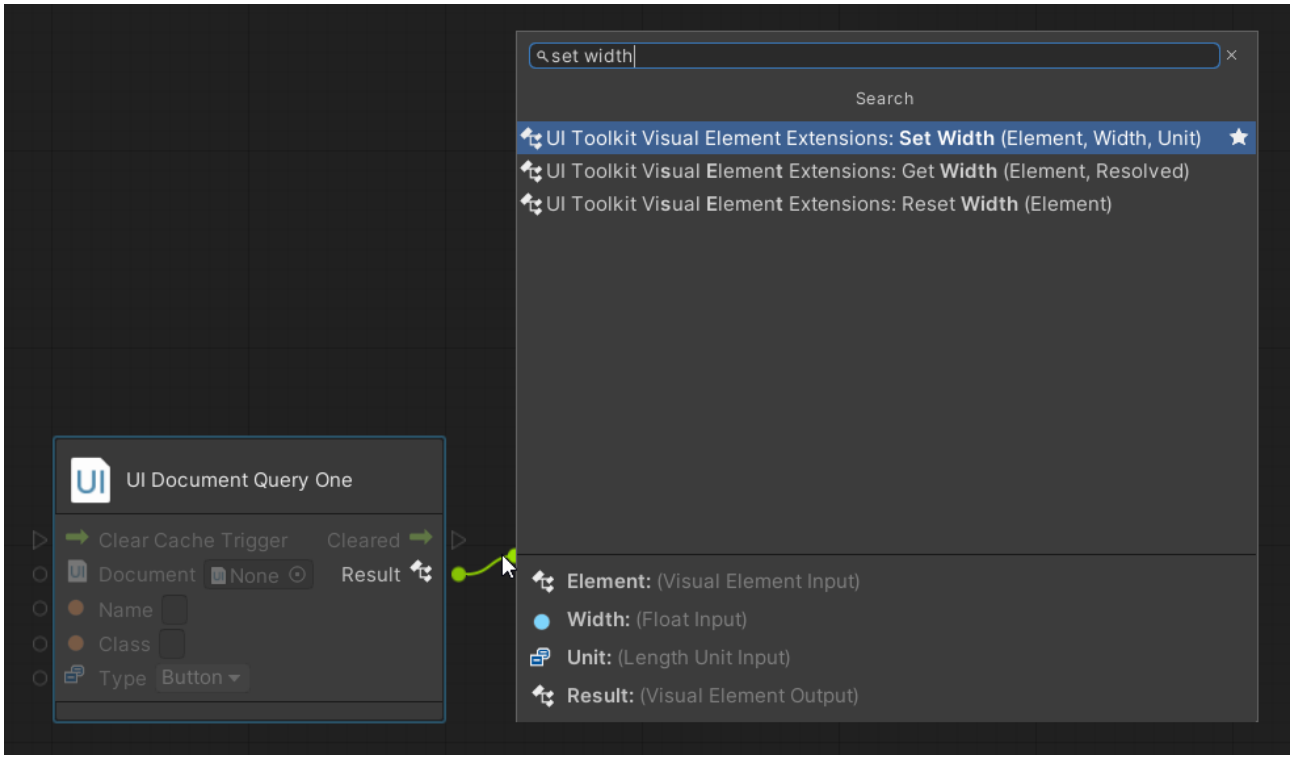
Helper Extensions (Visual Element Extension)

To access the functions of a VisualElement (the basic building block of UI Toolkit) you can either add the VisualElement class to your Visual Scripting „Type Options“ or you use the included **UIToolkitVisualElementExtensions**.

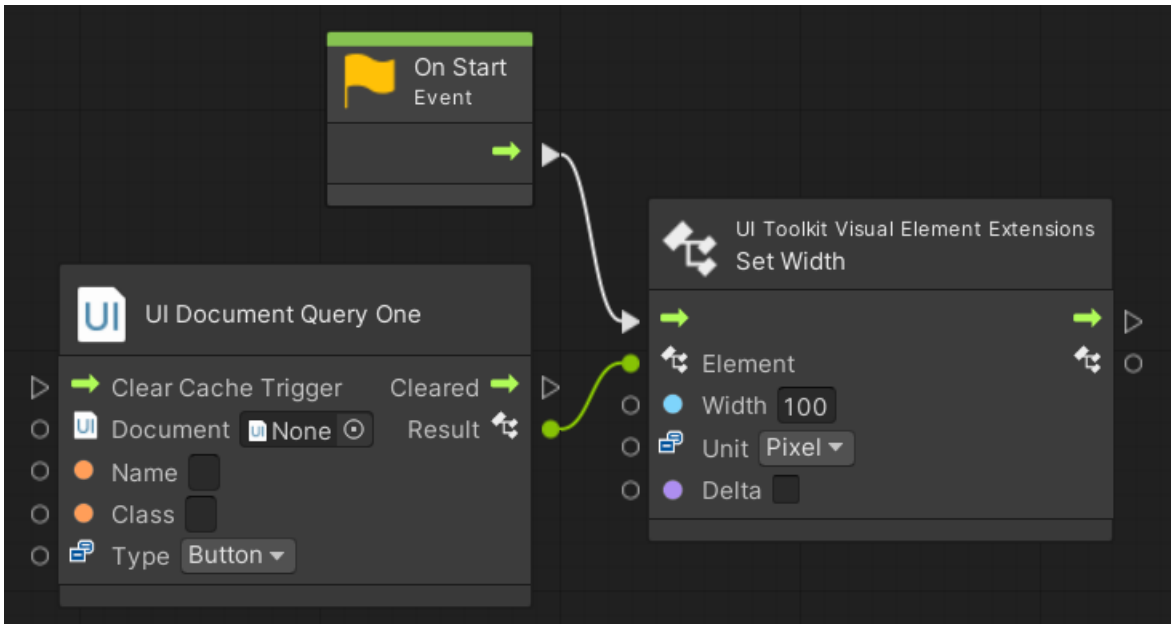
The quickest way to access them is to start dragging on the Result value of a query, like this:



If you then start typing commands like „set width“ the fuzzy finder will show you the available extension methods.



In the example below we set the width of a Button to 100 px at the start.



Inputs:

Element: The visual element on which to operate.

Width: The value of the operation

Unit: The value unit (pixel, percent, ..)

Delta: If enabled then the value (width) will be added to the current width instead of replacing it. The „delta" concept is used in many extensions.

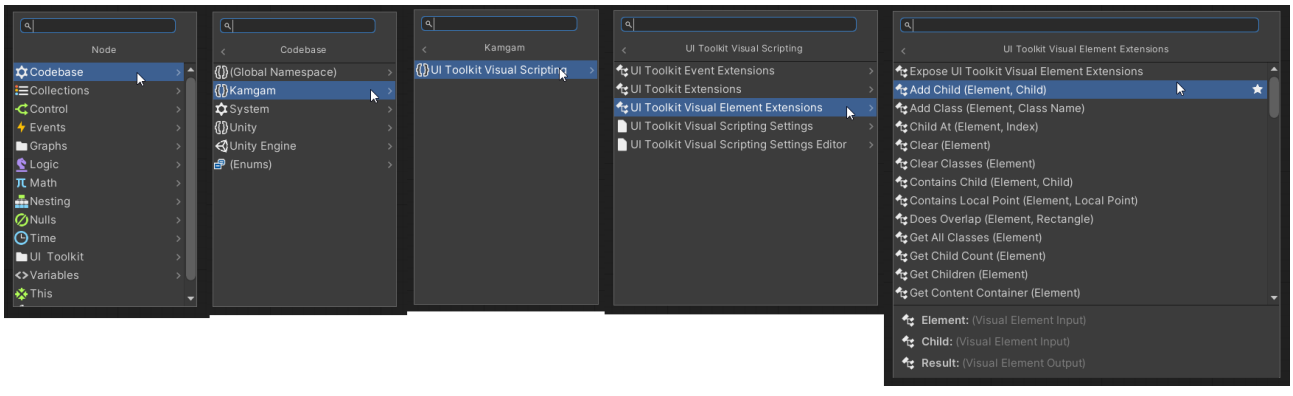
Outputs:

Out Trigger: Simply forwards the input trigger.

Result: Forwards the „Element" input to the output. Useful for chaining multiple nodes.

The are many more extensions like this.

If you want to explore then go to: **Codebase** > **Kamgam** > **UI Toolkit Visual Scripting** > **UI Toolkit Visual Element Extensions** and you will get a list of all the available methods.



Frequently Asked Questions

Here are some common issues that have been reported.

If you can, please upgrade to the highest LTS version of Unity. The newer the version the less „glitches" the UI Toolkit has.

Keep in mind, UI Toolkit as a whole it is still a work in progress and not quite ready for prime time. Unity itself still recommends using UGUI instead of UI Toolkit for runtime applications (source).

What about Drag and Drop Events

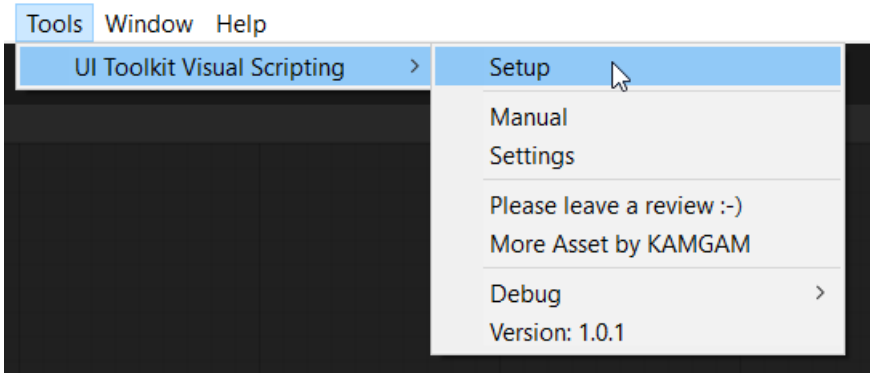
You may wonder why the drag and drop events have no nodes (DragEnterEvent, DragLeaveEvent, DragExitedEvent, DragPerformEvent, DragUpdatedEvent).

The answer is that these are not supported at runtime.
Source: https://forum.unity.com/threads/dragexitedevent-does-not-exist-in-the-namespace-unityengine-uielements.1430083/

To implement drag and drop at runtime Unity recommends using pointer capturing. Here is the page explaining it in the Unity Manual: https://docs.unity3d.com/Manual/UIE-create-drag-and-drop-ui.html

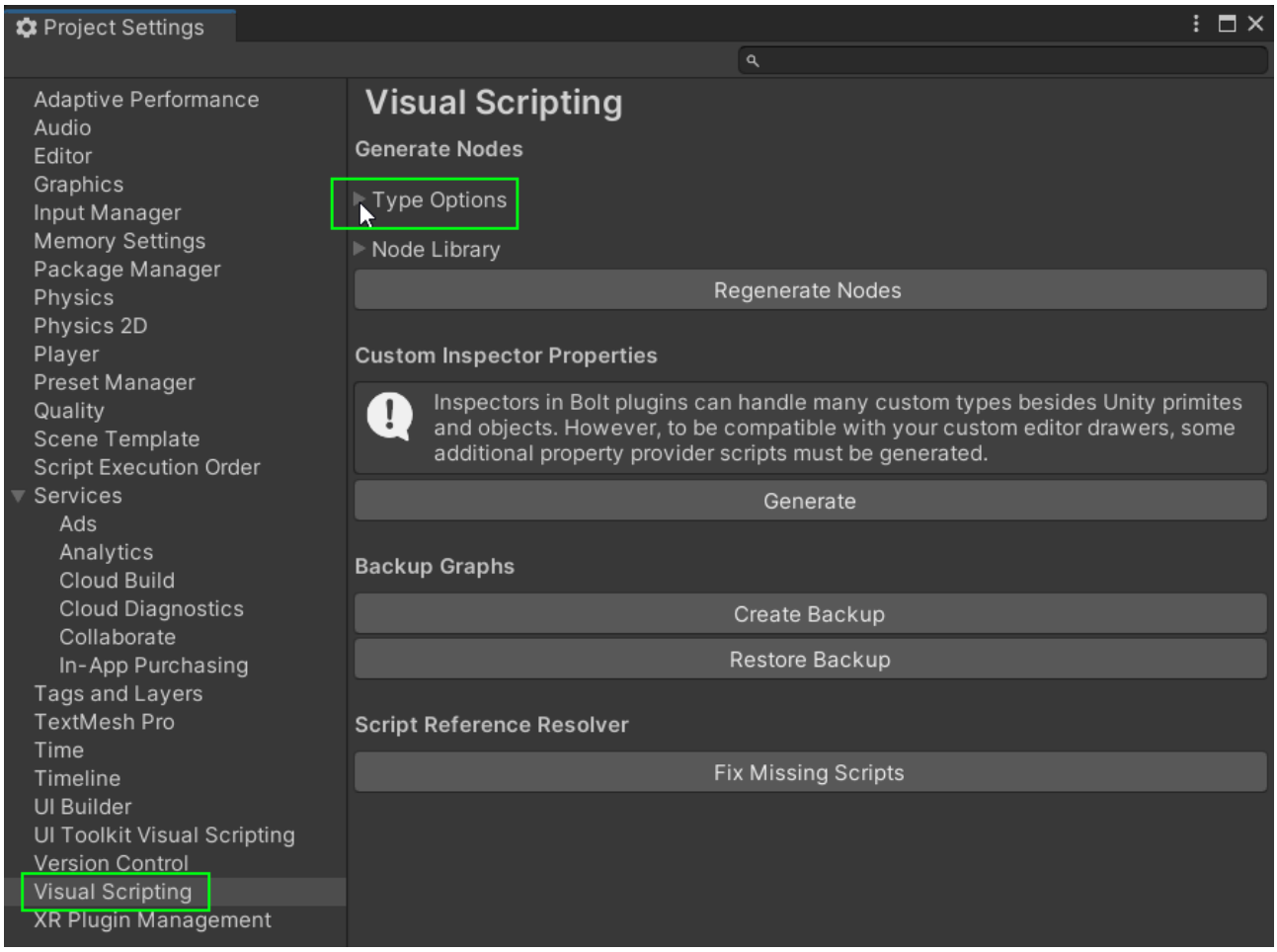
There is no „UI Toolkit" category in the Fuzzy Finder

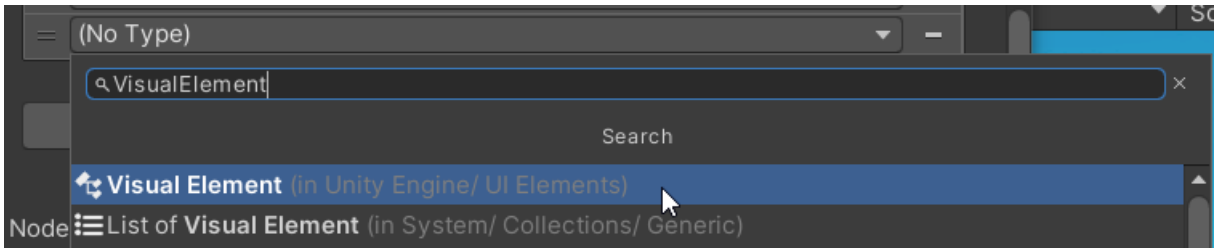
This may happen if the visual scripting has not yet been used in the project before. You will have to re-run the automatic setup again via Tools > UI Toolkit Visual Scripting > Setup OR you follow the Manual Setup instructions at the start of this document.



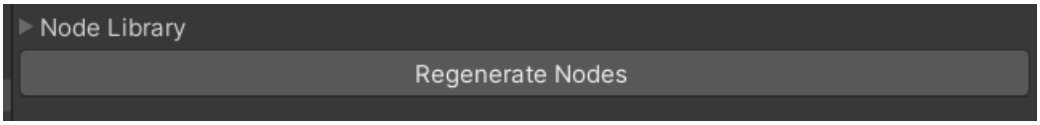
I need to access properties that are not covered by the extensions.

You can add the VisualElement to the type options. This will allow you to access any property directly.

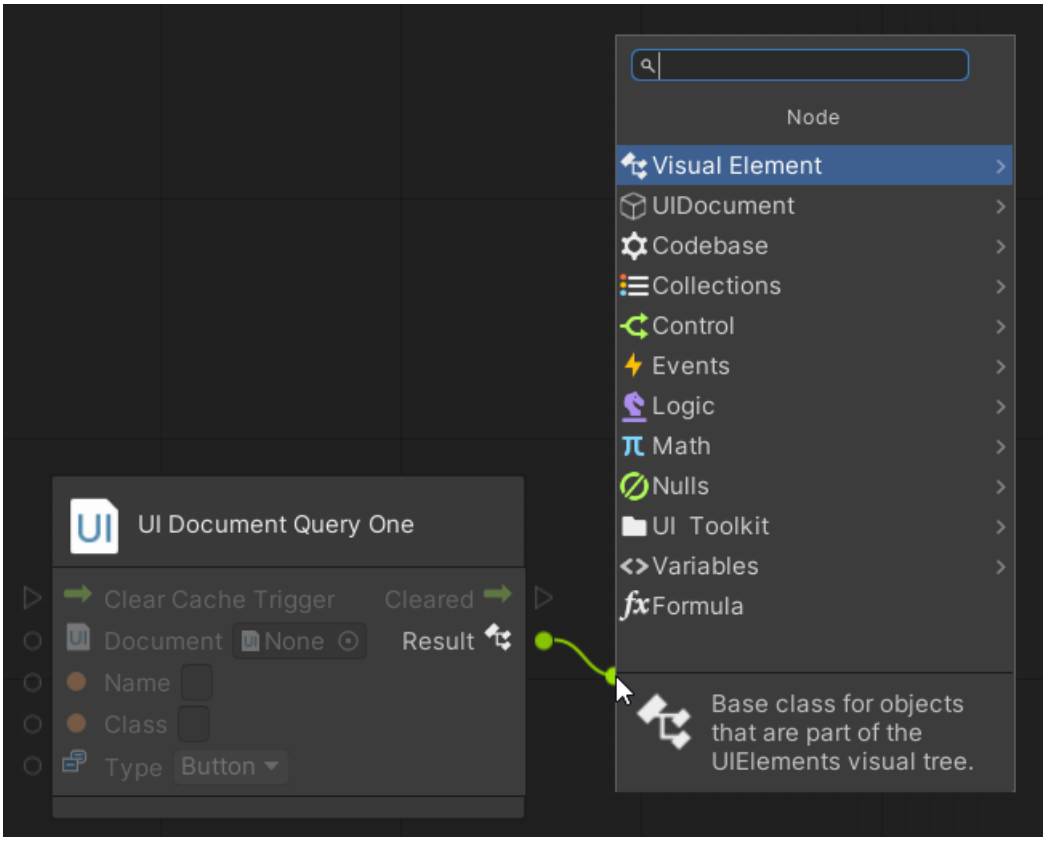




Don'T forget to regenerate the nodes once you have added the „Visual Element" class.



After that you should be able to access the VisualElement in the fuzzy finder.



How to use the „Cleared“ cache trigger output?

In case the UI in the document is changed dynamically it may happen that the old query result is no longer valid (you may have added a new button and deleted the old one for example).

In these cases you have to call the „Clear Cache Trigger“ to clear the query cache.

If another node (like the Button-Click-Node below) depends on the result of the query then it also needs to be notified of the changed query result.

That's what the „Cleared“ trigger does. It lets the Button node know that it has to re-register the onClick events on the new query result.