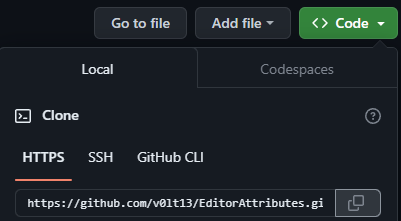
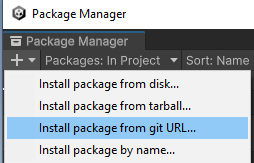
Getting Started

To begin using this package you must first import it into your unity project.

First copy the HTTPS link from the repository.



Inside your project open the package manager window click the ‘+’ button then “Install package from git URL…”.

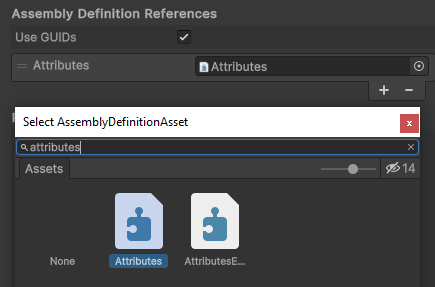


And in the text field paste the link copied from before and click install.

To start using the attributes you must add the using statement:

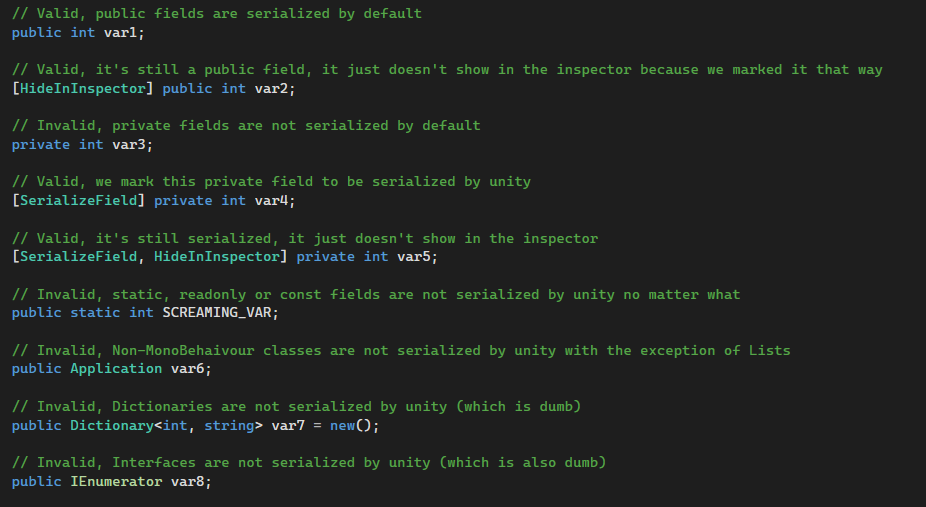


If you are using Assembly Definitions in your project make sure you reference the “Attributes” Assembly Definition where needed.



**Note:** Every field the attribute is attached to or the attribute is looking for must be a Unity serializable, meaning those fields must be either public or private with the SerializeField Attribute and valid to the unity serializer.

*Example:*



Void Struct

The Void struct is an empty struct to use as a holder property for attributes like the Button Attribute, HorizontalField Attribute, etc. that completely redo the way a property is drawn in the inspector.

While it’s not necessary to use the Void struct for those attributes (any serialized data type can be used), using the Void struct as a holder makes the code more readable.

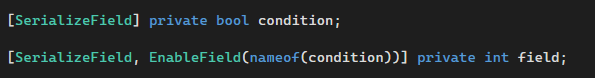
EnableField Attribute

Attribute to enable a field based on a condition.

Parameters:

* (string) conditionName -> The name of the boolean condition to evaluate

Use the “nameof()” expression to provide the name of the serialized boolean to look for; a string can also be used but is error prone.



The field will remain disabled by default until its condition becomes true.

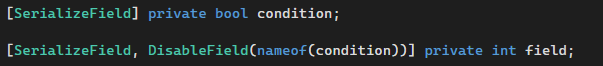
DisableField Attribute

Attribute to disable a field based on a condition.

Parameters:

* (string) conditionName -> The name of the boolean condition to evaluate

Use the “nameof()” expression to provide the name of the serialized boolean to look for; a string can also be used but is error prone.



The integer field will be enabled by default until its condition becomes true.

DisableInPlayMode Attribute

Attribute to disable a field when entering play mode.



Useful when you want to make sure no values can be modified in the inspector while you're playing the game but you can still see them.

ReadOnly Attribute

Attribute to make a field readonly in the inspector.



Useful when you want to display debug information in the inspector but you don’t want them to be accidentally edited.

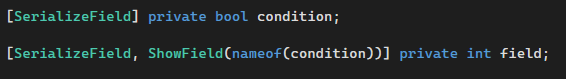
ShowField Attribute

Attribute to show fields in the inspector based on a condition.

Parameters:

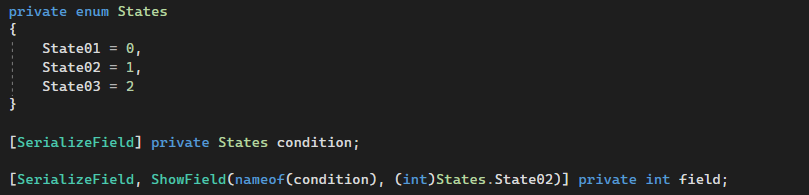
* (string) conditionName: The name of the boolean or enum condition to evaluate
* (int) enumValue: The value of the enum as an integer (if the provided condition is an enum)

Use the “nameof()” expression to provide the name of the serialized boolean to look for; a string can also be used but is error prone.



The field will be hidden by default until the condition becomes true.

You can just cast the state of your enum to an integer, you can also use integer numbers to represent your enum value but you have to make sure the number matches with the one assigned to the state.



The field will be shown when the enum is set to “State02”.

**Limitations:**

* The attribute cannot look for the condition inside a struct
* It cannot hide arrays or lists, only the fields inside them

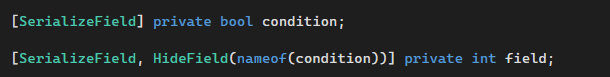
HideField Attribute

Attribute to hide fields in the inspector based on a condition.

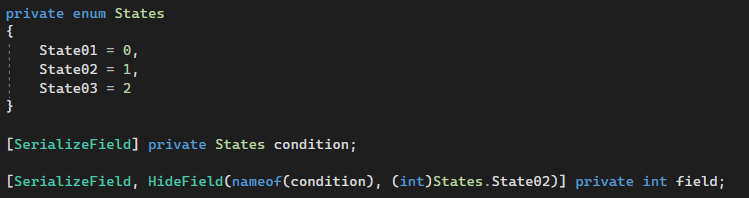
Parameters:

* (string) conditionName: The name of the boolean or enum condition to evaluate
* (int) enumValue: The value of the enum as an integer (if the provided condition is an enum)

Use the “nameof()” expression to provide the name of the serialized boolean to look for; a string can also be used but is error prone.

  
The field will be shown by default until the condition becomes true.

You can just cast the state of your enum to an integer, you can also use integer numbers to represent your enum value but you have to make sure the number matches with the one assigned to the state.



The field will be hidden when the enum is set to “State02”.

**Limitations:**

* The attribute cannot look for the condition inside a struct
* It cannot hide arrays or lists, only the fields inside them

HideInPlayMode Attribute

Attribute to hide a field when entering play mode.



**Limitations:**

* It cannot hide arrays or lists, only the fields inside them

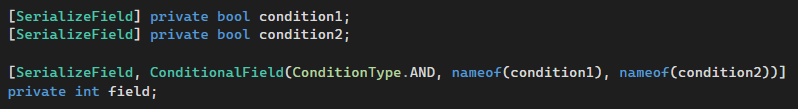
ConditionalField Attribute

Attribute to show/hide a field based on a bunch of conditions.

Parameters:

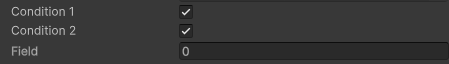
* (ConditionType enum) conditionType: How to evaluate the the specified booleans
* (optional, bool array) negatedValues: Specify which booleans to negate
* (params, string) booleanNames: The names of the booleans to evaluate

The field will be shown when the result of the specified conditions return true.



In this case we set the ConditionType to be AND so the evaluation will be something like this:

“*condition1 == true && condition2 == true*”





We can also set it to OR and it will evaluate like this:

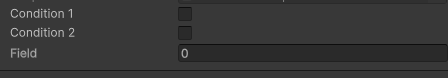
“*condition1 == true || condition2 == true*”





NAND which will evaluate like this:

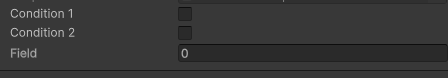
“*condition1 == false && condition2 == false*”





And NOR which evaluates like this:

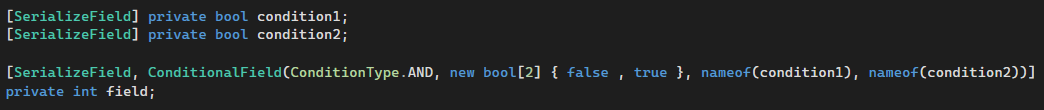
“*condition1 == false || condition2 == false*”.





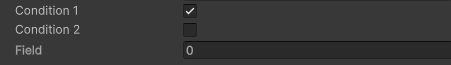
On top of the ConditionType we can also specify a specific value we want to negate by creating a bool array that must be the same size as the amount of conditions we have, in our case we only have 2 conditions so we create a new bool array of size 2 and in the brackets we specify the conditions we want to negate in the order we added them.

We set true for conditions we want to negate and set false for conditions we don’t.



So in this case we only negate “condition2” and the evaluation will be something like this:

“condition1 == true && condition2 != true”









**Limitations:**

* The attribute cannot look for the conditions inside a struct
* It cannot hide arrays or lists, only the fields inside them

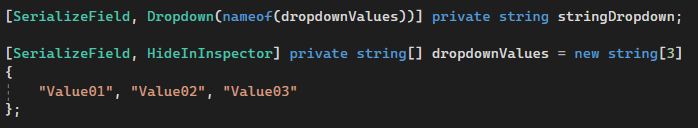
Dropdown Attribute

Attribute to make a string dropdown menu out of a string array.

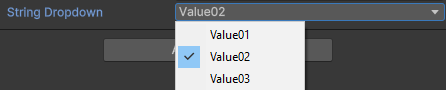
Parameters:

* (string) stringArrayName: The name of the string array (A list can also be used)

The dropdown attribute can only be attached to a string field.



Now you can specify the value of a string from a dropdown.



**Limitations:**

* The attribute cannot look for the string array inside a struct

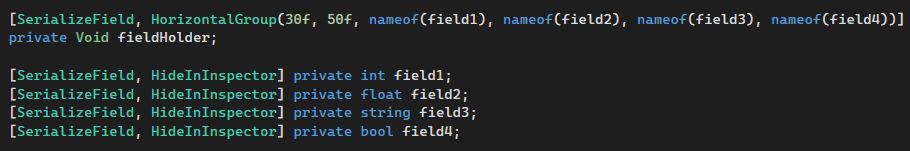
HorizontalGroup Attribute

Attribute to display the specified fields horizontally.

Parameters:

* (float) labelWidth: The width of the field labels (default value = 50f)
* (float) fieldWidth: The width of the input fields (default value = 50f)
* (params, string) fieldsToGroup: The name of the fields to group

We use the SerializeField and HideInInspector attributes because we don’t want to see those fields in the inspector outside of the horizontal group but they still need to be serialized by unity.





**Limitations:**

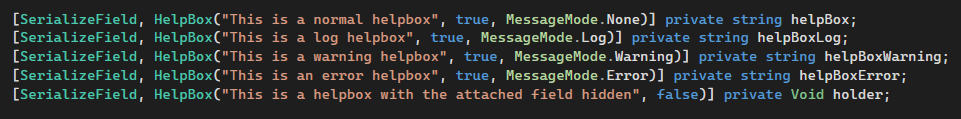
* The attribute cannot look for the fields inside a struct

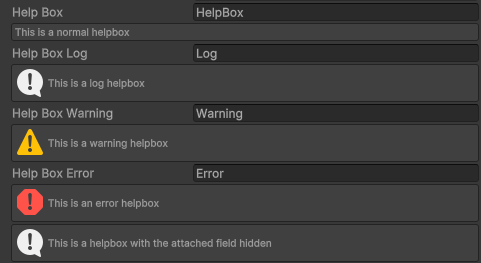
HelpBox Attribute

Attribute to display a help box.

Parameters:

* (string) message: The message to display
* (optional, bool) drawProperty: Draw the property this attribute is attached to (default value = true)
* (MessageMode enum) messageType: The type of the message (default value = MessageMode.Log)





**Limitations:**

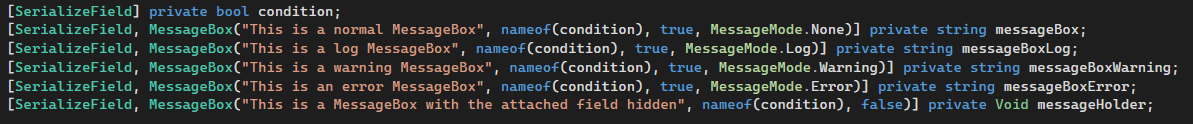
* Doesn’t work well with arrays or lists

MessageBox Attribute

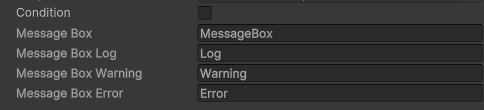
Attribute to display a message box depending on a condition.

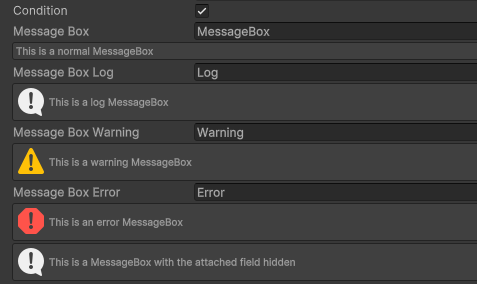
Parameters:

* (string) message: The message to display
* (string) conditionName: The condition to evaluate
* (optional, bool) drawProperty: Draw the property this attribute is attached to (default value = true)
* (MessageMode enum) messageType: The type of the message (default value = MessageMode.Log)



The message boxes will show only when the condition is true.





**Limitations:**

* The attribute cannot look for the condition inside a struct
* Doesn’t work well with arrays or lists

Button Attribute

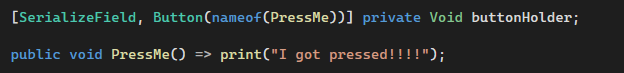
Attribute to add a button in the inspector.

Parameters:

* (string) functionName: The name of the function to call when the button is pressed
* (optional, string) buttonLabel: The label displayed on the button (default value = the name of the function)

Because a function is not a property that can be displayed in the editor we need to have a holder variable with the Button Attribute on it that looks for the specified function and draws the holder as a button while invoking the function when pressed.

The function must be public for it to be serialized and it cannot have parameters.





**Limitations:**

* The attribute cannot look for functions inside a struct
* Doesn’t work well with arrays or lists