

ADVANCED INSPECTOR

TECHNICAL DOCUMENTATION (v1.5)

Contents

TECHNICAL DOCUMENTATION (v1.5)	1
Introduction	6
Structure	7
How to Use.....	8
Tools.....	9
Copy and Paste.....	9
Drag and Drop Copy.....	9
Pick Tool.....	9
Watch.....	10
Save.....	11
Base Class.....	12
InspectorField	12
InspectorLevel.....	13
InspectorEditor	14
ExternalEditor	14
FieldEditor.....	15
EditedTypes.....	16
Expandable.....	16
Contextual.....	17
InspectorLevel.....	17
ComponentMonoBehaviour	18
Interface.....	19
IDataChanged.....	19
IInspectorRunning.....	20
IListAttribute	20
IPreview	21
IRuntimeAttribute.....	21
Data Type	23
UDictionary	23
RangeInt/RangeFloat	24

Attributes	25
AdvancedInspector	25
ShowScript	25
InspectorDefaultItems	25
Angle	25
Snap.....	25
Background	26
Bypass	26
Collection (attribute).....	26
Display.....	27
EnumType	27
Size	27
Sortable.....	28
CreateDerived	29
Descriptor.....	30
Name.....	30
Description	30
URL	30
Icon.....	30
Color.....	30
DisplayAsParent	31
DontAllowSceneObject	31
Enum (attribute)	32
Display.....	32
Masked.....	32
Expandable.....	33
Expanded.....	33
Expandable.....	34
FieldEditor (attribute)	34
Type.....	34
Group	34
Name.....	34

Style.....	34
Priority.....	35
Expandable.....	35
Help	35
HelpType	35
Message	35
Position	36
Inspect.....	36
Condition.....	36
Level	36
Priority.....	36
Method (attribute).....	36
Display.....	37
RangeValue	38
Min	38
Max	38
ReadOnly.....	39
Restrict	39
Display.....	39
RuntimeResolve	40
Space	41
Size	41
Style.....	41
Label	41
Name	41
Tab	42
TextField.....	43
Type.....	43
Title	43
Path	43
Extension.....	43
Toolbar	44

Name.....	44
Label.....	44
Style.....	44
Priority.....	44
Flexible	44
Contact.....	45
Package Revision.....	46
1.0:	46
1.1:	46
1.2:	46
1.3:	47
1.31:	48
1.32	48
1.4	49
1.41	49
1.42	50
1.43	50
1.44:	51
1.5:	51

Introduction

You got your hands - somehow - on the Advanced Inspector, and now you're wondering what it can do and how you are supposed to use it.

The Advanced Inspector is our answer to Unity's lacking Inspector. We believe someone should not have to write any Editor, or only in the very rare case of a very specific rendering. Most of the time, the Inspector should display data the way we want it, without a need to rewrite a custom behaviour each time.

While writing this tool, we decided to add numerous other features and it quickly became a huge undertaking. The purpose of this document is to list as much of those features as possible and how they work.

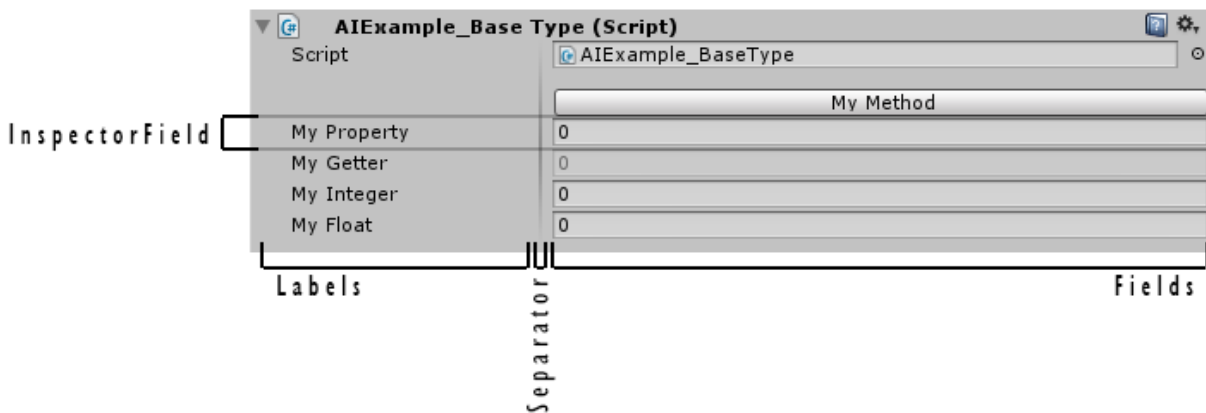
As with any tools we release, we follow the SOLID principle. This means our tools are always closed to changes, but opened to expansion. For this reason, the core of the tool is within a documented library, but every class that has an expansion purpose are left outside.

You can easily add your own attribute or FieldEditor and modify the basic behaviour of this package. You can adapt other package to be friendly to this tool.

If you wish to have access to the library source code, contact us. However, you should not have any need to modify the base classes. Note that our libraries are fully commented with an external .xml. You can browse those comments in the code editor of your choice.

Structure

Unlike Unity's Inspector, Advanced Inspector does not allow free-for-all rendering of items. It has a structured way of displaying data. This limitation, in counter-part, offer the chance of adding feature that would be impossible if no structure existed.



Each item displayed by the Advanced Inspector is represented by an instance of the InspectorField class.

Each InspectorField are handled by the AdvancedInspectorControl class and drawn into two specific zone separated by a drag-able separator.

The Labels zone is the name of each InspectorField. The name is selectable and right-clickable for a contextual menu related to that InspectorField's type. It can also be drag to other label to perform a quick copy/paste of the field's value. In front of the label, there's space for contextual icon, such as an expanding arrow, a destructible minus "-" icon, or a draggable icon for re-orderable list.

The Fields zone is where independent FieldEditor are drawing how the data is displayed and edited. When a new type is encountered and no FieldEditor is defined for it, the Advanced Inspector falls back to displaying the type's name and assumes it's an object which may or may not be expandable.

How to Use

By default, the package does not change the behaviour of the Inspector. To flag a class as one taking advantage of the Advanced Inspector, you must flag it as such;

```
[AdvancedInspector]
public class AIExample_BaseType : MonoBehaviour { }
```

The "AdvancedInspector" attribute flags a class as being Advanced Inspector friendly.

By default, the Advanced Inspector displays nothing unless told to with the Inspect attribute. However, you can force it to display fields by following the same rules as Unity does;

```
[AdvancedInspector(InspectDefaultItems = true)]
public class AIExample_BaseType : MonoBehaviour { }
```

This behaviour was chosen to be that way considering that any property or parameter-less method can be exposed. It quickly became clear that displaying member based on their serialization context or on their access keyword - private/public - was a bad idea.

In exception to the "InspectDefaultItems" flag, to expose a member to the Advanced Inspector, it has to sport the "Inspect" attribute.

```
[Inspect]
public int myInteger;

[Inspect]
public float MyProperty
{
    get { return myFloat; }
    set { myFloat = value; }
}

[Inspect]
private void MyMethod() { }
```

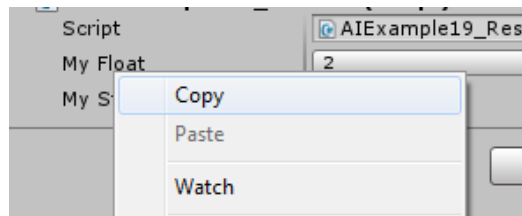
The Advanced Inspector doesn't take into account serialization context - if a value is serializable or not - or its access level - public, protected, private. In case of properties, it is up to you to make sure the backing field is properly serialized.

Tools

The Advanced Inspector offers a few tools that are not from Unity.

Copy and Paste

It is possible to copy and paste any kind of value with the Advanced Inspector. Simply right-click the label of the value you want to copy and select "Copy".



The "Paste" contextual menu is grayed out if the currently copied value cannot be pasted in the selected field.

This copy/paste is not limited to numbers or string; complex object and nested object hierarchy can also be copied.

Drag and Drop Copy

You can drag a label into another label. This automatically copies the content of the first field to the second one, similar as if you had done "right-click > copy, right-click > paste". The dragging method is obviously a lot faster and much more intuitive.

Unity used the dragging of a label to modify float and integer. This feature still exist, however you need to press Shift to access it, instead of the copy by drag.

Pick Tool

Some field may display a pick icon at their far right:



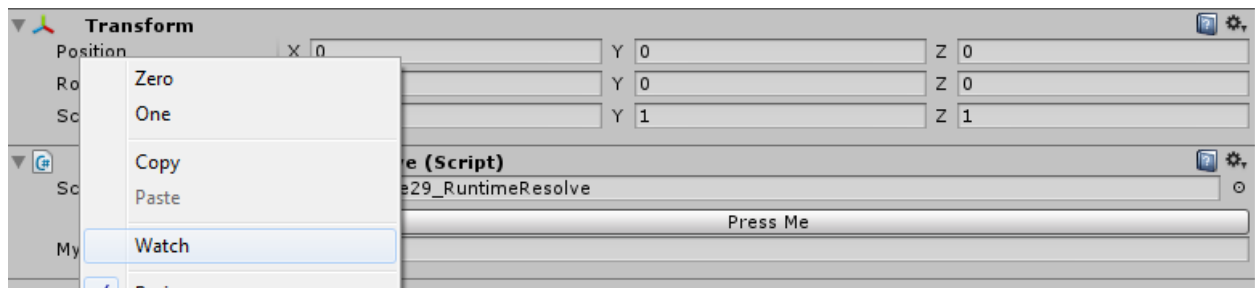
When clicked, the cursor turns into a crosshair and you can "pick" a GameObject from the SceneView. This icon shows up automatically when the following condition are met;

- The field does not have the DontAllowSceneObject attribute
- The field's type is GameObject or
- The field's type derive from Component or can be derived into Component.

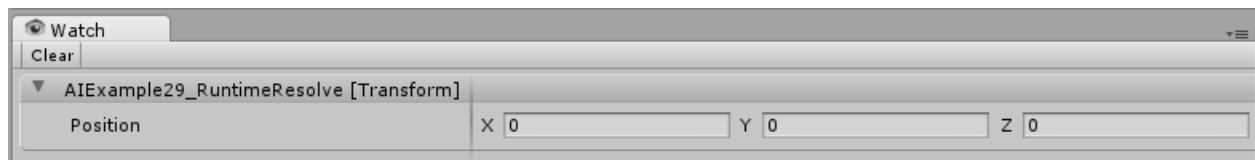
If you want to exit the picking tool, simply press Escape on the keyboard.

Watch

The watch window is accessible by right-clicking on any label and selecting "Watch" or from the menu Window > Watch.



This panel is similar to the watch feature of Visual Studio or MonoDevelop; it allows you to track a variable - in this case an Inspector Field - while working on something else.



You can remove a single field from the watch window by right-clicking the label again and selectionning "Watch". Or you can use the "Clear" button at the top of the panel to clear all watched fields.

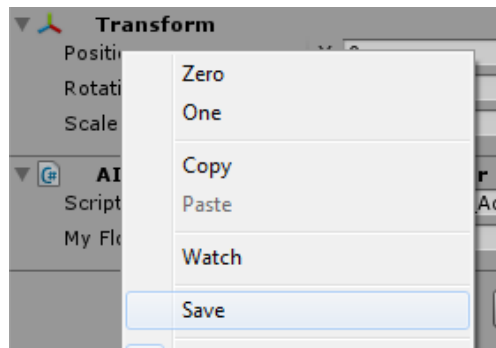
Right-clicking on a watched label shows another contextual option: "Select". This menu item selects the parent serialized object of this label; in most cases being the GameObject in which this field exists.

Note that the Watch feature also support multi-selection.

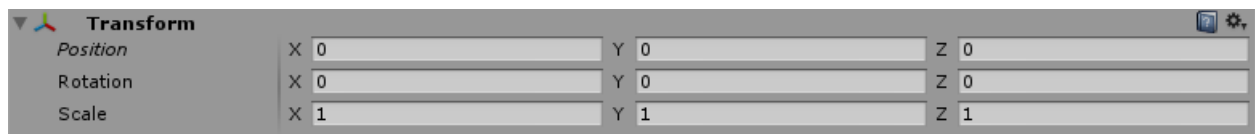
Save

Please note this feature is still in beta and may not be fully stable.

The "Save" contextual menu item is available up only while you are playing your game in the editor. It allows you to "save" the changes you have done on a specific values.



For example, if you move an object around and stop your game, Unity will bring back the object to its original position. While you have move your object and Unity is still playing, right-click the position label and select "Save". The menu item should now be selected and the label should appear in italic. If you stop Unity, the object will keep his newly saved position.



If you want to revert your "Save", you can select the same menu item again, and it should now be unselected.

Please note that the "Save" feature takes the current value, not the one when you stop Unity playback. If you change your value and want to keep the latest one, you will have to re-save it.

Base Class

The Advanced Inspector has a few base classes that are used to expand or simply make the system work. Here's a list of the most important one.

InspectorField

The InspectorField class is the class that replace what in Unity is known as SerializableProperty. It represents a single item drawn on the Inspector. However, unlike Unity, this class is not limited to a single serialization context and is not limited to handling fields. It is similar in aspect to a Tree View Item.

Similar to Unity, InspectorField handles undo, multi-selection, prefabs and so on. Unlike Unity, it also handles properties, method, groups, copy-paste, apply-revert, sorting and many other things.

Also, the InspectorField handles all the data by reflection, and not by serialization. This means non-serializable values are also displayable without custom editor. For this reason, you must understand that the Advanced Inspector is not a view on what is serializable, but a view on the data you choose to display.

The Advanced Inspector does not modify how Unity serializes (save) its data.

The InspectorField contains a huge collection of properties and method to handle data and its display. While you may never need to directly handle a InspectorField, you can if you wish from a class deriving from InspectorEditor.

GetMembers, GetIndexedProperties and GetKeyedProperties are three static methods used to create a collection of InspectorField from an object. The standard behaviours of the InspectorField come from those three methods.

However, you can create instances of InspectorField manually and build - by hand - the look you wish for. An example of that is how Unity's internal editor type has been "rewritten" to work with the Advanced Inspector. Since most of Unity's type does not store data in fields, but are only reflection of data stored in Unity's C++ scope, each exposed field must be declared by hand. See the custom InspectorEditor for this task. Here's the reworked TransformEditor;

```

[CanEditMultipleObjects]
[CustomEditor(typeof(Transform), true)]
public class TransformEditor : InspectorEditor
{
    protected override void RefreshFields()
    {
        Type type = typeof(Transform);

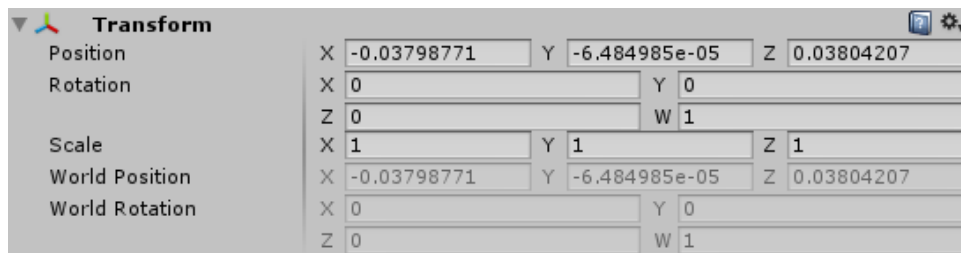
        fields.Add(new InspectorField(type, Instances,
type.GetProperty("localPosition"), new DescriptorAttribute("Position", "Local
Position"))));
        fields.Add(new InspectorField(type, Instances,
type.GetProperty("localRotation"), new DescriptorAttribute("Rotation", "Local
Rotation"))));
        fields.Add(new InspectorField(type, Instances,
type.GetProperty("localScale"), new DescriptorAttribute("Scale", "Local Scale"))));

        fields.Add(new InspectorField(type, Instances,
type.GetProperty("position"), new ReadOnlyAttribute(), new
InspectAttribute(InspectorLevel.Debug),
        new DescriptorAttribute("World Position", "World Position"))));
        fields.Add(new InspectorField(type, Instances,
type.GetProperty("rotation"), new ReadOnlyAttribute(), new
InspectAttribute(InspectorLevel.Debug),
        new DescriptorAttribute("World Rotation", "World Rotation"))));
    }
}

```

InspectorLevel

It's important to note here how the two last "fields.Add" lines create an InspectorField flagged in Debug level. Those fields are only shown up in the Advanced Inspector in debug mode. In this case, it's a binding towards the world-based position and rotation which are not usually exposed in the standard editor. The result is in debug mode;



InspectorEditor

The entry point of all objects in the Inspector panel is a type deriving from "UnityEngine.Editor". The class "InspectorEditor" is a direct abstract class deriving from Editor. Unlike Editor, it is not limited to Unity's type as object entry point.

From this, we have three other derived classes;

- BehaviourEditor; Handles all the MonoBehaviour
- ScriptableEditor; Handles all the ScriptableObject
- ExternalEditor; Used to draw a full Inspector within the EditorWindow of your choice, no Inspector panel needed. See the EditorWindow example in the package.

If you want to code your own Editor while still using the feature of the Advanced Inspector, you should derive your type from one of the above class if they are from MonoBehaviour or ScriptableObject.

It is also possible to do the same for Unity's internal type. You can see an example in "TransformEditor" which override Unity's own editor for the Transform type. The folder "UnityTypes" is a collection of Editor written using Advanced Inspector for Unity's internal type.

ExternalEditor

The ExternalEditor is simply a wrapper around the functionality of the AdvancedInspector and expose a IControl interface allowing it to be added to any EditorWindow as a kind of GUI element. It should be created like this;

```
private ExternalEditor editor;

private void OnEnable()
{
    editor = ExternalEditor.CreateInstance<ExternalEditor>();
}
```

To have it display something, you have to pass it objects like this;

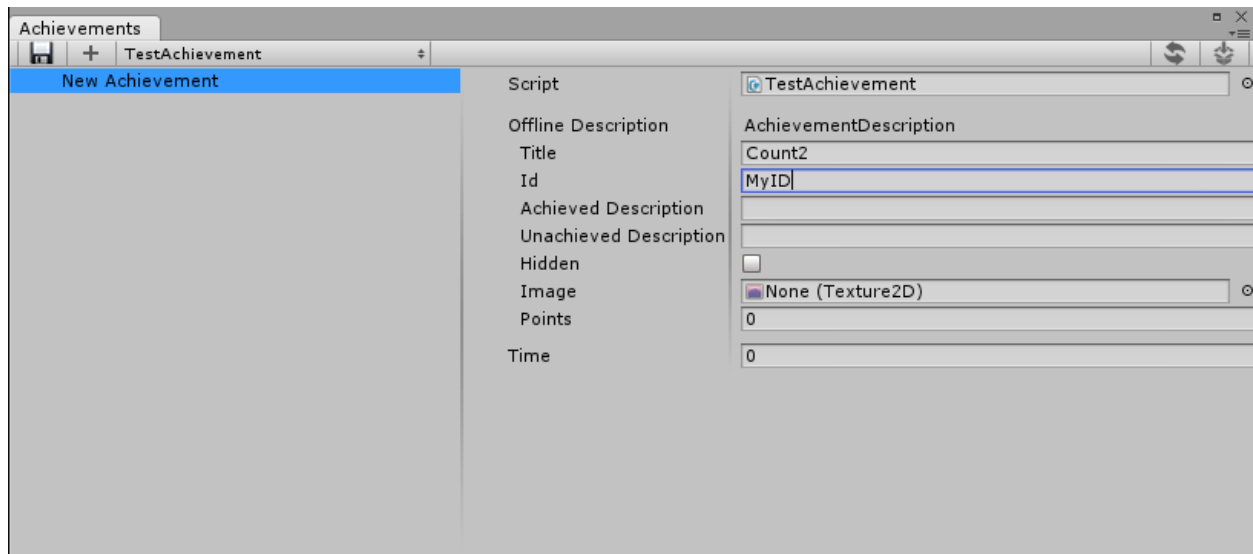
```
editor.Instances = new object[] { selection };
```

Finally, on your EditorWindow's OnGUI call, invoke the editor's "Draw" method;

```
if (editor.Draw(new Rect(100, 100, 250, 250)))  
    Repaint();
```

As per the IControl implementation, the Draw method returns true if the drawn control requires the window to be repainted.

If you did everything properly, you can easily make an EditorWindow like this;



FieldEditor

The FieldEditor is the base class for all objects that draws on the Field area of the Inspector. All FieldEditor packaged with the Advanced Inspector have their source exposed. You can add new FieldEditor or edit existing one.

At initialization, the Advanced Inspector retrieves all the type deriving from this abstract class and creates an instance of each. For performance reason, FieldEditor are not created and destroyed for each InspectorField. If you display ten floats, the same instance of the FloatEditor will be used to draw them all. For this reason, storing values directly into a FieldEditor is not advised.

Here's an example of a custom FieldEditor made to display a Guid;

```

public class GuidEditor : FieldEditor
{
    public override Type[] EditedTypes
    {
        get { return new Type[] { typeof(Guid) }; }
    }

    public override void Draw(InspectorField field, GUIStyle style)
    {
        // GetValue returns null if multi-selection and different values
        Guid id = field.GetValue<Guid>();
        if (id == Guid.Empty)
            GUILayout.Label("GUID: -----");
        else
            GUILayout.Label("GUID: " + id.ToString());
    }
}

```

You can notice that FieldEditor - unlike PropertyDrawer - fully support Layouting. (GUILayout/EditorGUILayout). Also note that everything drawn in the Fields zone is from a FieldEditor. Every existing Unity types have been re-coded using FieldEditor.

The Draw method also pass any GUIStyle that you may have tagged your fields/properties/methods with using the StyleAttribute.

EditedTypes

A single FieldEditor may handle more than one type. There is no limit of type that can be listed in the "EditedTypes" property. A FieldEditor may also edit a base type and all derived type. However, the Advanced Inspector will always prefer a FieldEditor that has an explicit type over a base type.

```

public override Type[] EditedTypes
{
    get { return new Type[] { typeof(Vector2), typeof(Vector3), typeof(Vector4) }; }
}

```

Expandable

A FieldEditor can also prevent that a specific InspectorField can draw children fields. For a field to be expandable, both the source data and the FieldEditor must "agree" on that field's state.

```

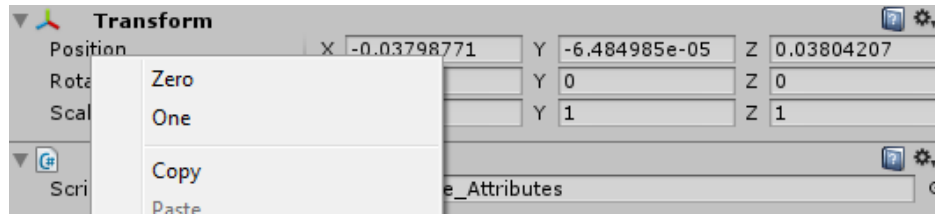
public override bool Expandable
{
    get { return false; }
}

```


This way, someone can prevent a specific type, drawn from a FieldEditor, from being expanded from a misuse of the Expandable attribute.

Contextual

Every FieldEditor has access to adding elements to the contextual menu invoked when right-clicking on the label of their InspectorField. Here's an example of the menu option added when invoking the menu for a Vector3 type;



They are added by doing;

```
public override void OnContextualClick(InspectorField field, GenericMenu menu)
{
    menu.AddItem(new GUIContent("Zero"), false, Zero);
    menu.AddItem(new GUIContent("One"), false, One);

    menu.AddSeparator("");
}
```

Other standard elements, such as copy, paste, apply, revert and so on are added after the type-specific one.

InspectorLevel

The inspector levels are also visible from the FieldEditors. As example, the Quaternion FieldEditor display its data as Euler angles in basic and advanced mode, and in normalized quaternion (X, Y, Z, W) in debug mode.

ComponentMonoBehaviour

Unity is able to keep proper polymorphism in only two cases; ScriptableObject and MonoBehaviour. Since ScriptableObject cannot exist in a prefab, it leaves us MonoBehaviour to work with for a polymorphic sub-component.

The base abstract class ComponentMonoBehaviour is an important part of the composition/sub-component pattern. You should never apply the CreateDerived attribute on a field or property that does not derive from ComponentMonoBehaviour, otherwise your object will revert to the field type on the next serialization reload.

The goal of this class is to flag an instance made from this specific class hierarchy as being "not-standalone", but as being owned by another MonoBehaviour. For this reason, any ComponentMonoBehaviour found by the Advanced Inspector to not have a valid parent, and said parent to not target the child in one of its fields, is automatically destroyed in a recursive manner.

The Advanced Inspector handles this specific class in a few particular ways compared to other MonoBehaviour;

- Destroyed when the direct parent is destroyed, and destroys children on its own destruction
- A copy/paste doesn't copy the reference, but performs a full recursive deep-copy
- Hidden in the Inspector as being a standalone script, only displayed in its parent
- Can only be referenced by one parent
- Assumes a choice of different derived class

See the "CreateDerived" attribute for more information in creating polymorphic instance directly from the Inspector.

Interface

The Advanced Inspector also automatically recognizes some interfaces and act accordingly. They allow changing the behaviour of attributes or contribute to inter-object communication.

IDataChanged

This interface is a two-ways binding between an inspected object and the Advanced Inspector. It has one method and one event that are called when internal structures are changed.

```
/// <summary>
/// Fired when the Inspector changed.
/// </summary>
void DataChanged();
```

The method is called by the Advanced Inspector when something occurs on the GUI. Currently, this is fired whenever something changed in the GUI, no matter if a value changed or not. We are aware of this issue.

```
/// <summary>
/// Should be fired internal by the object when the fields structure changed.
/// Ex.: Added an object to a list.
/// </summary>
event GenericEventHandler OnDataChanged;
```

This event should be implemented by the class and should be raised when the internal data of that class has changed without interaction from the Advanced Inspector. As example, if an external editor added an item to a list, the Inspector has no way to know the list has more indexes. Raising this event insure that the data displayed is up to date. Note that data displayed by a field is refreshed when the Inspector is redrawn. This event should only be used when the number of InspectorField changed, such as in a list or a dictionary.

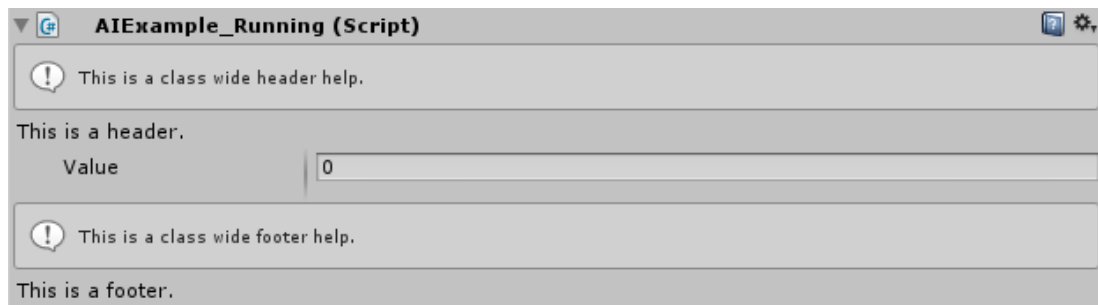
Note that in most cases, you don't have to implement this interface. It exists only when internal processing occurs independently of the Inspector.

IInspectorRunning

A "running" in a document define the space called "header" and "footer". In the Advanced Inspector, each object inspected has its own header and footer, which you can access and draw on using the IInspectorRunning interface.

```
[AdvancedInspector(false)]
public class AIEExample_Running : MonoBehaviour, IInspectorRunning
{
    public void OnHeaderGUI()
    {
        GUILayout.Label("This is a header.");
    }

    public void OnFooterGUI()
    {
        GUILayout.Label("This is a footer.");
    }
}
```

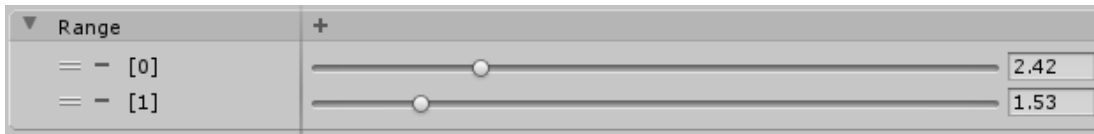


IListAttribute

This interface should only be implemented by attribute. This interface forces the Advanced Inspector to publish attribute tagged on their parent array/list to the nested InspectorField.

```
[Inspect]
[RangeValue(0, 10)]
public float[] range;
```

In the above snippet, the attribute "RangeValue" is flagged with IListAttribute. This attribute is passed down to the InspectorField used to display the different indexes of this array and gives the following result;



IPreview

This interface expose a single property that is used to return the objects that need to be previews in the Inspector. The Advanced Inspector automatically handle the object to be properly displayed.



An instance of a script is able to preview multiple object. Use the left/right scroll arrow to switch between object being previewed. To switch between script themselves, you have to select the script by clicking on any field currently drawn of that specific script. Unity does not support multiple script accessing the preview zone at the same time.

The currently supported type for the preview is;

- GameObject (Prefab)
- Mesh
- Material
- Texture
- Cubemap

IRuntimeAttribute

This interface is also used by Attributes only. This is probably one of the most powerful features of the Advanced Inspector. An attribute using this interface will see its internal value resolved at runtime by a method or delegate supplied by the creating object.

Usually, attributes can only be initialized by constant or type declaration. It means that a definition cannot change at runtime. To go around this, you can supply a method's name as constructor, or if your inspector is built at runtime, you can give it a delegate using a proper method signature.

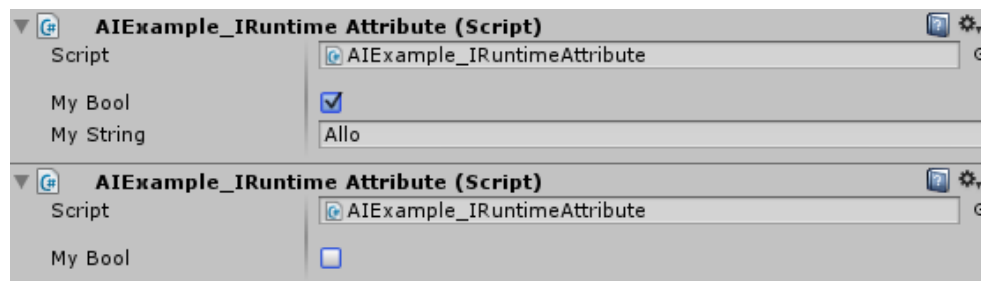
This way, the actual value of this attribute is resolved at runtime and can change based on different condition.

```
[Inspect]
public bool myBool = true;

[Inspect("IsVisible")]
public string myString = "Allo";

private bool IsVisible()
{
    return myBool;
}
```

In the previous snippet, the inspected field "myString" is visible only if the method named "IsVisible" returns true. Since this method returns the value of a member that is also inspected;



Note that InspectorField hidden this way are not destroyed, but simply not drawn. They still exist and can still be accessed.

Lot of attributes in this package are implementing `IRuntimeAttribute`. As example, the Descriptor attribute let's you change the name, tooltip, help url or even color of a field. The Help attribute is quite useful as it can show up help boxes depending on other values.

Data Type

In some cases, Unity doesn't always offer the base "data" type for all situation and leave lot in your hands. Since the Advanced Inspector is not bound to Unity's serialization, it is also able to display types that are not serializable directly by Unity.

This even truer in Unity 4.5 with the introduction of the `ISerializationCallbackReceiver` interface that allow conversion of data to and from a format Unity is able to understand.

UDictionary

Standing for Unity-Dictionary, this class is a fully working Unity 4.5 dictionary. It works right out of the box. The only exception is that Unity is still unable to properly serialize generic class; so you will need to create a non-generic top class from the base generic one.

Here's an example;

```
[Serializable]
public class SFDictionary : UDictionary<string, float> { }

[Inspect, RangeValue(0, 100)]
public SFDictionary dictionary1 = new SFDictionary();
```

In the above example, we created a dictionary with string as key and float as value. In Unity, the Advanced Inspector shows types implementing `IDictionary` as followed;



The top field with the "+" sign is used to add a new key to the dictionary. The button remains grayed-out if the key is invalid or already exist in the dictionary. Items in a dictionary cannot be re-ordered as they have no internal order.

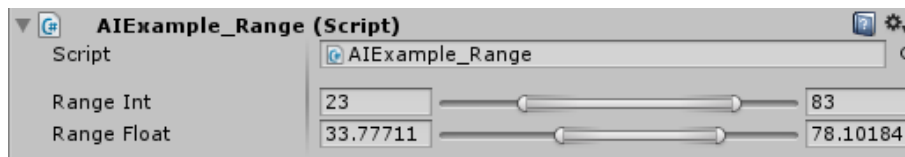
RangeInt/RangeFloat

Those two types are used as a min/max range struct. In Unity 4.5 and later, those types are structs, but in previous version they are classes. Unity introduced struct serialization only in 4.5.

Those types are only used with the RangeValue attribute. The Advanced Inspector does not display them otherwise. Here's an example of its use;

```
[Inspect, RangeValue(0, 100)]  
public RangeInt rangeInt = new RangeInt(25, 75);
```

And the result in the inspector;



Attributes

The Advanced Inspector is an attribute-based system. Almost every behaviour can be modified by adding an attribute to an exposed member. Most attributes are also fully combinable. The possibilities are almost endless. On top, the system is made so that you can create your own attributes as easily as possible. InspectorField class has methods to retrieve the current attributes.

AdvancedInspector

Class only, this attribute is the on/off switch between the default Unity's inspector and the Advanced Inspector.

ShowScript

This property is a switch to show or hide the "Script" field at the top of the inspected object. Default value is true.

InspectorDefaultItems

Emulate the default inspection pattern used by Unity. The Inspector displays all the following items; public fields without "HideInInspector" and without "NonSerialized" attribute, and all private field with the "SerializedField" and without "HideInInspector" attribute. Note that item that Unity doesn't usually display can still be inspected with the "Inspect" attribute.

Angle

Fields and properties only, this attribute turns float or integer fields into a spinning wheel.

Snap

This property allows the wheel to "lock" at an increment of a value. For example, with a value of 5, the wheel will "snap" at 0, 5, 10, 15, 20, etc. Default value is -1, for no snap.



Background

This attribute should only be applied to item that can be expanded, such as Expandable object or collection. Currently it only modifies the color of the background boxing, making it easier to find specific items.

Instance	0 (AIExample_ComponentB)
Script	AIExample_ComponentB
Value Of Class B	0
Value Of Base Class	0

Bypass

Fields and properties only, this attribute forces a non-advanced inspector type to behave like one, forcefully exposing every public fields and properties. Useful to expose internal Unity type that doesn't have their own Editor. Note that this attribute is passed down to every single child object referenced from the flagged one. ***Use at your own risk, some fields or properties from Unity's types are not safe to modify in some specific context.***

Collection (attribute)

This attribute is used to control how a collection is displayed and managed.

Usually, in Advanced Inspector or Unity's inspector, when you add an item to a collection - list or array - the item's type has to have a parameter-less constructor. Otherwise, the inspector issue errors that an instance cannot be properly created. Unity's usually goes around this by creating a serialized instance anyway. However, this can lead to an item being improperly initialized since the right constructor was not called.

The Collection Constructor attribute should only be placed on collection. It offers a delegate from which you can supply a proper instance of the right type. Since you create the object on your own, it allows you to use the proper constructor or even change parameters of your object before giving it to be added to the collection.

```
[Inspect, CollectionConstructor("OnReleaseConstructor")]
public ActionBinding[] onRelease = new ActionBinding[0];

private object OnReleaseConstructor()
{
    return new ActionBinding(new Type[] { typeof(Material) });
}
```

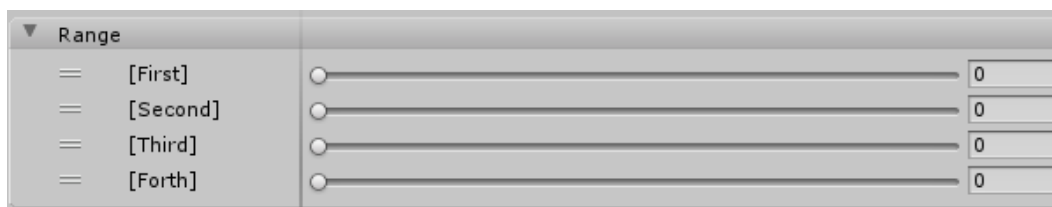
In this example, the ActionBinding instance is supplied manually. The method is invoked anytime a new instance has to be added to the collection. It is also called repeatedly when first initialized a collection with the Size attribute.

Display

The display property allows to change how a collection is displayed. The default value is "Default", which display every item of a collection. Every other option removes the full list and only display one item at a time. In this case, a selection control is shown in the collection display field, so the user can selection which index or key is inspected. The current option are default, buttons and drop down list.

EnumType

This property allows you to bind the size of a collection to the values of an enumeration. On top, the usual index display "[1]" is replaced by the enum's names; "[My Value]". The type passed to this property must be an enum, otherwise it is set to null.

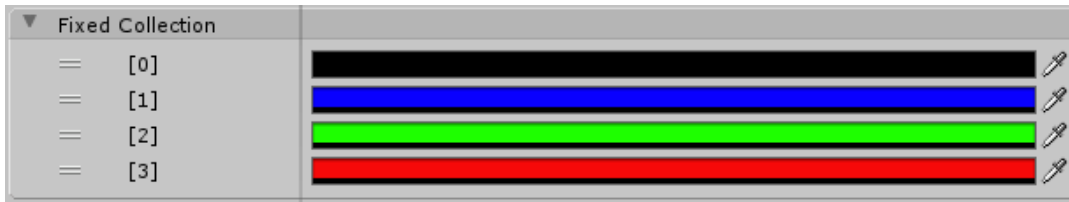


Range		
<input type="radio"/>	[First]	0
<input type="radio"/>	[Second]	0
<input type="radio"/>	[Third]	0
<input type="radio"/>	[Forth]	0

Size

This allows to restrict control over the size of the collection. The default value is "-1", which is the normal way of displayed collection. "0" removes the add/remove controls and leaves the size of the collection to the control of your own code. If you add any other positive numbers, the Advanced Inspector will initialize the collection with the specified size.

```
[Inspect, Size(4)]
public Color[] fixedCollection;
```



Sortable

This toggles the "sortable" control, the drag icon that allow the user to reorder a collection.

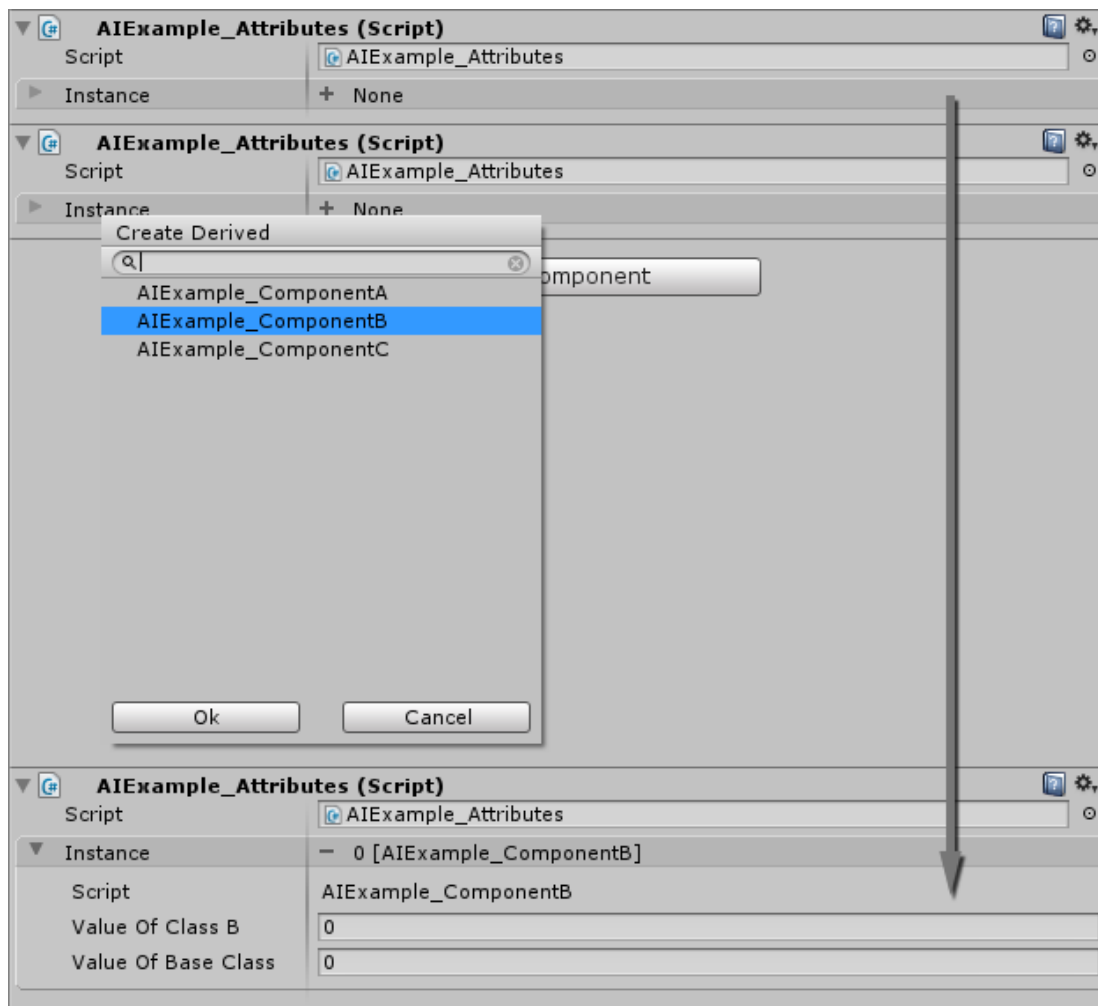
CreateDerived

Fields and properties only, this attribute works in conjunction with the `ComponentMonoBehaviour` base-class. It turns an object reference field into an object instance creator/destructor field.

This attribute should never be placed on a member that is not deriving from `ComponentMonoBehaviour`.

This allows the object to act as a polymorphic sub-component of the parent `MonoBehaviour`. Read more about this in the `ComponentMonoBehaviour` chapter.

```
[Inspect]
[CreateDerived]
public AIExample_ComponentBase instance;
```



Descriptor

Fields, properties, methods and class attribute, this one allows you to change all the information displayed for this member.

Name

Change the name displayed on the label.

Description

Changes the tooltip when the mouse is hovering over the label.

URL

Add a item in the contextual menu of this label. The item invokes a webpage using this URL.

Icon

Not used by the Advanced Inspector, but allow to store an icon for list or tree view.

Color

Change this InspectorField color, label and field. The constructor takes three float since class instance are not allowed in Attribute's constructor. **Unity Pro only feature.**

DisplayAsParent

Fields and properties attribute, this forces a nested object to be displayed as being part of the parent itself. Label indentation still occurs.

```
[Inspect]
[DisplayAsParent]
public Nested nested = new Nested();

[Expandable]
public class Nested
{
    [Inspect]
    public float nestedValue;

    [Inspect]
    [DisplayAsParent]
    public SubNested subNested = new SubNested();

    public class SubNested
    {
        [Inspect]
        public float subNestedValue;
    }
}
```

Nested	Nested
Nested Value	0
Sub Nested	SubNested
Sub Nested Value	0

DontAllowSceneObject

Fields and properties attribute, this prevent Unity's object picker to show the object within the current scene. This is for fields that should only references asset in the project.

Enum (attribute)

This attribute controls how an enum is displayed and managed.

Display

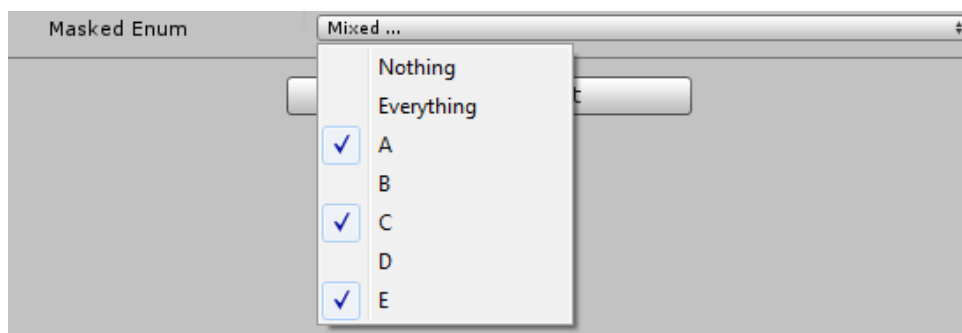
Controls how an enum is displayed. The default value is "DropDown". You have the choice between a drop down list (default), buttons and a checkbox field.

Masked

Turns a normal enum into a masked enum, also known as "bitfield". It's an enum that can contain multiple choices at the same time. **Important; the specified enum must support bitfield assignment.**

```
[Inspect]
[Enum(true)]
public MyMaskedEnum maskedEnum;

public enum MyMaskedEnum
{
    A = 1,
    B = 2,
    C = 4,
    D = 8,
    E = 16
}
```



Expandable

Fields and properties attribute. This attribute is used to override the expandability of an item on a field-by-field basis. It is sometimes useful that a specific object should not be expandable in some context.

```
[Inspect]
public NotExpandable notExpandable = new NotExpandable();

[Inspect]
public Expandable expandable = new Expandable();

[Inspect, Expandable]
public NotExpandable forcingExpansion = new NotExpandable();

public class NotExpandable
{
    [Inspect]
    public float myFloat;
}

[Expandable]
public class Expandable
{
    [Inspect]
    public float myFloat;
}
```

Not Expandable	
▼ Expandable	
My Float	4
▼ Forcing Expansion	
My Float	5

The expansion behaviour works for any field. As example, you can flag a MonoBehaviour as being expandable, allowing it to be inspected from another MonoBehaviour referencing it. Or you can expand ScriptableObject referenced in a MonoBehaviour.

Expanded

If true, forces an item to be expanded by default. Note that if someone closes the object, that state is saved. The Expanded property is only used when the item was never collapsed.

Expandable

If true, the item has an arrow on the left of the label and it can be expanded or collapsed. Any object with the `AdvancedInspector` attribute are expandable by default. It is sometimes useful to override this behaviour.

FieldEditor (attribute)

Fields and properties attribute, forces a field to be rendered by a specific `FieldEditor`.

Not to mix up with the `FieldEditor` base class.

Careful, the Advanced Inspector does not perform any verification as if the requested FieldEditor can support the flagged type. Requesting a FieldEditor that does not support the current type may result in exceptions being thrown.

Type

Name of the `FieldEditor` needed for displaying this member. The type of passed by name because the `FieldEditor` reside in the Editor side of Unity, which is invisible to the game side.

Group

Fields, properties and methods attribute. Group different items under a named expandable group. Group members are not indented.

Name

Name of the group. Item having group attribute with similar name are grouped.

Style

Style name. Override the style of the whole group. Allows making group "boxes". Must be a valid `GUIStyle` name.

Priority

Similar to Inspect attribute's priority value, allow the group to be placed in a specific spot in the list of InspectorFields.

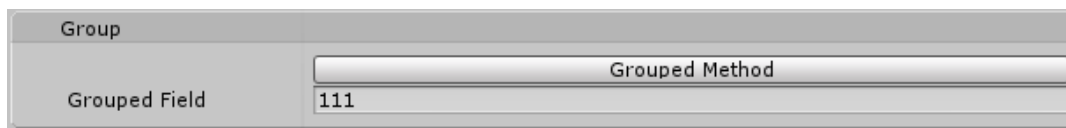
Expandable

If false, removes the foldout control from the group label and make it always expanded. This can

```
[Inspect]
public void GroupedMethod() { }

[Inspect]
public float groupedField;
```

be useful is you want to use the group only as a sorting box.



Help

Fields, properties, methods and class attribute. This attribute display an help box under the current member or in the header/footer in cases of classes. This is most useful as a IRuntimeAttribute since the help message can be modified following internal runtime condition.

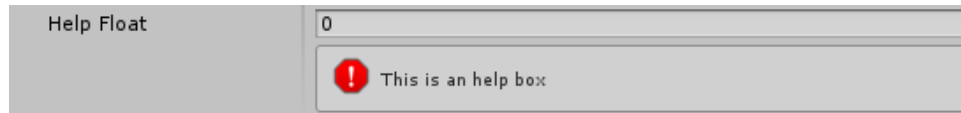
HelpType

The type of the box. Possible value; Error, Warning, Info, None. This changes the icon of the help box.

Message

The message displayed in the box.

```
[Inspect]
[Help(HelpType.Error, "This is an help box")]
public float helpFloat;
```



Position

The position of the helpbox, before or after the member. Default is after. In cases of classes, "before" defines the header while "after" is the footer.

Inspect

Fields, properties and methods attribute. This allows an object member to be displayed by the Advanced Inspector.

Condition

Boolean to inverse the condition received by a runtime method. Only useful if multiple members are using the same method to test their visibility.

Level

Flag an item to be visible only in a specific Inspector Level or above. Example, an "Advanced" item is also visible in "Debug" level.

Priority

When sorting item, priority of this item. Smaller numbers are displayed before larger one. Default is 0, negative value are supported.

Method (attribute)

Methods - functions - are inspectable too. By default, the Advanced Inspector displays a method in the form of a button. Currently, the only limitation is that the method must take no parameters. The Method Attribute is there to modify this default behaviour. For now, the current option allows to turn that button into an automatic invocation of the method on the Inspector redraw.

Display

The options are "Button", which is the default, and "Invoke", which invoke the method every times the Inspector refresh it's view. This allows you to draw whatever you wish on the Inspector without the need to create a dedicated FieldEditor and/or a custom type.

RangeValue

Fields and properties attribute, similar to Unity's "Range" attribute. However, unlike Unity's one, this one is not restricted to fields and sports the IListAttribute interface. Turns a integer or float field into a slider.

Min

Minimum value of the slider, left side.

Max

Maximum value of the slider, right side.

```
[Inspect]  
[RangeValue(0, 10)]  
public float rangeFloat;
```



ReadOnly

Fields, properties and methods attribute, disable a field from being editable. This is useful to display internal values that you do not wish to be possible to edit from the Inspector. Being a `IRuntimeAttribute`, it can be turned on and off based on internal condition.

Note that this does not prevent value from being modified by code.

```
[Inspect]
[ReadOnly]
public float readOnlyFloat;
```

Read Only Float

0

Restrict

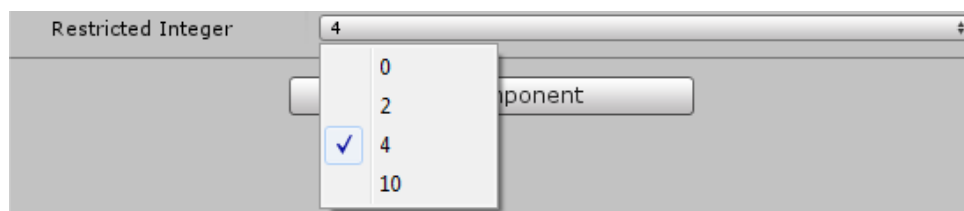
Fields and properties attribute, bypass the `FieldEditor` and turns any field into an enum. This attribute can only work in `IRuntimeAttribute` mode and does not support compile time values. Quite useful when a specific field should not offer endless choices.

```
[Inspect]
[Restrict("Restrict")]
public float restrictedInteger;

private IList Restrict()
{
    return new List<object>() { 0, 2, 4, 10 };
}
```

Display

Controls how the restrict field is displayed. The default choice is "DropDown". The current available options are button and toolbox.



RuntimeResolve

By default, the Advanced Inspector chooses a Field Editor based on the type of the field or property being displayed. It's faster this way, and prevent issue when a value is null or has no proper field editor.

However, in some cases, it's useful that the Advanced Inspector pick a Field Editor based on the current value's type. For example;

```
[Inspect]
[RuntimeResolve]
public object Value
{
    get { ... }
    set { ... }
}
```

In this snippet, the property can be linked to multiple backing field, and the proper type returned is unknown until inspected. In this case, the RuntimeResolve attribute forces the Advanced Inspector to check the value's type and take the proper FieldEditor for it.

RuntimeResolve can also be used to restrict an UnityEngine.Object field to only display a specific subtype.

```
[Inspect]
[RuntimeResolve("GetValueType")]
public object Value
{
    get { ... }
    set { ... }
}

private Type GetValueType()
{
    return typeof(Material);
}
```

In this second example, the object picker would only list Material. It is useful for simulating the behaviour of a generic types - which Unity cannot serialize - while not being generic.

Space

Fields, properties and methods attribute, it add a separating space after the current member. A simple attribute that allows to logic separation of exposed members. In case of InspectorField containing nested items, the space is added after the collection.

Size

Determine the size - in pixel - of the space added after the current InspectorField.

```
[Inspect]
[Space(20)]
public float beforeSpace;

[Inspect]
public float afterSpace;
```

Before Space	<input type="text" value="0"/>
After Space	<input type="text" value="0"/>

Style

Fields, properties and methods attribute, this is used to modify the GUIStyle used for drawing the item.

Label

Boolean allowing you to hide the label of a InspectorField. Useful when drawing toolbar. Methods have their label hidden by default.

Name

Name of the GUIStyle. Internally, we do GUI.skin.FindStyle() to get the named Style.

```
[Inspect]
[Style("ToolbarSeachTextField")]
public float styledFloat;
```

Styled Float

0

Tab

Fields, properties and methods attributes, links those members to a collection of tab-like item in a toolbar at the top of the script. Similar to the Skinned Cloth component. Takes the value of an enum as input for each items.

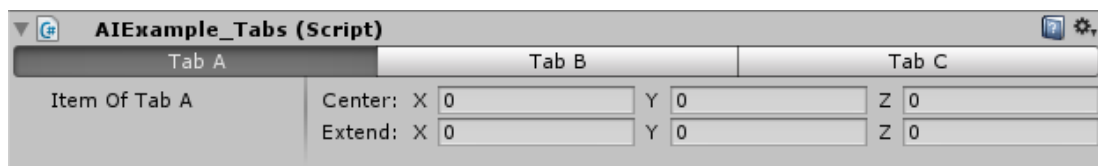
```
public class AIExample_Tabs : MonoBehaviour
{
    [Inspect, Tab(MyTabs.TabA)]
    public Bounds itemOfTabA;

    [Inspect, Tab(MyTabs.TabB)]
    public Material itemOfTabB;

    [Inspect, Tab(MyTabs.TabC)]
    public string itemOfTabC;
}

[Descriptor("GetDescriptor")]
public enum MyTabs
{
    TabA,
    TabB,
    TabC
}
```

Which gives;



It is possible to change the tabs name or icon, but it has to be done in a specific way. Please see the example in the package for binding an extension enum method to the Descriptor attribute.

TextField

Fields and properties attributes, modify how a string field is displayed.

Type

Define how the field is drawn. Options are;

- Standard; default text field.
- Area; expandable text field that react to line break.
- Password; field that display the text as "●●●●●●".
- Tag; drop down list of the existing tags.
- File; button with popup window for selecting file. Store the file path.
- Folder; button with popup window for selecting folder. Store the folder path.

Title

Title of the modal window invoke with File and Folder option.

Path

Default path when invoking File or Folder option.

Extension

Restrict selectable file type in the File option.

Toolbar

Fields, properties and methods attribute. Similar to a Group, but it draws the nested item as an horizontal toolbar.

Note that the toolbar does not change the Style of the nested member. For this reason, it is advices to flag the member with a custom Style.

Name

Name of the toolbar. Item having the same name are grouped.

Label

Draw the label - name - of the toolbar. Default is false.

Style

Style of the GUI group, the toolbar background. Default is "Toolbar".

Priority

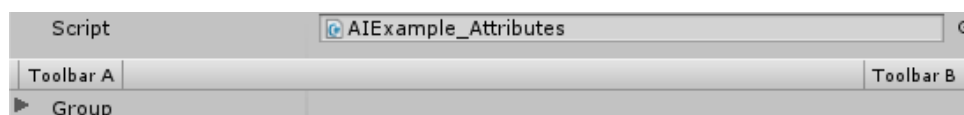
Priority of this toolbar, where it is displayed in the list of InspectorField.

Flexible

Allow an item to add Flexible Space before itself. Allows to stack items on sides of a toolbar and leaving a space in between.

```
[Inspect]
[Style("ToolbarButton", false)]
[Toolbar("Toolbar")]
public void ToolbarA() { }

[Inspect]
[Style("ToolbarButton", false)]
[Toolbar("Toolbar", Flexible = true, Style = "Toolbar")]
public void ToolbarB() { }
```



Contact

The Advanced Inspector was a huge undertaking, and for that reason the first few versions are expected to have hidden issues that we are not aware of. We will fix those bugs as quickly as we can, however we need to learn about them. To send us any bug report or feature request, contact us at;

Email : admin@lightstrikersoftware.com

Website : www.lightstrikersoftware.com

Package Revision

1.0:

- Release

1.1:

- Fixed the Camera preview (the window can be dragged around)
- Fixed the Camera perspective/orthographic drop down with new `RestrictedAttribute` option.
- Added `SkinnedMeshRenderer` and `Text Mesh` as converted editor.
- Added a way to decouple data from description in the `RestrictedAttribute`. Just pass `DescriptorPair` instead of the object directly.
- Added a missing constructor in `ReadOnlyAttribute`.
- Changed the constraint field in the `RigidBody` because "I prefer the checkboxes".
- Non-editable expandable field - such as object deriving from `System.Object` - now display info in the format `"ToString() [Class Name]"`. Overriding `ToString` becomes useful.
- Fixed a null error in the `Restricted > Toolbox`.
- `SizeAttribute` can now flag a collection as not-sortable. Useful for collection which the order comes from the code and should not be changed.
- Added a new Attribute; `RuntimeResolve` which display a field editor based on the object's type or a type returned by a delegate, instead of the field type.
- It can also be used to restrict a `Object` field to a specific type. For example, you can make a `UnityEngine.Object` field only display `Material`.
- Added a new Attribute; `CollectionConstructor` which let you create a collection item with the constructor and parameter of your choice.
- Moved the asset into `Plugins` and `Plugins/Editor` folder, so other language - like JS - can be supported without modification.
- Copied the `Examples` asset into `JavaScript`, also used to test other language support.
- Added a class named `"ActionBinding"` which is a way to bind an event to a data-driven method invocation. Similar to what Unity will roll out in 4.6 with their new GUI... but better.
- Take a look at the video for more information. It's an example of what the `Advanced Inspector` can do, as no editor were writing for that class.

1.2:

- Fixed an issue with `Editor` object showing the log message about no reference towards them when saving a scene. It was harmless, but annoying.
- Fixed an issue with internal texture showing the same log message.

- Fixed a index exception while deleting an item in some specific cases. It was harmless, but annoying.
- Fixed the multi-edition on boolean editor.
- Update some internal icon; +, -, open, closed and the drag icon. They should be easier to see in both light and dark.
- Added IPreview interface that gives control over the preview panel of the inspector. New class in the DLL; InspectorPreview.
- Added a new attribute; "Tab". Draws toolbar-like tabs at the top of a script, similar to the SkinnedCloth component.
- Added the standard "on scene" collider editor on the Box, Capsule, and Sphere collider.
- Added a class named TypeUtility. It is used only to retrieve a type in all loaded assembly based on its name.
- Added SkinnedMeshRenderer, TextMesh and Mesh editor.
- Added "very large array" support. Any collection having over 50 items only displays 50, and have arrows at the top and bottom to navigate in them.
- Added UDictionary, a fully Unity-serializable dictionary for Unity 4.5 and later. Does not work on previous Unity version.
- Added RangeInt/RangeFloat, struct used to set a minimum and a maximum. They use Unity's MinMaxSlider.
- The Space attribute can now add space before an item and/or after it. Before default size is 0, after default size is 1. Does not change existing behaviour.

1.3:

- **IMPORTANT:** All classes related to the Advanced Inspector have been moved to the AdvancedInspector namespace, for consistency with other assemblies (Ex.: System.Serializable) and not to pollute the global scope.
- The AdvancedInspector Attribute now have "InspectDefaultItem" property, which emulate the way Unity display items without the need to add [Inspect] on all your items. You can still inspect item Unity would not display by adding the Inspect Attribute.
- Help Attribute can now be tagged on a class, giving a help box in the header or footer. You can also place multiple instance of the Help attributes, which displays multiple help box.
- Help Attribute now has the "Position" property, which place the help box before or after the targeted member, or in case of a class, in the header or footer.
- Expandable Attribute now has the "Expanded" property, which forces an item to be expanded by default when first viewed.
- Group Attribute now has the "Expandable" property, which when false, the group is not collapsible and remains open, serving has a dividing box.
- Added a layout "header" and "footer" at the top and bottom of the inspector, before and after all the fields.

- Added an interface; `InspectorRunning`, which gives access to the Header and Footer zone in the form of a `OnHeaderGUI` and `OnFooterGUI` method.
- `InspectorField` now support `SerializableProperties`, but please do not use it, it's most likely full of bug since `SerializedProperty` is anything but generic.
- CTRL + Click expand all the child of the item expanded. (Ex.: All the subobject in a list)
- Removed the "[class name]" part of the label of a list when that item is the same type as the list itself. (Ex.: `List<item>`, no need to tell you it's an item)
- The collection type is now only display in Advanced or Debug mode.
- Removed the "shadows" from the following icon; add, delete, folder open, folder close.
- Fixed a null exception thrown when using enums in a Dictionary.
- Added cubemap to the supported type in the Preview.
- Reworked the layout a lot, such as adding 4 pixel to the left of the fields so they don't stick to the window. Now nested item are "boxed-in", which should make multi-level items easier to read.
- Lined up lot of items and subitem that were not properly lined up in specific case in both their label and fields.
- Fixed the Example6 where nested class where not serialized.
- Added methods `"GetAttributes"` and `"GetAttributes<T>"` to the Inspector field, returning all instance of a multi-attributes.
- Fixed the aspect ratio in the Camera preview on Scene.
- `SkinnedMeshRender` "Bones" property is no longer "read only", but it is now an advanced property.
- Added `LightEditor` to support Unity's Lights.
- Added support for the Gradient type, another of those fancy hidden thing in Unity.
- The cursor is now a "grab hand" when hovering over the drag control.
- Added an `EditorWindow` example of external inspector.

1.31:

- Added a type-check when displaying an object label to see if the type overload `ToString`, so it would be called only if a proper implementation exist, and not the base - `System.Object` - one.
- Inspector Level and Sorting are now saved in `EditorPrefs`.
- Expansion state are now saved in `EditorPrefs`.
- Dragging a label to another label automatically performs a copy-paste of that field.

1.32

- Fixed undo issue with restricted field.
- Fixed undo issue with struct such as `Vector`, `Quaternion`, `Bound`, `Rect`, `RectOffset`.
- Fixed undo issue with string.

- Added some shading to the expandable boxes.
- Added the GameObject/Component picker tool.

1.4

- Added a new attribute; Background, which color the box of expandable item.
- MaskedEnum attribute is replaced by the Enum attribute, which also controls how an enum is displayed.
- Merge the Size attribute with the CollectionConstructor attribute into the new Collection attribute.
- Toolbars are no longer using the toolbar style by default. You can do row of buttons this way.
- Toolbars are now drawn on the header, because they collided with the separator.
- Descriptor color now colors field instead of label. It made no sense to color the label, and was often hard to read.
- Descriptor with only a color no longer pass on an empty name to the label.
- Added a missing constructor in AdvancedInspector attribute.
- Expandable attribute now has the InspectDefaultItems similar to the AdvancedInspector attribute.
- Fixed an error raised when encountering a type with multiple overload of ToString.
- Fixed an issue when undoing creation or deletion of an item in a collection was not properly refreshing it.
- Fixed an issue that prevented proper undoing of collection reordering.
- Added multiple display option to the Enum attribute, see the EnumDisplay enum.
- Added multiple display option to the Collection attribute, see the CollectionDisplay enum.
- Added multiple display option to the Restrict attribute, see the RestrictDisplay enum.
- Added a "Collection Locked" option to the contextual menu, it locks every collection from adding/removing/sorting items.
- Added a Tutorial documentation, which gives steps by steps example of implementation.
- Fixed an issue that prevent copy pasting AnimationCurve.

1.41

- Fixed a Null Exception on the CameraEditor on Unity 4.6+.
- Fixed a Array initialization issue introduced with the changes to the Collection attribute.
- Fixed a ComponentMonoBehaviour destruction issue. Even while not referenced, the instance would fail to be destroyed.
- Fixed a collection failing to raise the Erase event on ComponentMonoBehaviour.
- Fixed a issue when a Dictionary would contain ComponentMonoBehaviour and would destroy them even when it shouldn't.
- Fixed a 4 pixels layout issue when multiple nested object would be part of a parent collection.

- Fixed an issue where multiple nested instance of the same type would prevent the child node from being expandable.

1.42

- Added a new interface; "ICopy" which give an object the power to handle how it should be copied over a targeted field.
- Added a new interface; "ICopiable" which allows an object to decide if it can be copied or not over a targeted field.
- The Inspector now takes the .NET ICloneable interface into account, allowing it to handle the copying. ICopy take priority over ICloneable.
- Fixed a recursive stack overflow in the copy/paste of a self-referenced object.
- Fixed a stack overflow in prefab comparaison with self-referenced object in editor mode.
- Added a "Take Screenshot" option in the Camera editor. It's available in advanced mode.
- The Collection attribute was missing the IRuntimeAttribute interface declaration.
- ActionBinding now properly sorts out properties, ignoring Getter when a Set is needed and vise versa.
- ActionBinging now flags Binding Parameter that are extra - not declared in the constructor - as being external or static, never internal.
- ActionBinging now control if it can be copied over, and what is copied. Same thing for BindingParameters. See ICopy/ICopiable.
- Fixed an issue when reloading the assembly context where Unity would "hang" for a few second while the Inspector rebuilds the type hierarchy tree.
- Added a missing construction in the Space Attribute.
- Fixed the stack overflow in the ComponentMonoBehaviour... again! What was I drinking?
- Fixed a object array initilization issue when using Collection(0) and a inlined field initialization.
- AnimatorEditor no longer display items twice.
- Added to the AdvancedInspector namespace the following; Toolbox, ModalWindow, WindowResult, IModal.
- The modal window are now draggable.
- Fixed the expansion of collection in Button mode.

1.43

- Fixed an expansion bug with groups.
- Fixed an expansion bug with multi-edition of collection and dictionary.
- Fixed an issue that would prevent ExternalEditor list from being sortable.
- Added the compile define ADVANCED_INSPECTOR to detect if the tool is installed or not.
- SHIFT+DRAG on labels of integer or float to change the value.
- DOUBLE+CLICK on labels expends or collapsed the item if it's expandable.
- DescriptorAttribute is now taken into account when affixed to an enum's value.
- The CollectionAttribute now has a "Enum Type" properties, which binds a collection to an enum's names. See the documentation for an example.
- Fixed an invalid index when a small array is turned bigger using the CollectionAttribute size property.
- Added missing attribute targets in a few Attribute so they could be added to structs.

- New Attribute; `MethodAttribute`, which gives control on how a method is invoked or displayed. For example, you can replace the button and draw whatever you want.
- `ExternalEditor` got more control over how it can be drawn; fixed separator or no space reserved for expander.
- The separator in the `ExternalEditor` is now uncoupled from the separator in the `Inspector`. Previously they shared the same settings.
- `Transform`'s global position/orientation are not read only anymore, and accessible in Advanced mode instead of Debug.
- `Expandable` attribute can now be placed on Interfaces. See `AIExample_Interface` for an example of an implementation.
- `CollectionDisplay` "default" value have been renamed to "List", to better reflect what it does.
- `UnityEngine.Component` that implement an interface can now be properly drag and dropped in a field of that interface type. See `AIExample_Interface`.

1.44:

- Made "SelectedTab" property public in `InspectorField`.
- Fixed an issue when mixing groups with tabs.
- Optimized the lookup of `FieldEditor` attribute. If you had slow down with this attribute, it should be fine now.
- Now able to inspect static field, properties and methods. Static fields and properties are read only, except when the game is playing.
- `InspectorField` now has a `Static` property.
- Added tooltips and URL to scripting documentation for the Camera, Animator, Light, and many other editors.
- Added editors for Animation, `Rigidbody2D`, `SpringJoint2D`, `Sprite Renderer`, and many other classes.
- Note that editors for 2D Physic (Ex.: `HingeJoint2D`) requires Unity 4.5 or higher.
- `InspectorField` now supports `SerializedProperty` arrays. Try to not break that feature.
- Fixed an issue with `IDataChanged` when the item inspected is at the root.
- Exception thrown on a method/delegate invocation now returns the inner exception, which would make debugging your own method a lot easier.
- `AdvancedInspector` attribute are now properly inherited in classes hierarchy.
- Enum drop down now properly uses `Descriptor` for their names.
- Removed the compile define, it didn't work from within a library.
- Fixed an issue that in some case an enum dropdown would become uneditable.
- Nicify enum variables.

1.5:

- Now support Unity 5.
- No longer support Unity 4.3, only 4.5+.
- Added the Watch window, which allows you to track inspector values. Right click any value and select "Watch".
- Added the Selected History. Use CTRL+Left Arrow and CTRL+Right Arrow to cycle in your previous selection.
- Added the Runtime Save feature; it allows you to save changed value while playing the game. This feature is in beta and may have issues.
- Redone all the examples, should be way easier to use.
- Added a collection of supported Unity type, like `Terrain Collider` and `Clothes`.

- Due to confusion and overlapping functionality, the Expandable attribute is now limited to fields and properties and only override the expandability of an item.
- Fixed an issue where sortable list would fail.
- Fixed an issue of recursivity with self-referencing objects.
- Fixed an issue with RuntimeResolved attribute in non-dynamic mode.
- Fixed the draggable icon in Unity Pro.
- Minor tweak to the expandable boxes visual; should look smoother.