User Manual

Contact us at support@illogika.com

Table of Content

Introduction

Heavy-Duty Inspector addresses the default Inspector's shortcomings with easy-to-use property attributes for your scripts.

Installation Instructions

Although importing this package in your project is enough to use all of its features, we have provided for your convenience script templates to create NamedMonoBehaviours, as well as new types of Keywords and the Keywords Config for each of these new types of keywords. To install them, simply use the Install Script Templates command under the File menu. This will add all text files found under the Assets/Illogika/ScriptTemplates folder to the currently running installation of Unity. (Windows users will have to run Unity as an administrator for this to work)

Using Heavy-Duty Inspector with UnityScript

For UnityScript to recognize the attributes in Heavy-Duty Inspector, the HeavyDutyInspector folder must be placed inside the Assets/Plugins/ folder to make sure it is compiled before the Javascript code.
You also need to add : import HeavyDutyInspector at the top of your scripts.

Then instead of using :

```
[ComponentSelection(System.Type componentType)]
```

use :

```
@ComponentSelection(System.Type componentType)
```

A last note. In javascript it is perfectly legal to add an Attribute at the top of a file, without it being followed by a variable. In this case the Attribute will be applied to the entire class (much like @pragma strict or @RequireComponent). Although this will compile just fine, the attribute has to be applied to a serializable variable to have any effect in the Inspector (either a public variable or a private or protected variable with the @SerializeField attribute.

General Instructions

To use any of these property attributes, simply add it before the variable you want to modify. Note that because of the way Unity's Property Drawers work, you can only use one of these attributes per variable.

```
[ComponentSelection]
public FakeState idleState;
```

It is also important to note that some of these attributes (like Button) don't really change the way the variable is displayed, but rather add something before it. It is done like this because property attributes have to be applied to a variable, and the only other way to do it would be to write a custom Inspector. Whenever this is the case, you will have the option to hide the modified variable.

```
[Button("A Button", "ButtonFunction", true)]
public bool hidden;
```

A Note on Decorator Drawers

Unity 4.5 introduced Decorator Drawers. Decorator Drawers differ from Property Drawers in the sense that they don't change the way the variable they are attached to is displayed. Instead they allow to display something else before the variable in the Inspector.

This is something that was lacking from the old Property Drawer system as some of the Attributes like Comment or Image had to be attached to "dummy" variables to work. While this is still the case with Decorator Drawers, you can add several of them to the same variable, or even add them to a variable that already has a Property Drawer.

```
[ComplexHeader("Organize your Scripts", Style.Line, Alignment.Center, ColorEnum.White, ColorEnum.White)]
[Comment("Headers come in two styles, Box and Line", CommentType.Info, 1)]
public string organizeAgain;
```

Since [HideInInspector] not only hides a field, but also prevents any of its Decorator Drawers from being called, I added a new attribute, [HideVariable] to allow attaching comments to invisible variables in the event you absolutely need it.

Also note that, while Buttons should be Decorator Drawers, for now they can't because Decorator Drawers currently don't have a reference to the MonoBehaviour they are displayed in, which prevents them from knowing whose function to call.

It is a known issue with Unity 5.0 that attaching a Decorator Drawer (like the Comment drawer) to a variable displayed with a Property Drawer that uses the default drawer implementation (like HideConditional) causes the decorator drawer to be called twice. In this case you should attach your comment to a hidden variable before the one with the Property Drawer. This has been fixed in later versions of Unity.

Key Features

NamedMonoBehaviours

NamedMonoBehaviours were created to help identify script references within an object. We have all ended up with an object full of instances of the same script, especially when designing a state machine, and the Inspector displays these script references in a way that is not helpful. You end up with countless variables, all with the same GameObject name and the same script name, until you no longer know which variable references which script. You could make a hierarchy of empty GameObjects to sort it out, but it is wasteful and time consuming to create and navigate through afterwards.

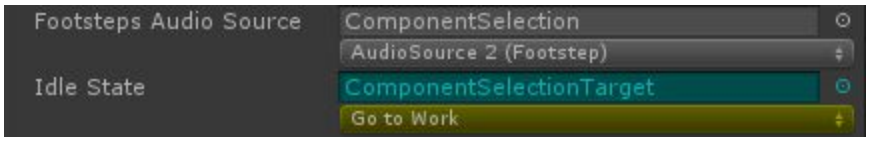
Using the NamedMonoBehaviour class, its Property Attribute and Property Drawer, you can now give a name to your scripts and have it displayed next to the reference in the Inspector.

Major Changes in Version 1.3

- In version 1.3, the ComponentSelectionAttribute now also displays the NamedMonobehaviour's color, rendering the NamedMonoBehaviourAttribute obsolete. You should now use ComponentSelection instead.
- Also, in version 1.3, a private serializable string named "typeName" was added to the NamedMonobehaviour class. The script will then store a textual representation of its type in the class' constructor, allowing you to know which type MissingMonobehaviours were, as long as you use NamedMonoBehaviour as the base class for all your classes, and use full text serialization. Of course this will only apply to script links that were not broken when you upgraded to version 1.3, but it should help you track down and fix broken links in the future.

[NamedMonoBehaviourAttribute]
This attribute is depreciated. You should use ComponentSelection instead.

Component Selection



One of the worst features or lack thereof of the default Inspector is the inability to choose which Component you want to select when the Component is on another GameObject. Also, it can become tedious to drag and drop your Components across an overcrowded Inspector several screens long.

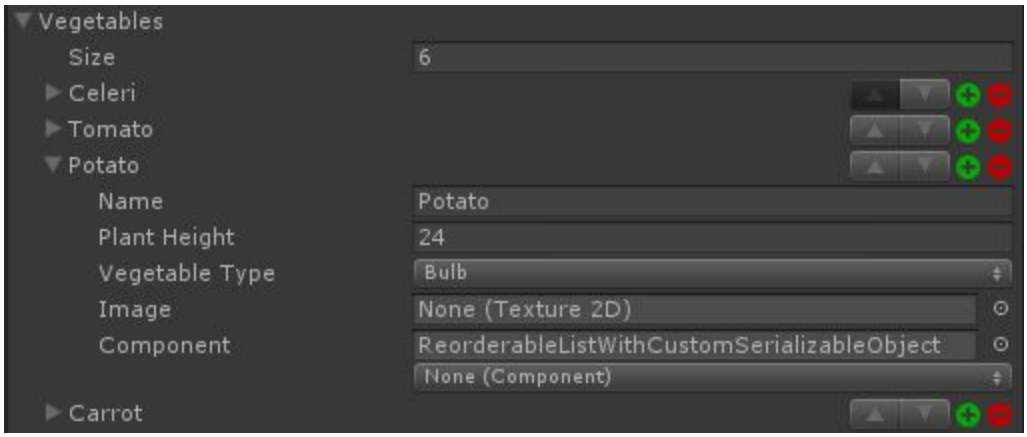
With the ComponentSelection Attribute, you can display a reference to a Component as a reference to its owning GameObject, followed by a popup box with a list of every Component matching the given type on the selected object. Most Components will be named after their type, and numbered, but NamedMonoBehaviours will display their full names (numbering will be added if duplicate names exist). You can also specify the name of a field you know your Component has and its value will be displayed in the list next to the Component.

To visualize your references even more easily, the object reference is displayed in YELLOW if it is null, and CYAN if it is on another GameObject.

```
[ComponentSelection]
[ComponentSelection(string fieldName)]
[ComponentSelection(string defaultObject, string fieldName)]
```

defaultObject: The name of a GameObject in your scene that will be selected by default.
fieldName: The name of a variable present in the Component whose value you want displayed next to the Component's name and numbering.

ReorderableList



[ReorderableList(bool useComponentSelectionDrawer = true)]
useComponentSelectionDrawer: whether or not you want Reorderable List use the ComponentSelection drawer (only applies with lists of objects that inherit from Component). If set to false, the list will use Unity's default Object selection for a more compact view.

Another flaw of the Inspector is that it does not take advantage of the full power of Lists. Both Arrays and Lists are displayed in the exact same way in the Inspector. With the ReorderableList attribute, you can move elements up and down the List, and even add or remove elements from anywhere in the List, not just at the end. It not only supports lists of custom serializable objects, but can also displays Component references using the ComponentSelectionAttribute UI.

[ReorderableArray(bool useComponentSelectionDrawer = true)]
There is also a version of Reorderable List to use with Arrays. Note that deleting the last element in the array will cause Unity to fire harmless error messages. There is currently no way around this.

For some unknown reason, Unity refuses to display the children of list elements if you display your list element with the PropertyField function. Because of this if you have a list of serializable objects, you cannot directly create your own property drawer for your object and use it in addition to reorderable list. You could however create a dummy containing object and create a list of that.

Example :

[System.Serializable]
public class A

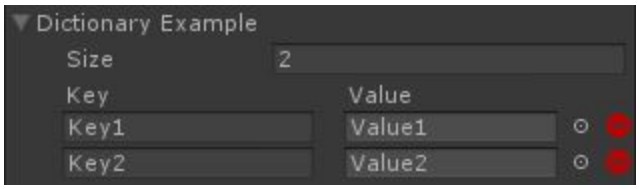
[ReorderableList]
List<A> myList;

cannot use a custom property drawer to display A

```
[System.Serializable]
public class A

[System.Serializable]
public class B
{
	public A myObject;
}

[ReorderableList]
List<B> myList;
```

can use a custom property drawer to display A

Dictionary



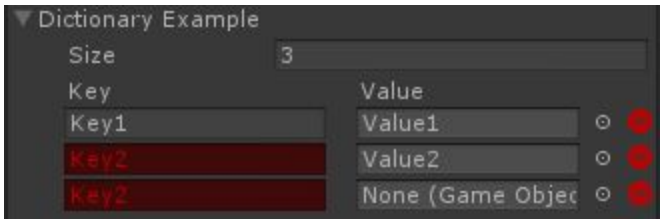
[Dictionary(string valuesListName)]
[Dictionary(string valuesListName, string keywordsConfigFile)]
valuesListName: the name of the List containing the values for the Dictionary.
keywordsConfigFile: the path to a KeywordsConfig file to be used as Keys for your dictionary This is relative to the Resources folder.

In Unity 4.5, Unity added the interface ISerializationCallbackReceiver in an attempt to circumvent the limitation preventing Dictionaries from being serialized. What it basically does is give you two callbacks, one right before the object is serialized, to enable you to convert a Dictionary into two lists, and another one that is called right after deserialization, to convert two Lists into a Dictionary. Although better than nothing, Unity didn't provide us with any way to ensure both lists are the same size, or to display both list side by side, like a dictionary should.

To improve upon this a convenience method was added in NamedMonobehaviour to help with this conversion, and the new DictionaryAttribute was created to help sync the two lists, and make the values appear beside their keys in a single unified display of a Dictionary.

The DictionaryAttribute goes on the list of Keys, and points to the list of values. The list of values should have the HideInInspector attribute applied so it doesn't appear twice.

Duplicate keys in the dictionary are highlighted in red.



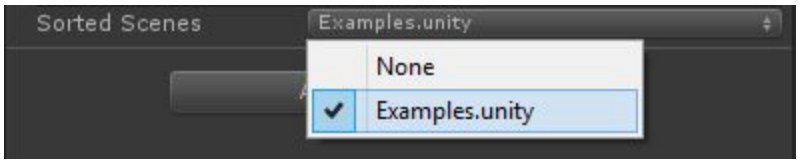
Wrappers

Scenes

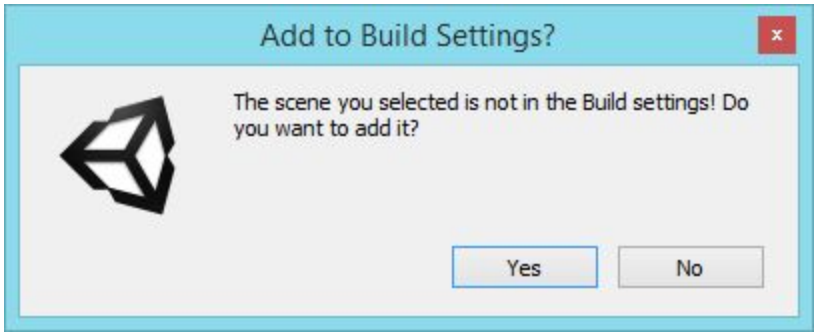
Scene is a serializable class that converts implicitly to and from a string and has its own Property Drawer. What it means for you is you can do something like this :

```
public Scene nextSceneToLoad;

...

Application.LoadLevel(nextSceneToLoad);
```

Returning only the name of the scene (without its extension or containing directory) exactly what LoadLevel expects, acting in the way YOU would expect a Scene reference to work.

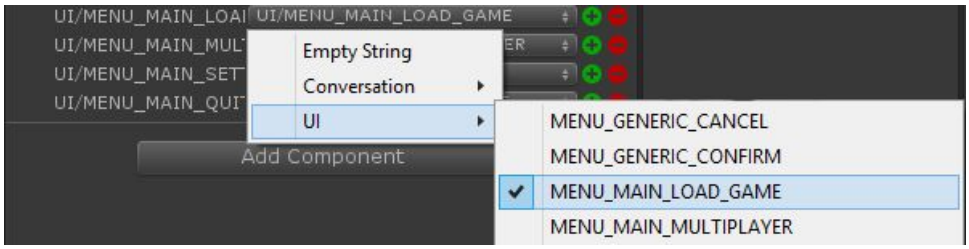


In the inspector, your "nextSceneToLoad" variable will be displayed as a Drop Down menu showing all the scenes in your project, all neatly sorted in sub categories named after their containing folders. And if the scene you selected was not in the build settings, you will have the option to add it automatically.



Alternatively you could use the [Scene(string basePath)] attribute on your Scene object to specify a folder where you want the editor to look for your scenes.

Keywords and the Keywords Editor

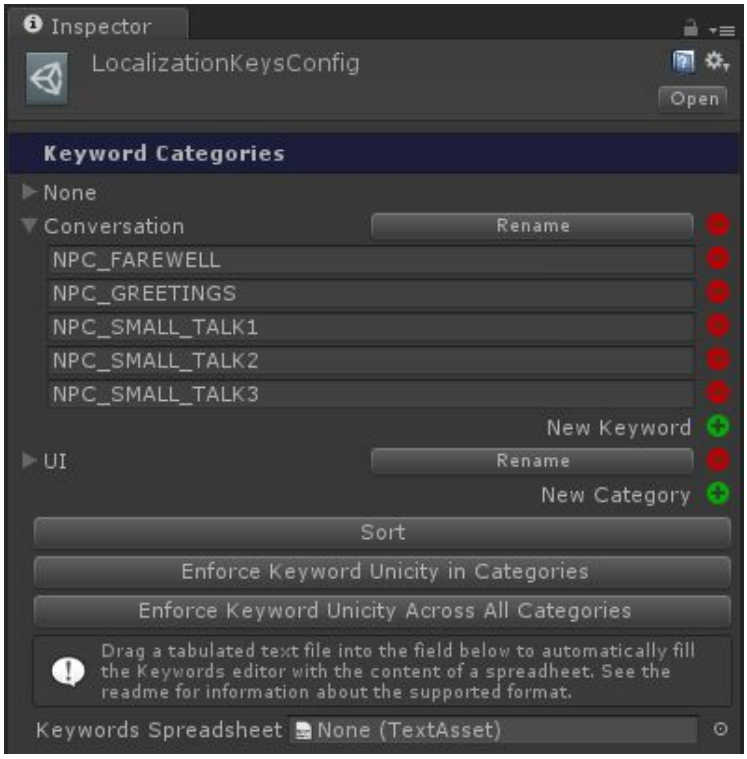


A Keyword, like a Scene is a serializable class that converts implicitly to a string and has its own Property Drawer. It allows you to select a string in the Inspector in a drop down menu from a categorized (in sub menus) list of strings instead of typing it. You can even add new strings to an existing category directly from the Keyword inspector.

Alongside that, there is the KeywordConfig Scriptable Object (which you can create from the Assets menu, under ScriptableObjects)

This asset allows you to set the categories for your keywords and add or remove keywords in them. To create subcategories, just add slashes (/) as separators in the category name.

It also has the following options:



Sort : Sorts the keywords alphabetically inside each category.

Enforce Keyword Unicity in Categories : Deletes duplicate keywords from each individual categories.

Enforce Keyword Unicity Across All Categories : Finds keywords that are duplicate in different categories and gives you the choice as to which category should keep the duplicate keyword. Duplicate in other categories are deleted.

There is also an option to load all your keywords directly from a spreadsheet. Simply export your spreadsheet to a tab separated text format (in .txt) and drag the text asset in the corresponding field.

The parser assumes the first element at the top of each column is the name of a category, and that each subsequent element in the same column is a keyword that belongs to this category.

You can also create your own Keywords that each use a different Config file using the two new script templates provided. The first one (C# Custom Keyword) should be placed alongside the rest of your scripts,

while the second (C# Custom Keyword Editor) should be placed in an Editor folder. Both should have the same name for the script template to generate the classes correctly, and your scripts should not have a plural name (as one of the classes it generates will have an 's' added at the end.

You can refer to the LocalizationKey example provided.

Component Field

The ComponentField type allows you to select any child property or field of any component, going as deep as you want through its children, and retrieve its PropertyInfo or FieldInfo at runtime to change its value.

You can also use the ComponentFieldRestriction attribute to restrict which type can be the end property or field, or which type of component can be selected.

```
[ComponentFieldRestriction(System.Type endMemberType)]
[ComponentFieldRestriction(System.Type endMemberType, System.Type componentType)]
```

If an end property type is specified, the last selected member will be displayed in red if it is not of the right type, and in green if it is of the right type. If your last selected member is of the right type, no more child member drop downs will be displayed after it.

SType

The SType wrapper (for SerializableType) allows you to serialize a System.Type in Unity. In the inspector, it is presented as a field in which you can drag a script.

You can also use the TypeInspector attribute to specify a parent type. In this case, the field will be displayed as a pull down menu with a list of all the types that inherit from the specified type. You also have options to ban abstract classes from this list, to ban classes with a specific substring in their name, or ban specific type names.

Serialization Wrappers (Depreciated... (Mostly))

Since Unity 5 now supports serialization for all primitive types, most of these have been depreciated. They are still included for backward compatibility.

Unity's drawer for UInt64 is broken though, so I replaced it with my own. (If you've ever tried to use a UInt64, you might have noticed that UInt64.MaxValue displays as -1 in the Inspector… For an UNSIGNED int.

Also, Undo doesn't work for Int64 and UInt64 in Unity even with my custom drawer, so you might still want to use my wrappers instead. I think this is an issue with Unity's undo feature, not the Drawers themselves though.

Below is the original documentation :

This only applies if you are still using Unity 4, or were using these wrappers in a Unity 4 project that you upgraded to Unity 5. You shouldn't use these in a new Unity 5 project as Unity now supports serialization for these types by default.

Several primitive numeric types are not currently supported by Unity's serializer. Currently, only ints, floats and doubles are serialized. Shorts and longs are not, neither are any unsigned variations of numeric types, and none of them have an Editor function to draw them in the inspector.

With version 1.2, Heavy-Duty Inspector adds wrappers for these types. They are classes that you can use as if they were of the type they wrap, and have an implementation of their drawer to be displayed in the inspector. They are named after the type they replace, with an S at the end (for Serializable) :

Int16S
Int64S
UInt16S
UInt32S
UInt64S

NB. : Since UInt32, Int64 and UInt64 can hold values outside the range of an int, I had to implement their drawers using a TextField. You might notice that you can type non numeric characters into these fields, although the field will correctly reject them when you press enter or deselect the field. There is currently no way around this.

More Property Drawers

AssetPath

[AssetPath(PathOptions pathOptions)]
[AssetPath(System.Type type, PathOptions pathOptions)]
type: The type of asset that can be dragged into the reference box.
pathOptions: The way your path should be formatted. Relative to the Assets folder, relative to a Resources folder and with no extension, or just the filename.

Display a string as an object reference in the Inspector. Never have to type your assets' paths again.

Background

[Background(ColorEnum color)]
[Background(float r, float g, float b)]

Adds a solid color background to the affected variable. Useful to make Serializable Objects appear as a foldable section Header.

Backgrounds are always displayed last, after all other DecoratorDrawers to make sure they are applied to the variable, not another DecoratorDrawer.

Button

[Button(string buttonText, string functionName, bool hideVariable = false)]
buttonText: The text displayed on the button.
functionName: The name of the function to call.
hideVariable: Whether or not you want to display the variable this attribute is applied to.

Add a button with a function callable by reflection. Pass the function's name to the constructor as a string. You should not use this attribute with Arrays or Lists.

ImageButton

[ImageButton(string imagePath, string functionName, bool hideVariable = false)]
imagePath: Path to your image. This path must be relative to your Assets folder and include the file's extension.
functionName: The name of the function to call.
hideVariable: Whether or not you want to display the variable this attribute is applied to.

Add a button with a function callable by reflection. Pass the function's name to the constructor as a string.

You should not use this attribute with Arrays or Lists.

ChangeCheckCallback

[ChangeCheckCallback(string callbackName)]
callbackName: The name of the function to call when the value of the variable changes.

Use this attribute to call a function when the value of its variable changes.

This attribute needs to start a coroutine to work correctly and will not work on classes extending ScriptableObjects as they lack the ability to use a coroutine.

Comment

[Comment(string comment, CommentType messageType = CommentType.None, int order = 0)]
comment: The comment to be displayed.
messageType: The icon to be displayed next to the comment. This is the same as the Editor enum MessageType.
order: The order in which to display DecoratorDrawers, from smallest to largest.

Add a comment to explain variables to your game designer. Code comments are not read by designers and are sometimes ignored by programmers. Add a comment in the Inspector for everyone to see without the hassle of writing a custom Inspector.

Comment attribute uses a DecoratorDrawer, so it is now safe to apply it to Lists and Arrays. You can also add it multiple times on the same variable.

CompactView

[CompactView(HeaderStyle headerStyle, DisplayStyle displayStyle, params string[] relativeProperties)]
[CompactView(string keywordFilePath, DisplayStyle displayStyle, params string[] relativeProperties)]
headerStyle: Should a header be displayed, and if yes, what should it contain. If it displays a variable, it will be the first one specified in the relativeProperties
displayStyle: How to display the elements in the compact view. Side By Side or each on a line, with or without labels.
keywordFilePath:If the variable displayed in your header is a keyword, the path to its KeywordConfig file.
relativeProperties:Names of the variables to display in the CompactView, in the rder you wish them to appear.

Displays the content of a serializable object (especially one that contains lists and is itself inside a list) in a more compact manner than the default sea of foldouts you would normally get.

ComplexHeader

[ComplexHeader(string text, Style style, Alignment textAlignment, ColorEnum textColor, ColorEnum backgroundColor)]
[ComplexHeader(string text, Style style, Alignment textAlignment, float textColorR, float textColorG, float textColorB, float backgroundColorR, float backgroundColorG, float backgroundColorB)]
text: The text to be displayed in the header.
style: The header style, either Box or Line.
textAlignment: The text's alignment, either Left, Centered or Right.
textColor: Either an enum with the standard colors or three floats for the R, G, and B values of the Text color. This is because Attributes can only be constants and the Color class cannot be a constant.
backgroundColor:Either an enum with the standard colors or three floats for the R, G, and B values of the Text color. This is because Attributes can only be constants and the Color class cannot be a constant.

Displays a header in the inspector for easy categorization of variables. The header can either be of style "Box", with the title displayed over a solid color background, or of style "Line" with the title being surrounded by a line.

Headers are always displayed first, before any other DecoratorDrawer.

DynamicRange

[DynamicRangeAttribute(float min, string maxDelegate)]
[DynamicRangeAttribute(string minDelegate, float max)]
[DynamicRangeAttribute(string minDelegate, string maxDelegate)]
min: Minimum value.
max: Maximum value.
minDelegate: A variable or function returning the correct numeric type that will set the minimum value.
maxDelegate: A variable or function returning the correct numeric type that will set the maximum value.

Works like Unity's default Range Attribute, but you can set the minimum and the maximum using a variable or a function that returns the correct numeric type.

Editable Comment

[EditableComment]
[EditableComment(ColorEnum headerColor)]
[EditableComment(float r, float g, float b)]
headerColor: Color the for the comment header's background.

Displays a string as a comment that can be edited in the inspector. (must be attached to a string)

EditableProperty

[EditableProperty(string accessorName)]

Apply with [SerializeField] to a private variable to access it in the inspector using its getter and setter. Both need to exist for this to work. If the name of your accessor is the same as the name of your variable minus the prefix and underscore, you don't need to specify the name.

IE. if your private variable is m_myVariable, and your accessor is named myVariable, you don't need to specify any parameter.

EnumMask

[EnumMask]

Displays an enum as an enum mask instead. Your enum must start at 1 and have bit shifted values for this to work.

Unity stores enums and enum masks as ints, but enum masks only really make sense as unsigned ints. You might observe that settings the values of an enum mask manually doesn't produce the same result as setting it to everything, and then removing some of the elements. This is because setting an enum mask to Everything, will set all of its bits to 1, even if the corresponding flag doesn't exist in the enum, which will result in the value of the enum mask to turn negative. To help with this, we provide the ToMask() extension function to convert int32s into uint32s.

Hide Conditional

[HideConditional(bool hide, string conditionMemberName)]
[HideConditional(bool hide, string conditionVariableName, params int[] visibleState)]
[HideConditional(bool hide, string conditionVariableName, float minValue, float maxValue)]
[HideConditional(bool hide, string conditionMemberName, string comment, CommentType messageType)]
[HideConditional(bool hide, string conditionVariableName, string comment, CommentType messageType, bool visibleState)]
[HideConditional(bool hide, string conditionVariableName, string comment, CommentType messageType, params int[] visibleState)]
[HideConditional(bool hide, string conditionVariableName, string comment, CommentType messageType, float minValue, float maxValue)]

hide: Whether the variable should be hidden or displayed until the condition is met.
conditionMemberName: The name of the member to use as a condition. If the member is a boolean the condition will be that it is true. If the member is a nullable variable, the condition will be that the variable's value is not null. Alternatively, you can specify the name of a parameterless function which returns a boolean and do some advanced logic in there instead. This is useful if you are testing against the state of an enum mask, as using the overload with the int[] params can quickly get crowded.
conditionVariableName: The name of the variable to use as a condition.
visibleState: The state the condition variable needs to be in for the variable to be displayed. This can be either a boolean or a list of integers corresponding to the values of an int or enum.
minValue: The minimum value the condition variable can be for this variable to be displayed.
maxValue: The maximum value the condition variable can be for this variable to be displayed.
comment: A comment that will be displayed if the variable is visible.
messageType: The icon to be displayed next to the comment. This is the same as the Editor enum MessageType.

Hide a variable in the Inspector until another variable has the specified value. The condition can be that the variable is not null, that it has a specific boolean value, that it has a specific int value among a set of int values (this can be the int value of an enum), or has a float value between a min and a max or that a parameterless function returns true. You can also optionally display a comment since the Comment attribute does not work well with HideConditional.

HideVariable

[HideVariable]

Like [HideInInspector] but doesn't prevent DecoratorDrawers from being called.

Image

[Image(string imagePath, Alignment alignment = Alignment.Center, int order = 0)]
imagePath: Path to your image. This path must be relative to your Assets folder and include the file's extension.
alignment: The image's alignment, either Left, Center or Right.
order: The order in which to display DecoratorDrawers, from smallest to largest.

When a comment is not enough, a picture is worth a thousand words. Add a diagram to the Inspector, or maybe just a logo that visually represents the class function.

The image attribute uses a DecoratorDrawer, so it is safe to apply it to Lists and Arrays. You can also add it multiple times on the same variable.

InterfaceTypeRestriction

[InterfaceTypeRestriction(Type interfaceType, bool allowSceneObjects = true)]
interfaceType: type of your interface. It doesn't really have to be an interface, but if you specify a class, it won't do anything that Unity doesn't already do.

Allows you to restrict the type of what can be dragged in a field to an interface. Unity still won't serialize interfaces so you will have to attach this attribute to a variable whose type Unity can serialize (like MonoBehaviour or ScriptableObject), and cast it to the right interface when you want to use it.

Readonly

[Readonly]

Displays a variable in the inspector, but doesn't allow changing its value. The variable will be displayed with the inspector's lock icon to its left. You can always set the variable's value by switching your inspector to Debug mode, but this should prevent unintentional tampering of important variables.

ReadonlyProperty

[ReadonlyProperty(bool hideVariable, params string[] properties)]
hideVariable: Whether or not you want to display the variable this attribute is applied to.
properties: The name of the properties you want to display, ordered in the order you want them to appear in the inspector.

Display readonly and computed properties in the inspector. Attach this Property drawer to a variable to have your properties displayed after this variable. The order in which the properties will appear is the same as the order of the Attribute's parameters.

Readonly Properties are displayed with a darker background to differentiate them from Readonly Variables.

ReservedSpace

[ReservedSpace()]
[ReservedSpace(float space)]
space: the vertical space in pixels to leave before the next variable.
If you don't specify the space, the default space is equal to the height of a single line in the inspector.

The reserved space drawer allows you to reserve an empty space in the inspector when calling the DrawDefaultInspector function. You can later get a list of all the Rects associated with these reserved spaces by calling ReservedSpaceDrawer.spaceRects.

With this, you can create a custom Inspector without having to add your custom parts at the end of the default inspector, or needing to re-write the whole inspector.

Scene

[Scene(string basePath)]
basePath : the base path (relative to the Assets/ folder) where the drawer should look for scenes to populate its list.

Although the Scene class has its own drawer, it can sometimes be useful to get a subset of the scenes in the project, instead of all the scenes in the project.

Tag

[Tag]

Display a string using the Tags drop-down menu in the Inspector. It is both convenient and prevents typos.

Tag List

[TagList(bool canDeleteFirstElement = true)]
canDeleteFirstElement: Whether or not you want the first element of the list to be deletable.

Display a List of strings using the Tags drop-down menu in the Inspector. This also allows you to delete tags from anywhere in the list.

Layer

[Layer]
Display an int using the Layers drop-down menu.

Change Log

V1.4

- Added the CompactView attribute to display small serializable objects in a way that's visually clearer than a ton of small foldouts.
- Added the InterfaceTypeRestriction attribute that allows you to restrict the type of an object reference to an interface type. Unity will still not serialize interfaces, so the variable it is attached to needs to be an object type, and you need to cast it yourself to the right interface.
- Added the ReservedSpace attribute that allows you to reserve a space when drawing the default inspector, and later request its Rect to draw something else in a custom inspector. Allows you to mix the ease of use of Heavy Duty Inspector with the power to create your own custom inspectors.
- Images used by the property drawers are now moved to the Editor Default Resources when you first install the plugin, instead of being loaded with Resources.Load.

V1.34

- Added the ReadonlyProperty Drawer that allows you to display Readonly properties in the inspector.
- Added the EditableComment drawer, that allows you display a string as a comment that can be edited in the inspector.

v1.33

- Readme is now in PDF format
- Added the ReorderableArray attribute.
- Added the possibility to reverse the behaviour of HideConditional
- Added using a delegate method to get the hide state for HideConditional
- Added an EditableProperty drawer to edit a variable through its getter and setter.
- Added a ReadOnly drawer to display a variable in the inspector but not allow changing its value.
- Added a ComponentField wrapper that allows you to select any child field or property of a component.
- No longer restore the object reference for FilenameOnly asset paths as this was causing significant lag in larger projects.
- When adding a new keyword directly from the Keyword drawer, changed the Plus and Minus buttons to clearer Check and X buttons with tooltips (Confirm, Cancel)
- Fix an exception in dictionary when adding a first element after deleting all elements.
- Changed the way dictionaries are initialized to fix a bug where they would not initialize.
- Several other small fixes
- Several new examples in the example scene.

v1.32

- First official Unity 5 version.
- Removed #ifdefing for old versions of Unity for code clarity.
- Added a Dynamic Range Drawer that acts like Unity's Range drawer, but with variable minimums and maximums. These can either be a variable or a function returning the right numeral type.
- Added the possibility to add an optional comment to HideConditional, since the Comment drawer cannot work with HideConditional.
- Fixed a bug preventing Dictionary Drawers to properly display lists of Keywords used as Keys.

- Dictionary Drawer now highlights in Red duplicate keys. This works best with strings, Keywords, and primitive types, as any other type is compared by reference and not by content.
- Added summaries to DictionaryAttribute's and EnumMaskAttribute's constructors.
- Fixed a bug where the height of UnityEvents would not be properly calculated in HideConditional and reorderable lists.
- Fixed a bug where the height of variables hidden with HideConditional would not be properly calculated in another object that's conditionally hidden or in a reorderable list.

v1.31

- Foldouts inside objects inside reorderable lists now properly fold out.
- Reorderable Lists now properly work when nested.
- Comment Drawers now properly calculate their height when inside a serialized object used in a reorderable list.
- Adding an element in a reorderable list of serialized objects now properly copies the values in the selected object.
- Fixed the Dictionary Drawer wrongfully throwing an uncaught exception when resizing a Dictionary in the Editor in Unity 5.
- Fixed the Dictionary Drawer to properly initialize the values when the value list is of primitive types or a type that doesn't have a default parameterless constructor.

v1.3

- Added the new DictionaryAttribute to take advantage of Unity's ISerializationCallbackReceiver interface.
- Fixed a bug that prevented drawers that use reflection to work with arrays.
- Fixed a bug where Buttons, ImageButtons and ChangeCheckCallbacks would not work properly when used inside a custom serialized classes because they would look for their function in the MonoBehaviour.
- Improved HideConditional so it can properly display custom serialized Objects.
- Improved the way Serialized Classes are drawn to prevent a bug where Vectors would be displayed, followed by their inner properties, resulting in a Vector3 taking up four lines, one for the vector itself, than three more for duplicate display of its X, Y and Z properties.
- Added a tooltip to ComponentSelection explaining how it works.
- Changed the color scheme on component selection to something more neutral than the old Green and Yellow. YELLOW is now used for null reference, self reference is WHITE (default color), and reference on another object is CYAN, thus removing the Good/Bad bias there used to be.
- ComponentSelection now displays a NamedMonobehaviour's color rendering the old NamedMonobehaviour drawer obsolete. It has been left for backwards compatibility but might be removed in a future release.
- ComponentSelection now keeps a reference to the last selected GameObject, even if you set its Component to null. This reference is somewhat persistent through your Unity session, but is not serialized and will be reset if you restart Unity or when Unity compiles its scripts.
- Component selection will now default its GameObject reference to null if the selected object has no valid Components.

v1.2

- Reorderable lists now work even when inside a Serializable Object or another List.

- Added serializable classes that wrap Int16, Int64, UInt16, UInt32 and UInt64 and implicitly convert to their wrapped type, to allow serialization of these primitive types that are currently not supported by Unity's serializer.
- Added Scene class, an attribute and two drawers for it. Scene works like a reference to a scene, even though it really is a string to work with the LoadLevel functions.
- Added the Keyword class to select strings from categorized lists, and the KeywordConfig class to edit this keyword list.
- Added two new script templates to generate new classes that inherit from Keyword and use a KeywordConfig file with a different name, allowing different types of strings from being clearly separated from one another. ie LocalizationKeys for string localization vs ModelNames for models that will be assigned to a prefab at runtime.

v1.11
- Added support for private or protected fields to the Drawers that were using reflection. You still need to specify [SerializeField] for Unity to actually serialize it, or even call the Drawer.

v1.1
- Updated Heavy Duty Inspector for Unity 4.5
- Removed TextArea in Unity 4.5 as it has been added by Unity. It is still present if you are using an older version of Unity.
- CommentAttribute and ImageAttribute now use the new DecoratorDrawer, which means you can apply several of them to the same variable, and even add them to a variable that already uses a Property Drawer.
- Added the HideVariable attribute, which does the same thing as HideInInspector, but without preventing DecoratorDrawers from being called.
- Added the ComplexHeader attribute to display headers in the inspector that catch the eye better than the basic HeaderAttribute introduced in Unity 4.5
- Added a Background attribute to add a solid color background to a variable.

v1.06
- Fixed a bug where Component Selection and Reorderable list would not serialize their object when changing the target object for component selection, or when reordering elements in the list without making any changes to the value of these elements.

v1.05
- Added support for classes extending ScriptableObject.
- ComponentSelection now selects the first component matching the type by default when the target object has changed instead of keeping a null value.

v1.04
- Moved all Attributes and Drawers to the HeavyDutyInspector namespace to prevent name clashing with other plugins.
- added "using HeavyDutyInspector;" to the script template.

v1.03
- Added the TextArea attribute.
- Fixed a bug with ImageButtons not displaying their attached variable correctly.
- Fixed a bug with height calculation for Rect types

Known Issues:

- There appears to be a bug with the EditorGUI.PropertyField function that prevents it from displaying Quaternions properly. This prevents Buttons, Comments and Images from displaying Quaternions in Unity 4.3+

v1.02

- Added the ImageButton attribute. Works like the Button attribute, but takes the path to an image instead of the button's text as a parameter.
- Added a parameter to the Image attribute to Left, Right or Center align the image.
- Fixed a bug that would stretch images horizontally.

v1.01

- Initial Release