Attributes

TilePlus Script Attributes

If you've done any Unity coding then you're probably familiar with Attributes by now, such as those having to do with serialization, or those affecting inspectors. The normal Unity Tile Selection inspector only displays the fields of the Tile class and no others. But we'd like to be able to view and modify fields and properties from TilePlusBase-derived tiles.

The Tile+Brush and Tile+Painter Inspectors and underlying library functions control what you see when using the Selection and Brush inspectors. It's a fair amount of IMGUI code. But normal humans should not have to struggle with that. So here in TilePlus land, we have several new Attributes which can be added to your code to display fields, properties, and even invoke methods or provide custom IMGUI code for functionality that the inspectors can't handle.

The Selection Inspector and the Brush Inspector (and their associated equivalents in Tile+Painter) use these attributes to display simple fields and properties. Fields can be modified, and the changes are saved in the Scene. Like any other change made to a scene, you need to save the scene to persist the changes. Normally, the Configuration Editor sets "Autosave" on and the save is done automatically for every change you make.

Your original TPT tile in your Project folder will never be altered. Note that field, property, and method declarations need to be public or protected to work with these attributes.

Attribute	Affected	Description
[TptShowField]	Selection Inspector	The types of fields that you can use this on are bool, int, float, string, Color, Vector2, Vector3, Vector2Int, and Vector3Int. Ints and floats can optionally use range sliders.
[TptShowEnum]	Selection Inspector	Show Enums in a pop-up.
[TptShowObjectField]	Selection Inspector	Objects such as GameObjects can be referenced with this attribute.
[TptShowAsLabelSelectionInspector]	Selection Inspector	For a field or property, the value returned is displayed with ToString, so whatever you mark with this attribute must have a ToString() method or return a string.
[TptShowAsLabelBrushInspector]	Brush Inspector	Same as above.
[TptShowMethodAsButton]	Selection Inspector	Invoke a method, see below.
[TptShowCustomGui]	Selection Inspector	Create your own IMGUI function
[Tooltip]	Selection Inspector, Brush Inspector	This is a normal Unity attribute that you can find in the scripting reference. Note: Fields only.

Attribute	Affected	Description
[TptNote]	Selection Inspector	Add a note to a field or method. Note can be a static string or be provided by a property.

If a tooltip is provided as part of any attribute, then any normal [Tooltip] attribute will be ignored.

Display Order

The display formatter organizes the various attributes in the same order as the class hierarchy for the tile. For example, a TpFlexAnimatedTile tile shows information from TpFlexAnimatedTile followed by TilePlusBase.

In each section, Attributes are processed in the following order:

- Properties with TptShowAsLabelSelectionInspector or TptShowAsLabelBrushInspector.
- Methods with TptShowCustomGui
- Methods with TptShowMethodAsButton
- Simple fields with TptShowField
- Enum fields with TptShowEnum
- Object fields with TptShowObject field

Common Parameters

Common Parameters

These parameters apply to all Attributes except TptShowAsLabelBrushInspector.

Param	Value	Description
spaceMode	enum SpaceMode	Adds space or a line before or after the item, or both before and after.
showMode	enum ShowMode	Controls visibility: Always, only when playing, only when not playing, or use a property.
visibilityProperty	string	When showMode == Property then this is the name (in quotes) of the property is used to control visibility. If the property returns true, then the item is shown. Prepending a ! character to the name of the property inverts the property's value.

About "Show as Label" Attributes

- You can create a property just to show it as a label, based on other values in your class. See "Tips," below.
- The Brush Inspector is only affected by `[TptShowAsLabelBrushInspector]`, hence, it can only show strings for both fields and properties. This is because what's displayed is information about the actual asset, and that should only be changed from the Project window and a normal Unity inspector.
- These attributes have a few extra parameters:
  - useHelpBox: show as an IMGUI HelpBox rather than a label.
  - splitCamelCaseNames: controls how field or property names are displayed.
  - toolTip: a Tooltip for the field or property.

About TptShowCustomGui

This attribute is placed above a method in a tile's code. The method should use IMGUI to display whatever information desired. Note that the method name itself is irrelevant.

The method should have a signature like this: public CustomGuiReturn CustomGui(GuiSkin,Vector2,bool)

The GuiSkin may be useful in the method, and the Vector2 has button sizes. You may or may not need these. The boolean value is true if the tile is in a prefab and shouldn't be edited. You can use this to display a warning.

If the access is not public or protected, then the method will be ignored. The return value from the custom GUI method indicates if any fields were modified, whether the tile should be refreshed, and whether to force a scene save.

It’s not advised to make custom GUI methods virtual, but if you do, please use ‘new’ rather than ‘override’ in subclasses as using override will make the generated GUI appear in the same foldout section as the overridden method.

About TptShowField for int and float fields

TptShowField’s min and max parameters control whether ints and floats display editable fields or slider controls. If these are both zero (the default) then normal editable fields are displayed. If not, then a slider using those two values will appear. When the slider is visible and the slider values change, the configuration setting “AutoSave” is ignored, as that results in way too many scene saving invocations as the sliders are moved. The configuration setting “Allow sliders” can be unchecked to revert to editable fields. Using these parameters with other types of fields has no effect.

How to Trigger Methods

The TptShowMethodAsButton attribute is very handy for debugging. If you place this attribute right above a public or protected method, a button will appear which you can use to Invoke the method from within the Selection Inspector. The buttons won’t appear if the tile is simulating. Note that if the method access is private then the method is ignored.

This comes with one restriction: the method must have no parameters, and you’ll see an error message in the Selection Inspector if the method has parameters. You can find numerous examples of this technique in the supplied TPT tiles.

About Object Field Attributes

The TptShowObjectField attribute requires a specification for Type. For example:

```
[TptShowObjectField(typeof(GameObject))]
public GameObject m_AGameObject;
```

This field can be used to drag Scene or Project references into fields for GameObjects, Components, Transforms, Prefabs, etc., when the Selection Inspector is focused on a TPT tile.

Another important parameter for this attribute is allowSceneObjects. If this is true, the Object field can include scene objects. Set this parameter to false if the tile will end up in a Prefab or will be archived in a TpTileBundle.

Using TppShowField with Unsupported Types

If the TppShowField attribute is used on an unsupported Type or custom Type, an Object picker is shown, or in some cases just the ToString() representation of the object. If the field represents a UnityEngine.Object (for example, a Scriptable Object) then an Open Inspector button will appear.

For example, TpFlexAnimatedTile uses this feature to display the name of the AnimationClipSet. Since that’s a Scriptable Object asset, the button opens an Inspector where you can edit the list of sprites. Note that the list of sprites is a normal asset, so if you change anything it affects any and every TPT tile that uses the asset.

Field, Enum, Object special feature

The Field, Enum, and Object attributes have another parameter: updateTpLib. The default value is false. If set true and the field, enum, or Object is changed in the Selection Inspector or TilePlus Utility then TpLib.UpdateInstance is called. This feature was created to accommodate multiple tags for a tile, and TpLib.UpdateInstance handles that automatically.

To support custom use of this feature, during Editor sessions, TpLib.UpdateInstance writes an array of field names that were modified to the overridable property UpdateInstance, although it’s unlikely that there’d be more than one field name.

You can use .Contains on the received array to see if there’s a field you’re interested in if you ever need to use this. Examples can be found in TpFlexAnimatedTile.cs and TpSlideShow.cs, where changes in the sprite and slide assets require some internal updates to the tile instance.

In Play Mode

The ShowMode value for attributes can be used to control what is displayed when in Play mode. However, some features automatically change in Play mode or when a TilePlus tile is Locked.

- Unsupported field editor buttons become unavailable.

- All fields display as IMGUI Helpboxes, except if you set ForceFieldInPlay = true
  - Note that ForceFieldInPlay is intended for development only and depending on the field Type may have unintended results including crashes.

If you have any concern that the display might affect your playing app, either ensure that the Tile+Painter window is closed, and that the Selection Inspector isn’t focused on a tile or use the Configuration Editor (Tools/TilePlus/Special/Configuration Editor) to toggle the “Safe Play Mode.”

Notes

The '[TppShowAsLabel...]' attributes allow you to display a property. This can be useful in many ways. Here’s an example from TilePlusBase.cs that shows how the Tag property as shown in the Brush Inspector works.

```
/// <summary>
/// Property to get the optional tag
/// </summary>
/// <value>The tag.</value>
[TptShowAsLabelBrushInspector(true)]
public string Tag
{
    get => string.IsNullOrEmpty(m_Tag)
               ? string.Empty
               : m_Tag.Trim();
    set
    {
        #if UNITY_EDITOR
        if (!TpLib.IsPlayMode)
            #endif
            m_Tag = value.Trim();
    }
```

It’s not a great idea to do anything too complex since there’s repeated access during an Editor session when the Selection Inspector or TilePlus Utility window are in use.

Another way to add a display string is to use the TptNote attribute, for example:

```
[TptNote(false, "Only X and Y are used")]
public Vector3 m_EndSize = new(2, 2, 1);
```

This attribute is only used in the Selection Inspector.