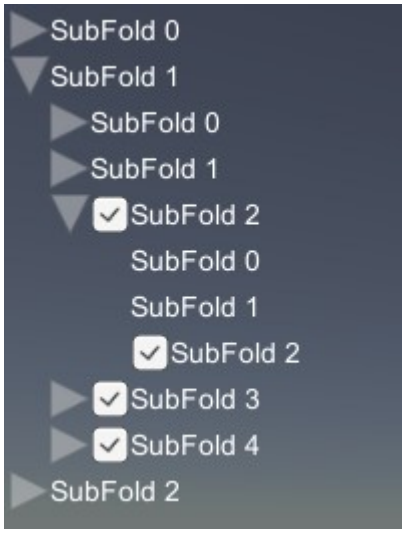
Proposition One Script-Based TreeView Control for Unity UI

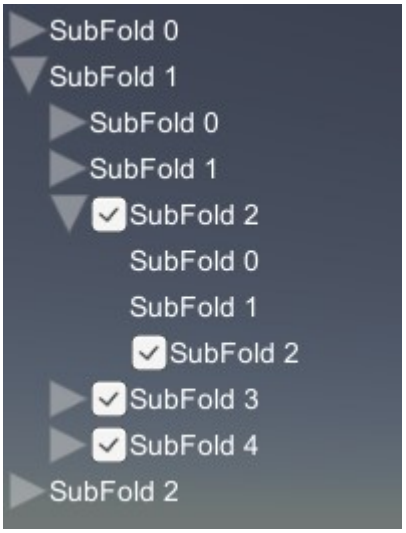
Instructions



Proposition One
2015
http://propositionone.wordpress.com/
PropositionOneSW@gmail.com

Table of Contents:

Introduction:

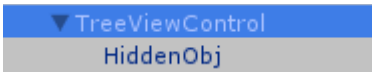


The TreeView control is a runtime-oriented control for use with Unity 3D UI system from versions 5.0 and up. The control is designed for easy-access, script-based creation, manipulation, destruction and interaction of a TreeView hierarchy.

The concept is a simple TreeView with drop down customizable objects which can be manually designed and implemented. Custom modules can be scripted with relative ease and implemented at runtime to allow user interaction.

Usage:

Core Prefab (TreeViewControl):

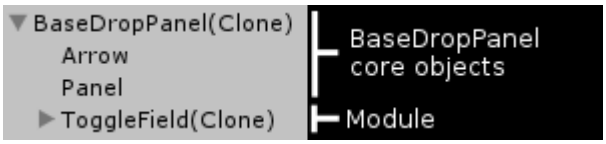


The "TreeViewControl" prefab is the root element of the TreeView control object. This is a panel consisting of two scripts and a background image module to provide contrast against anything the control is rendered over. The two scripts attached to this object form the basis of interaction, creation, and deletion of the hierarchy for the TreeView.

The "HiddenObj" is a place for "invisible" SubFolds to be placed off screen. Objects in the hierarchy are always extant as GameObjects, when a branch is folded, it's children are moved off to the "HiddenObj" located about 40,000 units away.

(Note: if you have the option selected for this root object to autosize itself relative to the treeview, it is necessary for the anchors to have zero difference due to the usage of the sizeDelta method.)

BaseDropPanel:



The base drop panel is the "container" for any modules you would like to appear in your hierarchy. TreeView comes with, by default, only two modules: A toggle module and a text module. Other modules may be designed and implemented by the user provided they are correctly scripted and assigned to the prefab dictionary by key. Creating modules is covered later.

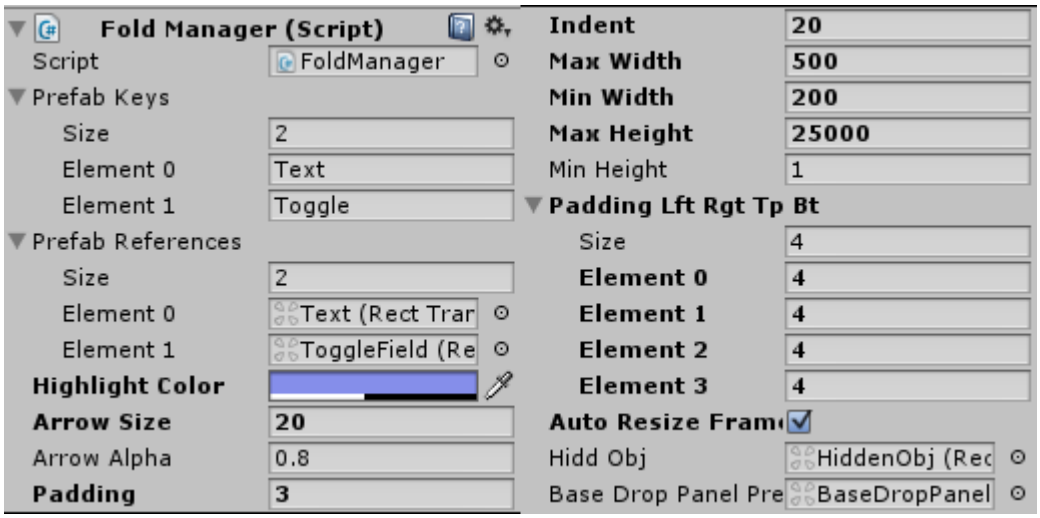
The "**Arrow**" object of the BaseDropPanel is a simple toggle that may be switched at will provided there is a tree to expand attached to the SubFold. Under the "ArrowScript" the images used for the "Arrow" may be changed to suit the user's desires.

The “**Panel**” object does very little except function as a background highlighter for the module and otherwise acts as an aid in sizing the module.

Scripts:

I. FoldManager.cs

FoldManager.cs is the core script for all subunits of the hierarchy, one exists on every “subfold” that is created. The primary one, located on the “TreeViewControl” prefab, controls the overall options for the hierarchy.

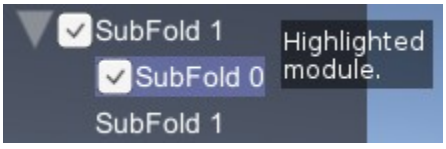


Options:

Prefab Keys/Prefab References:

These two List<T>'s make up the keys and prefab references of the modules of the treeview. Modules are assigned to a Dictionary<string, Transform> upon Awake() of the script. Due to the fact that Unity does not allow assignment of Dictionary objects in the inspector, the keys and values of this dictionary are split into two List<T>'s for easy assignment of modules. The “Text” key in “Prefab Keys” corresponds to the “Text(RectTransform)” of “Prefab References” and the “Toggle” key corresponds to the “ToggleField(RectTransform).” Customized user modules must be added to these lists with an appropriate key to allow for script creation of the hierarchy.

Highlight Color:



The highlight color is the default color that is used when a module is highlighted. How to highlight a module will be explained later. This highlight color can also be individually overridden for each individual module instance if so desired.

Arrow Size:

All modules must have the same height in the TreeView, the "Arrow Size" option sets this height as well as the size of the Arrow toggle object which will always be a square object matching the height of the SubFold module.

Arrow Alpha:

When a SubFold has child SubFolds, the arrow will be enabled with this alpha level. If the SubFold has no child SubFolds, no arrow image will be displayed.

Padding:

This is the spacing that will be placed between each subfold allowing for a dense or sparse appearance to the treeview.

Indent:

The indent to be applied to each subsequent level of the hierarchy. It is recommended that this be a value close to or matching the Arrow Size.

Max Width:

When auto-resize of the TreeViewControl is enabled, this is the absolute maximum that it will resize the panel to. Furthermore, no SubFold will ever exceed this value in width measured from the far left of the TreeViewControl object. If the treeview control is placed in a scroll rect, it is recommended to set this to a very large number to ensure absolutely all parts of the hierarchy can be properly displayed and scrolled to.

Min Width:

When auto-resize of the TreeViewControl is enabled, this is the minimum width the TreeViewControl panel will be sized.

Max Height:

Same as Max Width, except with height.

Min Height:

Same as Min Width, except with height.

Padding Lft Rgt Tp Bt:

This array must be kept at *four* or greater indexes in order for the script to work. The four indexes represent the offset from the border of the TreeViewControl unit and the elements in the hierarchy of the TreeView. "Lft" is short for left, "Rgt" for right, "Tp" for top, and "Bt" for bottom. All elements in the TreeViewControl panel will never exceed these margin limits.

Auto Resize Frame:

This determines whether or not you want to script to automatically resize the TreeViewControl frame. Best used when the control is in a ScrollRect to allow dynamic resizing of the scroll bars relative to the displayed expansion of the hierarchy.

HiddenObj & Base Drop Panel Prefab:

These are references to the prefab objects necessary for creation and usage of the TreeView. DO NOT CHANGE THESE. If they are somehow lost, HiddenObj is the "HiddenObj" child of the TreeViewControl and BaseDropPanel Prefab is the "BaseDropPanel" object.

II. TreeViewMain.cs

```
public class TreeViewMain : MonoBehaviour {

    Internal

    #region Object Accessors/Manipulators

    /// Adds a new object to the TreeView. Returns A...
    public FoldObject CreateObject(string Path, object[] Parameters, string TypeKey)...

    /// Returns the extant FoldObject for the designated...
    public FoldObject GetExtantObject(string Path)...

    /// Returns an array of FoldObjects representing the...
    public FoldObject[] GetSubfoldsInPath(string Path)...

    /// Sorts the TreeView according to the (simple) ...
    public bool SortOrder(FoldObject[] ObjectsInOrder)...

    #endregion
```

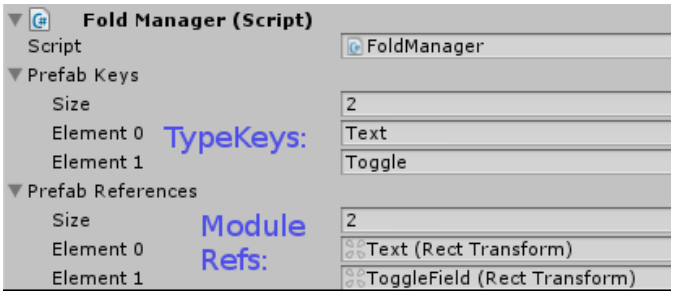
All SubFolds of the TreeView must be dynamically created via script. TreeViewMain.cs is the primary interaction point for creation and manipulation of the TreeView. There are four methods available in this script and the script is located on the TreeViewControl object.

All SubFolds are arranged by a **Path**. A path must satisfy the following requirements:

1. Consist solely of alphanumerics. There may be no underscores, spaces, slashes, or any other punctuation as part of path names; only A-Z, a-z, and 0-9 are permitted. Paths are case-sensitive. Periods are used to designate hierarchy.
2. As stated above, SubFold names must be separated by periods. A proper path may look something like this: "RootObject0.SubFoldA.SubSubFold1" *(Continued on next page)*

3. Before creating a sub-branch of an object, that object must be created first. Observing the image pictured right we may assume the highlighted object is at the path "SubFold0.SubFold0.SubFold0". "SubFold0.Subfold0" may not be created after the highlighted fold. Similarly, "SubFold0.SubFold0" may not be created before "SubFold0". i.e.: Any parent object in the hierarchy must exist before its children can be created.

All SubFolds have a "**TypeKey**" that must be used to create them. This is a string which refers to the "TypeKey" designated on the FoldManager.cs script. The TypeKey corresponds with the module object reference in the Prefab References List<Transform>. There are two preincluded modules in this package: "Text" and "Toggle".



Methods:

This leads to a description of the functions in TreeViewMain.cs. The first and foremost function is:

FoldObject CreateObject(string Path, object[] Parameters, string TypeKey);

"**Path**" is described above. It is the "name" of the object to be created including its parents' names. The "Path" of the highlighted SubFold shown in the above picture is "SubFold0.SubFold0.SubFold0."

"**Parameters**" varies depending on the type of module you are adding to the tree. It is an array of objects in this method to allow maximum versatility in creating modules. For the included simple modules "Text" and "ToggleField," the parameter types are "object[]{ string, Color, int, FontStyle, Font }" and "object[]{ bool, string, bool, Color, int, FontStyle, Font }" respectively.

For the **"Text" module**, there are 5 parameters: the string to display, the Color, the FontSize, the FontStyle, and the Font . This module is customized to take at least 1 parameter and up to 5. Therefore the parameters may be, for example: new object[] { "TextToDisplay" }, or new object[] { "TextToDisplay", Color.blue, (int)16, FontStyle.Bold } or any number of the full 5 parameters. "null" may be chosen to selectively not change a parameter in the array.

For the **"Toggle" module**, there are seven paramters: the state of the check, the text to display, whether or not the checkbox is "enabled," the Font Color, the (int) Font Size, the FontStyle, and the Font. Therefore the "Parameter" array for "CreateObject(...)" can be "new object[]{ true/false, "Text To Display", true/false }". Or it may be a longer array to accommodate customizations to the font.

"**TypeKey"** is described above. It is the key specfied which represents the module desired for this particular SubFold. "Text" is the string to create a "Text" module, and "Toggle" is the string to create a "ToggleField" module.

To create the root SubFold0 object pictured above, the following method would be invoked:

```
CreateObject("RootDesignation", new object[]{ "SubFold0"}, "Text");
```

To create its child, which is a toggle, the following method would be invoked:

```
CreateObject("RootDesignation.SubDesignation", new object[]{true, "SubFold0", true}, "Toggle");
```

FoldObject GetExtantObject(string Path);

This method returns a "FoldObject" already existent object in the hierarchy. "FoldObject" will be covered later, but it is the primary class for interacting with individual modules. You must utilize the full Path to retrieve the object.

FoldObject[] GetSubfoldsInPath(string Path);

This method returns an array of objects attached to a specific parent SubFold designated by the Path. If I have a root SubFold located at the path "RootSubFold," this method, with the path "RootSubFold," will return any child SubFolds of that. Similarly, passing a blank ("") string will pass back the child objects of the root. This method is best used in conjunction with SortOrder(...)

bool SortOrder(FoldObject[] ObjectsInOrder);

This method will rearrange the objects in *one* subfold into the passed order starting with index 0 at the top. For the method to function, all extant subfolds must be a part of the array passed to the method. This method returns false if the sorting attempt failed for any reason. This may be because an incorrect number of FoldObjects were passed or because one or more of them was not present in the same the path.

void SuspendLayout();

Suspends layout of control for adding or removing large amounts of subobjects. Each time an object is added a new layout is performed so adding many tree objects can cause unnecessary lag.

void ResumeLayout();

Resumes layout of control after a call to SuspendLayout(). Immediately lays object out if modifications have been made.

III. FoldObject.cs

```
4  public class FoldObject
5  {
6      /// Used internally
7      public delegate void ParameterChangeDel(object[] Params);
8      /// Used for the "ParametersChangedByUser" event.
9      public delegate void ParameterChangeNameDel(object[] Params, FoldObject Object);
10
11     /// Passes the parameters and the FoldObject of an...
14     public event ParameterChangeNameDel ParametersChangedByUser;
15
16     #region Basic Interactions
17     /// The prefab Transform of this module (Do NOT change) ...
20     public KeyValuePair<string, Transform> MyType;
21     /// The full path of this module (Do NOT change) ...
24     public string Name;
25     /// The name of this module without its parent path...
28     public string NameSimple;
29     /// The path of the parent of this module. (Do NOT change) ...
32     public string Path;
33     /// The transform for the module itself (e.g. a...
36     public Transform ChildObj;
37     /// The transform of the "BaseDropPanel" of this...
40     public Transform Obj;
41     /// The FoldManager script of this module. (Do NOT change) ...
44     public FoldManager DataManager;
45     /// The VisualManager of this script. (Do NOT change) ...
48     public DropControl VisualManager;
49     #endregion
50
51     /// Gets or sets a value indicating whether this...
55     public bool Highlight...
73
74     /// Color for overriding the default highlight color...
77     public Color HighlightColor = Color.black;
78     /// Set this to "true" to use the override color on...
81     public bool HighlightColorOverrideDefault = false;
82
83     /// Gets or sets the parameters of the module...
87     public object[] Params...
100
101    /// Destroys this object and all of its children on the tree...
104    public void Destroy()...
115
116    /// Used interally to pass events from the module to...
120    public void InternalParameterSet(object[] Params)...
127 }
```

FoldObject.cs forms the base class for interacting with individual modules and the primary means of "bridging" the GUI and the scripted logic of individual objects on the TreeView.

Events:

event ParameterChangeNameDel ParametersChangedByUser;

Assuming a module can both receive user input and is correctly scripted to relay events upon change, this event will trigger when the user has changed the module in some way allowing for script consideration of the changes. It passes an object array containing the parameters of the module the FoldObject controls as well as the FoldObject itself to identify the sender.

Variables:

KeyValuePair<string, Transform> MyType;

This is the base "TypeKey" and Prefab for this module. This is used internally and otherwise can serve as a reference to differentiate the type of the module the FoldObject controls.

(*For the following variables, the following full path will be used as an example:* "RootFold1.SubFold1.SubSubFold1"*)*

string Name;

The full path of the SubFold object. From the path above, this would be the string: "RootFold1.SubFold1.SubSubFold1".

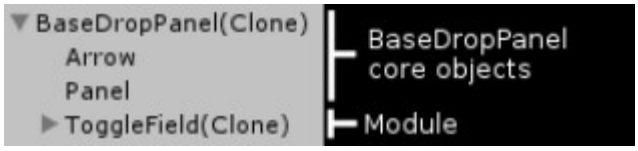
string NameSimple;

The "SimpleName" of the object. It is the name of the module this FoldObject controls without the path. From the path above, this would be the string "SubSubFold1".

string Path;

This is the path of the object without the simple name, or the name of the modules parent ("" if it is a root object). From the path above, this would be the string: "RootFold1.SubFold1".

Transform ChildObj;



This is a reference to the "Module" object of the referenced SubFold. This can be used to retrieve specific scripts from the module for more fine-tuned interaction with the module itself.

Transform Obj;

This is a reference to the BaseDropPanel object of the SubFold. It is not recommended that the user make any alterations to this object as they may interfere with the internal logic of the TreeView.

FoldManager DataManager;

This is a reference to the FoldManager script of the BaseDropPanel. It is not recommended that the user utilize this script without careful examination and/or alteration of the script to fit his or her personal needs. This script manages layout and subfolds of the TreeView and is used internally.

DropControl VisualManager;

This is a reference to the DropControl script of the BaseDropPanel. It is not recommended that the user utilize this script without careful examination and/or alteration of the script to fit his or her personal needs. This script contains some of the layout methods of the BaseDropPanel and is used internally.

Color HighlightColor;

If one wishes to override the default color of the module's highlight, this is the place where that color may be set.

bool HighlightColorOverrideDefault;

Toggles whether or not the HighlightColor variable is used for overriding the highlight color. Changes to the highlight color will be made on the next toggle of the “Highlight” property.

Properties:

bool Highlight{ get; set; }

Toggles highlighting on or off for this module.

object[] Parameters{ get; set; }

Gets or sets the parameters of the module which will vary depending on the module type.

Methods:

void Destroy();

Destroys the object and any children that branch off of this SubFold. This is the method that MUST be used to destroy objects on the TreeView.

void InternalParameterSet(object[] Params)

An internal method used to connect module parameter change events to the “ParametersChangedByUser” event. This method is of no conceivable use to the outside user.

IV. IModuleInterface.cs

```
4  public interface IModuleInterface  {
5
6      /// Occurs when parameters change...
9      event FoldObject.ParameterChangeDel ParameterChange;
10
11     /// Gets or sets the parameters of the embedded object...
15     object[] Parameters {get; set;}
16
17     int HeightAllowance { set; }
18
19     /// Returns the desired width for the object...
23     int DesiredWidth {get;}
24
25     /// Called when the control resizes...
28     void SizeEvent(RectTransform Parent);
29
30 }
31
```

IModuleInterface.cs is the most important class in ***designing new modules*** for the TreeView. Any module (e.g. "Text" and "ToggleField") in the TreeView must contain a MonoBehavior script which also inherits from IModuleInterface on its root object.

Events:

event FoldObject.ParameterChangeDel ParameterChange;

This event should be implemented so as to pass the parameter array of objects to it whenever a user changes part of the GUI. For example, if one creates an InputField module, when the user presses enter, it should fire this event allowing the script to process the parameter change. On modules like "Text" which have no parameters that can be changed by the user on the GUI, this method does nothing but still must be declared in all modules.

Properties:

object[] Parameters { get; set; }

This property should allow the getting and setting of whatever parameters are desired for runtime access to the module's settings. It takes an object array with unlimited amounts of parameters that may be set and got. It is *recommended* that any implementations of this do type checking on the object array as well as permitting null objects in the array which do not change the module's settings. When one accesses the "Parameters" property in FoldObject.cs, it accesses this property on the IModuleInterface interface.

int HeightAllowance { set; }

When laying out modules, every module will be sized to the exact same height. If your module has child objects that must be sized in proportion to the height of the module, it is recommended to implement this property's functionality. This will be called by the layout routine prior to sizing the module. It will pass the height of the module to the module so the module will be able to determine its desired width. After the HeightAllowance has been called, the layout script will try to get the "DesiredWidth" property from the IModuleInterface.

int DesiredWidth { get; }

Closely related to the above property, this property is requested by the layout routine prior to resizing the root of your module. Simply pass whatever width you calculate your module will need to this and the layout module will attempt to size the module to that width. If the DesiredWidth exceeds the maximum width available on FoldManager.cs – it will go with the Maximum Width property. This property allows modules to not take excessive width to the detriment of the look of the TreeView. Because the DesiredWidth may depend on the height of the module, the "HeightAllowance" property above is passed to the IModuleInterface prior to asking for the DesiredWidth so the IModuleInterface can determine the desired width based with respect to the height the module will be sized to.

(For example, the "ToggleField" module's checkmark is designed to scale to the square of the height of the module resulting in a very large square check box for a large module height and a very small square check box for a small module height. Because the checkbox must remain square, the Text module may be further left or right depending on a small or big height for the module which will vary the DesiredWidth of the module)

WARNING: you must pass something back to this, even if it is int32.maxvalue, forcing every module to take up the MaxWidth of the TreeView. Behavior is undefined with a null return.

Methods:

void SizeEvent(RectTransform Parent);

This method is called after the DesiredWidth property. If you have child components to your module, here is where you will adjust their size and position relative to the resizing of the module. Your root module object should NOT be re-sized as that is handled by the layout routine, however, the "Parent" RectTransform which is passed into this will give you size of your module allowing you to layout any child elements of the module's root to your desire relative to the actual size of your root module object.

For example, the "ToggleField" object has a blank panel serving as it's "Root Module Object." This is the object to which the IModuleInterface script is attached. This panel has a "Toggle" child object and a "Text" child object. When "SizeEvent" is called by the layout routine, the "ToggleField" module adjusts the size and positions of its child "Toggle" element and child "Text" element to fit the "Parent" RectTransform.

Included Modules:

“Text” --- Key: “Text” --- Prefab: “Text”

This is a regular text label with no user interaction possible. It has 5 parameters for its parameter array:

object[] { string TextToDisplay, Color FontColor, int FontSize, FontStyle Style, Font FontType }

It is presumed these are self explanatory. A “null” reference may be used for any of them except “TextToDisplay” when passing a parameter array to not change the current setting of the module. Furthermore, the array may be any size from 1 to 5 in length as long as the parameters always occupy the same index.

“Toggle” - Key: “Toggle” --- Prefab “ToggleField”

This is a standard toggle module that consists of a toggle check mark and a non-interactive text field. It has a 7 parameter array:

object[] { bool IsChecked, string TextToDisplay, bool CheckUIEnabled, Color FontColor, int FontSize, FontStyle Style, Font FontType}

It is once again presumed these are relatively self-explanatory. “CheckUIEnabled” simply toggles the functionality of the checkmark on or off. A “null” object may be used in any array element to indicate the value should not be changed. The array passed may be of any size from 1 to 7 provided all passed objects occupy its proper index in the array.