Extending Unity

How to create tools for you and your designers inside Unity Editor

Robot Gentleman

About Me

- My name is Mateusz Pusty!
- Unity Developer @ Robot Gentleman
- Worked on *60 Parsecs!*
- Co-organizer of Poznań Unity User Group meetings.
- Organizer of PGG Jam: All Play - Accessibility

mateusz@robotgentleman.com

@mateuszpusty

60 Parsecs!

60 Parsecs! is an Atomic Space Age adventure of scavenge and survival. Keep your crew alive and ready for action. Make difficult choices, face soup shortages and other horrors of outer space. And maybe reach your destination. Or not.

Who is this for?

- Everyone who wants to learn about writing Editor Scripts.
- Developers that know the basics of Unity and want to extend their skillset.
- People that want know where to look for further information about Unity Editor.

Who is this not for?

- Masters of writing Editor Scripts that already know all of this stuff. :)

But you're welcome to stay and correct me!

Agenda

1. Introduction to Editor Scripting
2. Present - IMGUI
3. Future - UI Elements
4. Tips & Tricks
5. Noteworthy Materials & Assets

Introduction to Editor Scripting

What is Editor Scripting?

Editor Scripting is a way to make the development of your game bit easier.

"You can use editor scripting inside Unity to make life easier for your game designers, or even yourself. With a small amount of code, you can automate some of the more tedious aspects of using the inspector to configure your behaviours, and provide visual feedback on configuration changes."

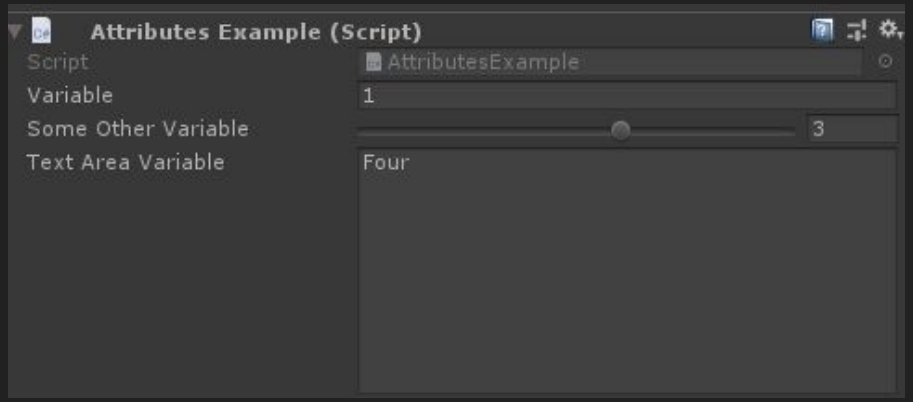
---

https://unity3d.com/learn/tutorials/topics/scripting/introduction-editor-scripting

Basic Editor Scripting

Using Built-In Attributes

```
public int Variable = 1;

[HideInInspector]
public int SomeVariable = 2;

[SerializeField]
[Range(0, 5)]
private int _someOtherVariable = 3;

[SerializeField]
[Multiline(10)]
private string _textAreaVariable = "Four";
```

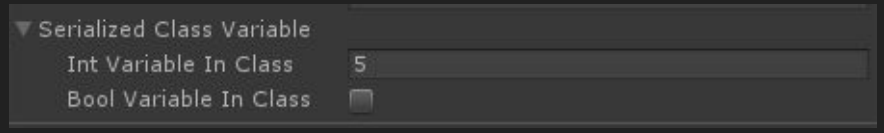


https://docs.unity3d.com/Manual/Attributes.html

Basic Editor Scripting

Using Built-In Attributes

```
[Serializable]
public class SerializedClass
{
    [SerializeField]
    private int _intVariableInClass = 5;

    [SerializeField]
    private bool _boolVariableInClass = false;
}

[...]

[SerializeField]
private SerializedClass _serializedClassVariable;
```

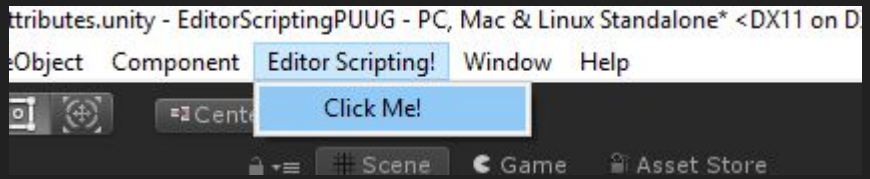


https://docs.unity3d.com/Manual/Attributes.html

Basic Editor Scripting

Using Built-In Attributes

```
[MenuItem("Editor Scripting!/Click Me!")]
public static void CustomMenuEntry()
{
    Debug.Log("I am invoked from custom menu entry!");
}
```



https://docs.unity3d.com/Manual/Attributes.html

Basic Editor Scripting

Scriptable Objects

Scriptable Objects are asset files that can be used to store data outside of class instances and MonoBehaviours. Because they are not MonoBehaviours they cannot be attached as components but their instances are stored as assets in our project instead. This makes them great data aggregators as they are independent from scenes. They are also not reset when exiting Play Mode therefore making them persistent between sessions. They will also persist their values between scenes, but they won't persist it if you exit and re-enter builded game.

---

https://docs.unity3d.com/Manual/class-ScriptableObject.html

Basic Editor Scripting

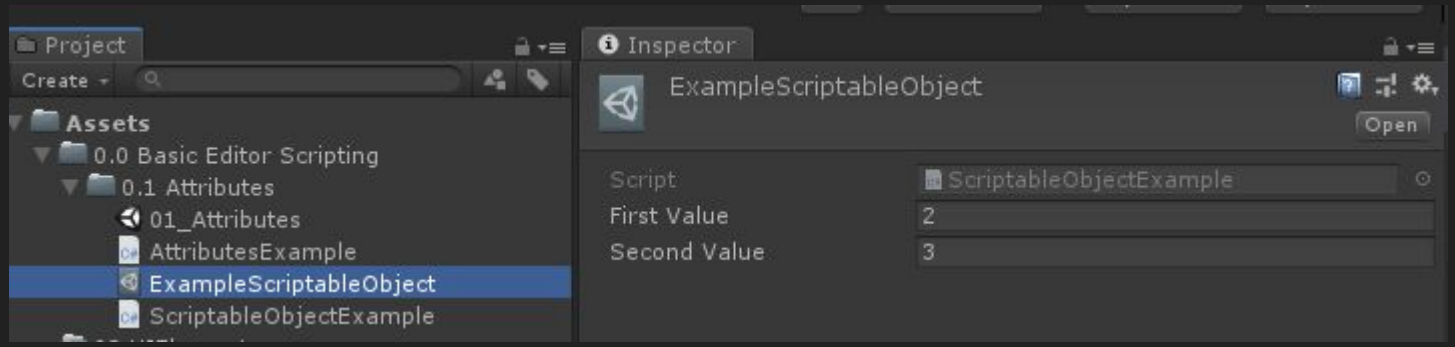
Scriptable Objects

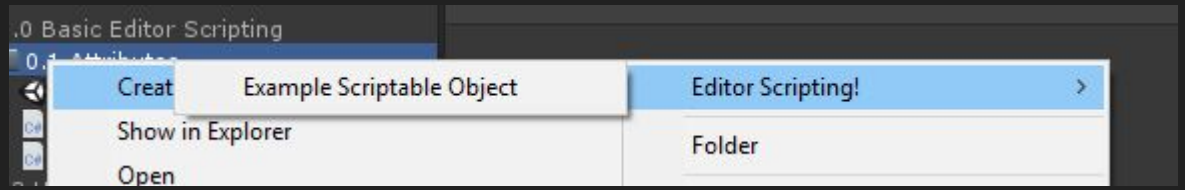
```
[CreateAssetMenu(menuName = "Editor Scripting!/Example Scriptable Object",
fileName = "New Scriptable Object")]
public class ScriptableObjectExample : ScriptableObject {
    [SerializeField]
    [Tooltip("First Value needs to be smaller than second value.")]
    private int _firstValue;

    [SerializeField]
    [Tooltip("Second Value needs to be larger or equal to first value.")]
    private int _secondValue;
    [...]
}
```

https://docs.unity3d.com/Manual/class-ScriptableObject.html

Basic Editor Scripting

Scriptable Objects



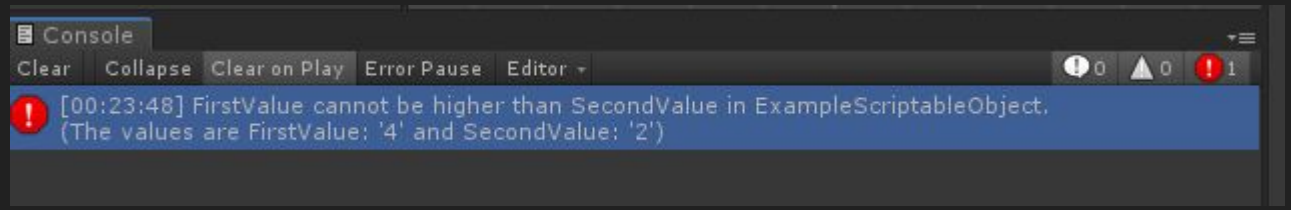


https://docs.unity3d.com/Manual/class-ScriptableObject.html

Basic Editor Scripting

OnValidate()

Allows to check custom conditions in our objects when they are modified. Works both on MonoBehaviours and ScriptableObjects (although the second one is undocumented and not detected in VS).



https://docs.unity3d.com/ScriptReference/MonoBehaviour.OnValidate.html
https://github.com/JetBrains/resharper-unity/issues/79

Basic Editor Scripting

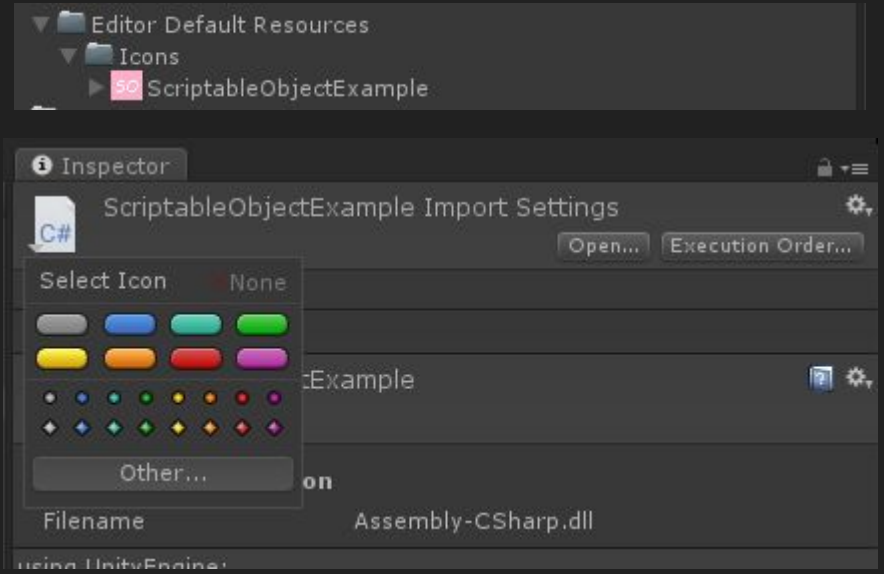
OnValidate()

```
public class ScriptableObjectExample : ScriptableObject
{
    [...]

    public void OnValidate()
    {
        if (_firstValue > _secondValue)
            Debug.LogErrorFormat(this, "FirstValue cannot be higher than SecondValue in {0}." +
                "\n(The values are FirstValue: '{1}' and SecondValue: '{2}'",
                name, _firstValue, _secondValue);
    }
}
```

https://docs.unity3d.com/ScriptReference/MonoBehaviour.OnValidate.html
https://github.com/JetBrains/resharper-unity/issues/79

Basic Editor Scripting

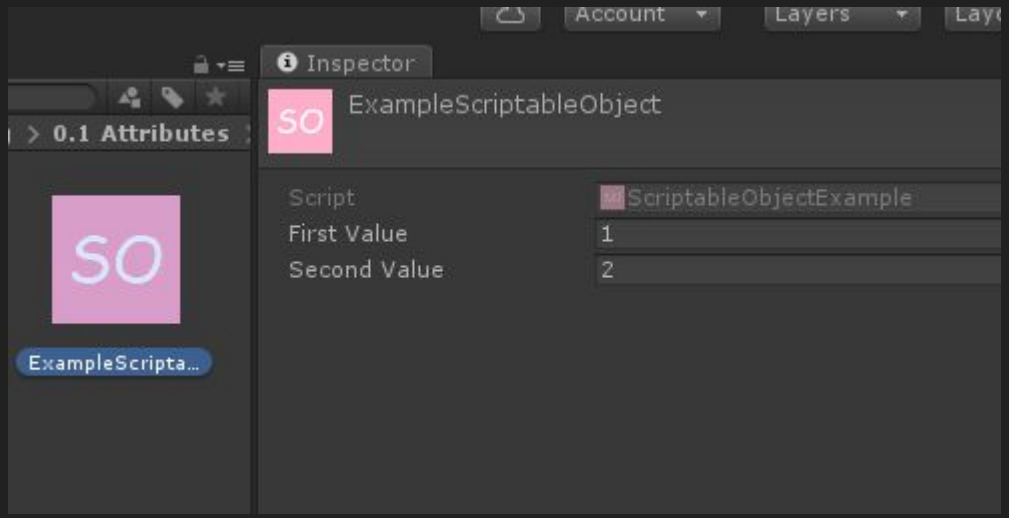
Custom Script Icons



https://docs.unity3d.com/Manual/AssigningIcons.html
https://docs.unity3d.com/Manual/SpecialFolders.html

Basic Editor Scripting

Custom Script Icons



https://docs.unity3d.com/Manual/AssigningIcons.html

Basic Editor Scripting Example

Present
IMGUI

Unity Serialization

Need to know.

- Read which types are serialized.
- Unity serializes data into YAML (YAML Ain't Markup Language) format.
- It's possible to edit YAML files (.prefab, .scene, .asset, etc.) directly in text editor of choice.
- **Unity serializes object references using AssetGUID and Local IDs (translated to Instance IDs for runtime usage).**

https://docs.unity3d.com/Manual/script-Serialization.html
https://unity3d.com/learn/tutorials/topics/best-practices/assets-objects-and-serialization
https://docs.unity3d.com/Manual/TextSceneFormat.html

Unity Serialization

Need to know.

```yaml
%YAML 1.1
%TAG !u! tag:unity3d.com,2011:
--- !u!114 &11400000
MonoBehaviour:
  m_ObjectHideFlags: 0
  m_CorrespondingSourceObject: {fileID: 0}
  m_PrefabInstance: {fileID: 0}
  m_PrefabAsset: {fileID: 0}
  m_GameObject: {fileID: 0}
  m_Enabled: 1
  m_EditorHideFlags: 0
  m_Script: {fileID: 11500000, guid: f9b88640025e50d439a59d741829eef0, type: 3}
  m_Name: ExampleScriptableObject
  m_EditorClassIdentifier:
  _firstValue: 1
  _secondValue: 2
```

https://docs.unity3d.com/Manual/script-Serialization.html
https://unity3d.com/learn/tutorials/topics/best-practices/assets-objects-and-serialization
https://docs.unity3d.com/Manual/TextSceneFormat.html

What is IMGUI?

IMGUI stands for “Immediate Mode” GUI. It’s a legacy GUI system that was used as a runtime GUI prior to 4.6 version of Unity Editor. It’s still used as a base for almost all editor code in Unity. (Although it’s being slowly replaced by UI Elements)

> “IMGUI is a code-driven GUI system, and is mainly intended as a tool for programmers. It is driven by calls to the **OnGUI** function on any script which implements it.”

---

https://docs.unity3d.com/Manual/GUIScriptingGuide.html

What is IMGUI?

IMGUI Example


```
using UnityEngine;

public class IMGUIExample : MonoBehaviour
{
    private bool _wasPressed = false;

    private void OnGUI()
    {
        GUI.Box(new Rect(10, 10, 200, 100), "IMGUI Example");

        if(GUI.Button(new Rect(20, 40, 180, 20), "Press me!"))
            _wasPressed = true;

        GUI.Label(new Rect(20, 70, 180, 40), "Was the button pressed?\n" +
            (_wasPressed ? "yes" : "no"));
    }
}
```

https://docs.unity3d.com/Manual/GUIScriptingGuide.html

What is IMGUI?

IMGUI Classes

```
//Creating the same label using four available GUI classes.
GUI.Label(new Rect(10, 10, 100, 20), "Test Label");
GUILayout.Label("Test Label");
UnityEditor.EditorGUI.LabelField(new Rect(10, 10, 100, 20), "Test Label");
UnityEditor.EditorGUILayout.LabelField("Test Label");

//Helpful classes
UnityEditor.EditorGUIUtility
UnityEditor.EditorApplication
UnityEditor.EditorUtility
UnityEditor.EditorStyles
```

https://docs.unity3d.com/ScriptReference/GUI.html
https://docs.unity3d.com/ScriptReference/EditorGUI.html

Advanced Editor Scripting

Custom Editor (Direct Method)

```
using UnityEditor;

[CustomEditor(typeof(ScriptableObjectExample))]
public class CustomEditorExample : Editor
{
    private ScriptableObjectExample _scriptableObjectExample;

    private void OnEnable()
    {
        _scriptableObjectExample = target as ScriptableObjectExample;
    }

    [...]
}
```

https://docs.unity3d.com/Manual/editor-CustomEditors.html

Advanced Editor Scripting

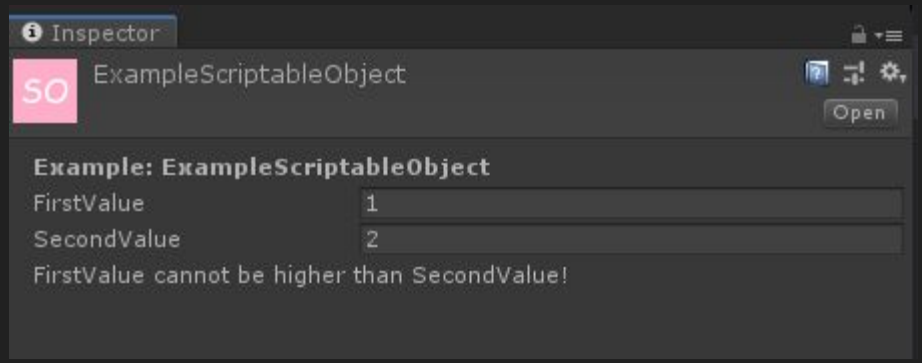
Custom Editor (Direct Method)

```
public override void OnInspectorGUI()
{
    EditorGUILayout.LabelField("Example: " + _scriptableObjectExample.name, EditorStyles.boldLabel);

    EditorGUI.BeginChangeCheck();

    _scriptableObjectExample.FirstValue = EditorGUILayout.IntField("FirstValue",
        _scriptableObjectExample.FirstValue);
    _scriptableObjectExample.SecondValue = EditorGUILayout.IntField("SecondValue",
        _scriptableObjectExample.SecondValue);

    EditorGUILayout.LabelField("FirstValue cannot be higher than SecondValue!");

    if (EditorGUI.EndChangeCheck())
        EditorUtility.SetDirty(_scriptableObjectExample);
}
```

https://docs.unity3d.com/Manual/editor-CustomEditors.html

Advanced Editor Scripting

Custom Editor



https://docs.unity3d.com/Manual/editor-CustomEditors.html

Advanced Editor Scripting

Custom Editor (Property Method)

```
[CustomEditor(typeof(ScriptableObjectExample))]
public class CustomEditorExample : Editor
{
    private SerializedProperty _firstValueProperty;
    private SerializedProperty _secondValueProperty;

    private void OnEnable()
    {
        _firstValueProperty = serializedObject.FindProperty("_firstValue");
        _secondValueProperty = serializedObject.FindProperty("_secondValue");
    }

    [...]
}
```

https://docs.unity3d.com/Manual/editor-CustomEditors.html

Advanced Editor Scripting

Custom Editor (Property Method)

```
public override void OnInspectorGUI()
{
    serializedObject.Update();

    EditorGUILayout.LabelField("Example: " + target.name, EditorStyles.boldLabel);

    EditorGUILayout.PropertyField(_firstValueProperty);
    EditorGUILayout.PropertyField(_secondValueProperty);

    EditorGUILayout.LabelField("FirstValue cannot be higher than SecondValue!");

    serializedObject.ApplyModifiedProperties();
}
```

https://docs.unity3d.com/Manual/editor-CustomEditors.html

Advanced Editor Scripting

Custom Editor (Direct Method)

Pros

- It gives you more control over how your editor looks.
- You can implement more complex behaviours in your editors.

Cons

- It requires usually more work into your editors.
- It doesn’t support Undo out of the box.
- It doesn’t allow you to take usage of full Property Drawers and built-in attributes.

---

https://docs.unity3d.com/Manual/editor-CustomEditors.html

Advanced Editor Scripting

Custom Editor (Property Method)

Pros

- It allows you to create your editors more quickly.
- You don’t need to reinvent the wheel in most of the cases.
- It operates on serialized data.
- It supports Undo out of the box.

Cons

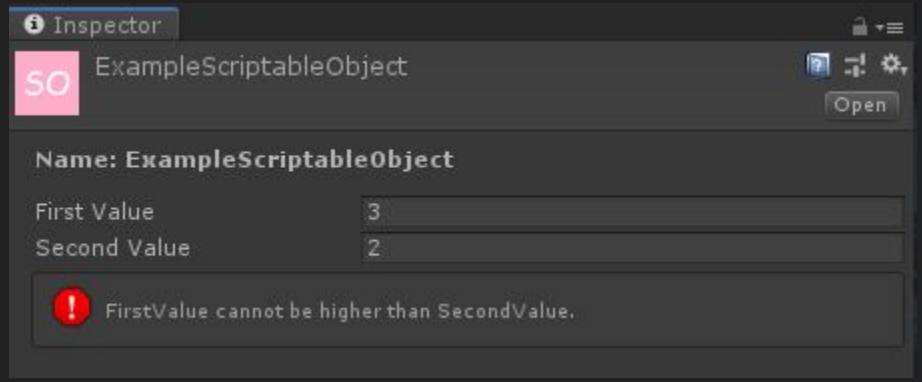
- You need to find references to serialized fields by strings.
- There is no other way to add custom behaviour to property field than creating custom property drawer.

---

https://docs.unity3d.com/Manual/editor-CustomEditors.html

Advanced Editor Scripting

Custom Editor (Pretty Version)



https://docs.unity3d.com/Manual/editor-CustomEditors.html

Advanced Editor Scripting

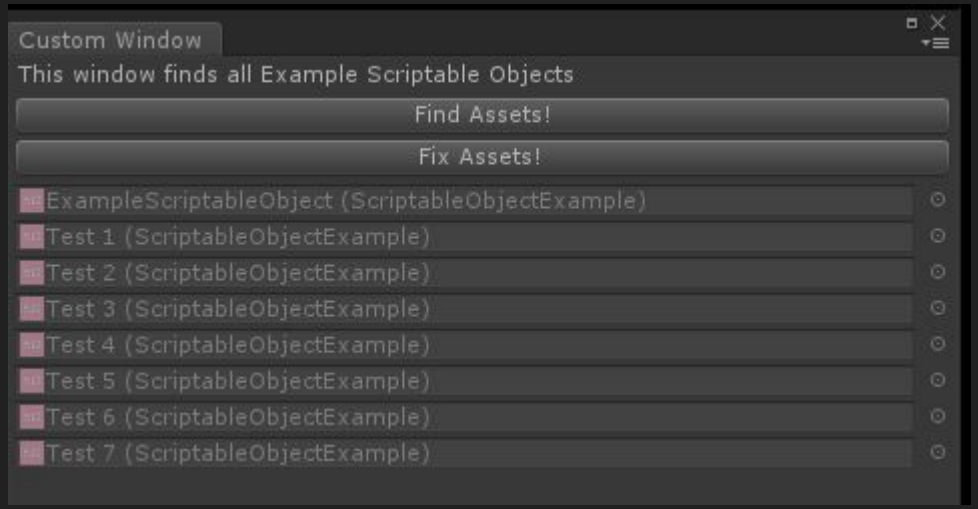
Custom Window

```
public class CustomWindowExample : EditorWindow
{
    [...]

    [MenuItem("Editor Scripting!/Custom Window Example")]
    public static void OpenWindow()
    {
        var window = GetWindow<CustomWindowExample>("Custom Window");
        window.position = new Rect(100, 100, 300, 600);
        window.Show();
    }

    private void OnGUI()
    {
        [...]
    }
}
```

https://docs.unity3d.com/Manual/editor-EditorWindows.html

Advanced Editor Scripting

Custom Window



https://docs.unity3d.com/Manual/editor-EditorWindows.html

Advanced Editor Scripting

Custom Window

```
//Shows a window with dropdown behaviour and styling.
window.ShowAsDropDown();
//Show the editor window in the auxiliary window.
window.ShowAuxWindow();
//Show a notification message.
window.ShowNotification();
//Shows an Editor window using popup-style framing.
window.ShowPopup();
//Show the EditorWindow as a floating utility window.
window.ShowUtility();
```

https://docs.unity3d.com/Manual/editor-EditorWindows.html

Advanced Editor Scripting

Custom Property Drawer

```
[Serializable]
public class CustomProperty
{
    public string ID;
    public ScriptableObjectExample ExampleObject;
}

public class CustomPropertyExample : MonoBehaviour
{
    public CustomProperty FirstProperty;
    public CustomProperty SecondProperty;
}
```

https://docs.unity3d.com/ScriptReference/PropertyDrawer.html

Advanced Editor Scripting

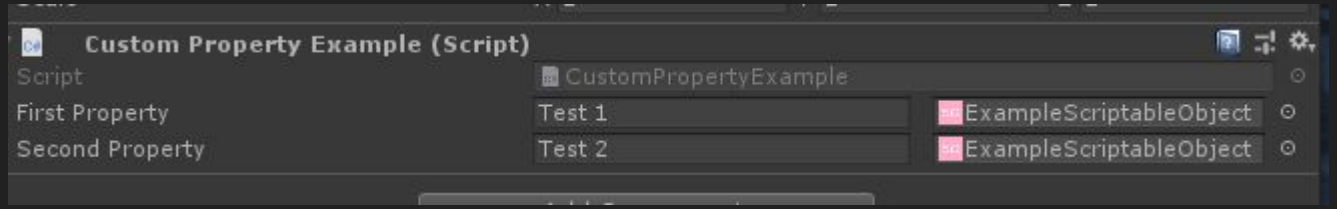
Custom Property Drawer

```
using UnityEngine;
using UnityEditor;

[CustomPropertyDrawer(typeof(CustomProperty))]
public class CustomPropertyDrawerExample : PropertyDrawer
{
    public override void OnGUI(Rect position, SerializedProperty property, GUIContent label)
    {
        var actualRect = EditorGUI.PrefixLabel(position, label);

        EditorGUI.PropertyField(
            new Rect(actualRect.x, actualRect.y, actualRect.width / 2 - 5, actualRect.height),
            property.FindPropertyRelative("ID"), GUIContent.none);

        EditorGUI.PropertyField(new Rect(actualRect.x + actualRect.width / 2 + 5,
            actualRect.y, actualRect.width / 2 - 10, actualRect.height),
            property.FindPropertyRelative("ExampleObject"), GUIContent.none);
    }
}
```

https://docs.unity3d.com/ScriptReference/PropertyDrawer.html

Advanced Editor Scripting

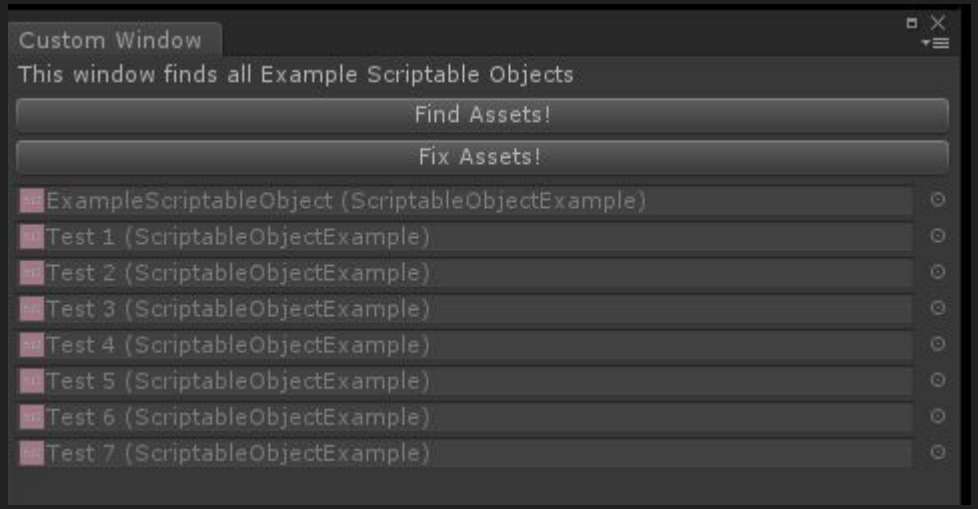
Custom Property Drawer



https://docs.unity3d.com/ScriptReference/PropertyDrawer.html

Advanced Editor Scripting

Editing Assets



https://docs.unity3d.com/ScriptReference/AssetDatabase.html

Advanced Editor Scripting

Editing Assets

```
private void FindAssets()
{
    if (_foundAssets == null)
        _foundAssets = new List<ScriptableObjectExample>();
    else
        _foundAssets.Clear();

    var assetGuids = AssetDatabase.FindAssets("t:ScriptableObjectExample");

    if (assetGuids == null || assetGuids.Length == 0)
        return;

    [...]
}
```

https://docs.unity3d.com/ScriptReference/AssetDatabase.html

Advanced Editor Scripting

Editing Assets

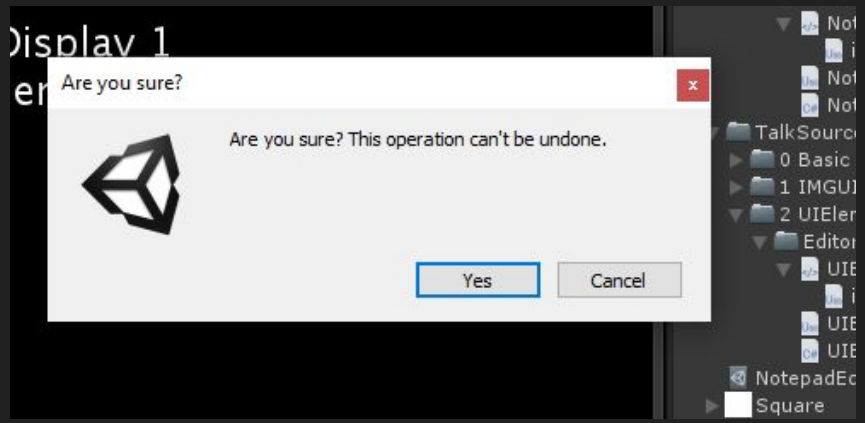
```
private void FindAssets()
{
    [...]

    foreach(var assetGuid in assetGuids)
    {
        var assetPath = AssetDatabase.GUIDToAssetPath(assetGuid);

        if (string.IsNullOrEmpty(assetPath))
            continue;

        var asset = AssetDatabase.LoadAssetAtPath<ScriptableObjectExample>(assetPath);

        if (asset != null)
            _foundAssets.Add(asset);
    }
}
```

https://docs.unity3d.com/ScriptReference/AssetDatabase.html

Advanced Editor Scripting

Editing Assets

```
private void FixAssets()
{
    if (!EditorUtility.DisplayDialog("Are you sure?", "Are you sure? This operation can't be
undone.", "Yes", "Cancel"))
        return;

    FindAssets();

    [...]
}
```



https://docs.unity3d.com/ScriptReference/AssetDatabase.html

Advanced Editor Scripting

Editing Assets

```
foreach(var asset in _foundAssets)
{
    var serializedObject = new SerializedObject(asset);

    serializedObject.Update();

    var firstValue = serializedObject.FindProperty("_firstValue");
    var secondValue = serializedObject.FindProperty("_secondValue");

    if (firstValue.intValue > secondValue.intValue)
    {
        firstValue.intValue = secondValue.intValue;
        Debug.LogFormat(asset, "Fixed object: {0}.", asset.name);
    }

    serializedObject.ApplyModifiedPropertiesWithoutUndo();
}
```

https://docs.unity3d.com/ScriptReference/AssetDatabase.html

IMGUI
Example

Future UIElements

What are UIElements?

UIElements is an experimental feature that is going to replace current UI systems (both Legacy and the 4.6 version) as an editor and runtime UI framework. Currently some of the editor features in 2018.3 and 2019.1 are already ported with UIElements

	Runtime dev UI	Runtime game UI	Editor
IMGUI	for debugging	not recommmended	✓
UGUI	✓	✓	not available
UIElements	2019.x	2020.x	2019.1

https://docs.unity3d.com/Manual/UIElements.html

What are UIElements?

UIElements are basically Unity implementation of old school web design principles. They are build using three main elements:

- UXML File - it is used to define the structure of our UI.
- USS File - basically a CSS file with smaller amount of features
- UQuery - queries that allow to search for elements in structure and assign new classes, etc (basically a jQuery)

---

https://docs.unity3d.com/Manual/UIElements.html

What are UIElements?

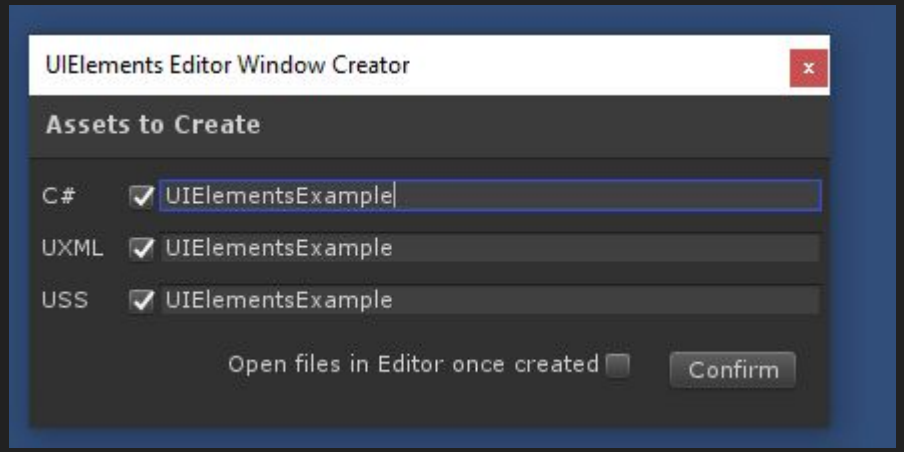
- They are built using open-source Flexbox integration called Yoga.
- Unity have also created in editor tools for debugging UIElements (that is basically the Developer modes from web browsers)
- It is a retained mode.
- You can use it in > 2019.1 versions of Unity. But be prepared for lack of materials about them.

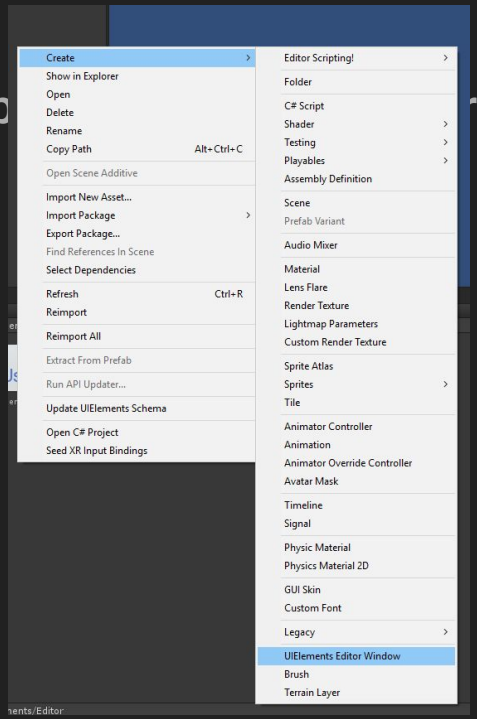
---

https://github.com/facebook/yoga

UIElements Example

Creating UI Elements can b rough Create Asset Menu





https://github.com/facebook/yoga

UIElements Example

Adding Clickable Button (UXML)

```
<?xml version="1.0" encoding="utf-8"?>
<engine:UXML
[...]
>
  <engine:Label text="Hello!" />
  <engine:Label text="I'm UIElements Example!" class="Test"/>
  <engine:Button text="Press me!" name="PressButton"/>
</engine:UXML>
```

https://docs.unity3d.com/Manual/UIElements.html

UIElements Example

Adding Clickable Button (USS)

```
Label {
    font-size: 25;
    font-weight: bold;
    text-color: rgb(68, 138, 255);
}

.Test {
    font-size: 17;
}

#PressButton {
    text-color: #ff0000;
    font-size: 15;
}
```

https://docs.unity3d.com/Manual/UIElements.html

UIElements Example

Adding Clickable Button (CS)

```
public void OnEnable()
{
    // Each editor window contains a root VisualElement object
    VisualElement root = rootVisualElement;

    // Import Files
    var visualTree =
AssetDatabase.LoadAssetAtPath<VisualTreeAsset>("../UIElementsExample.uxml");
    var styleSheet =
AssetDatabase.LoadAssetAtPath<StyleSheet>("..//UIElementsExample.uss");

    [...]
}
```

https://docs.unity3d.com/Manual/UIElements.html

UIElements Example

Adding Clickable Button (CS)

```
public void OnEnable()
{
    [...]

    //Create Visual Element Tree
    VisualElement uxmlLayout = visualTree.CloneTree();
    uxmlLayout.styleSheets.Add(styleSheet);
    root.Add(uxmlLayout);

    var pressButton = uxmlLayout.Q<Button>("PressButton");
    pressButton.clickable.clicked += OnPressButtonClicked;

}
```

https://docs.unity3d.com/Manual/UIElements.html

UIElements Example

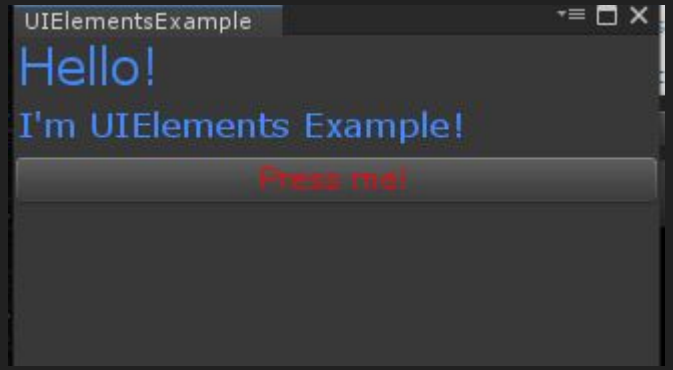
Adding Clickable Button (CS)

```
private void OnPressButtonClicked()
{
    Debug.Log("I was pressed!");
}
```

https://docs.unity3d.com/Manual/UIElements.html

UIElements Example

Adding Clickable Button (CS)



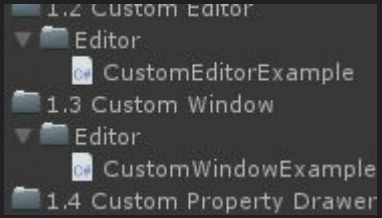
https://docs.unity3d.com/Manual/UIElements.html

UIElements
DEMO

Tips & Tricks

Tips & Tricks

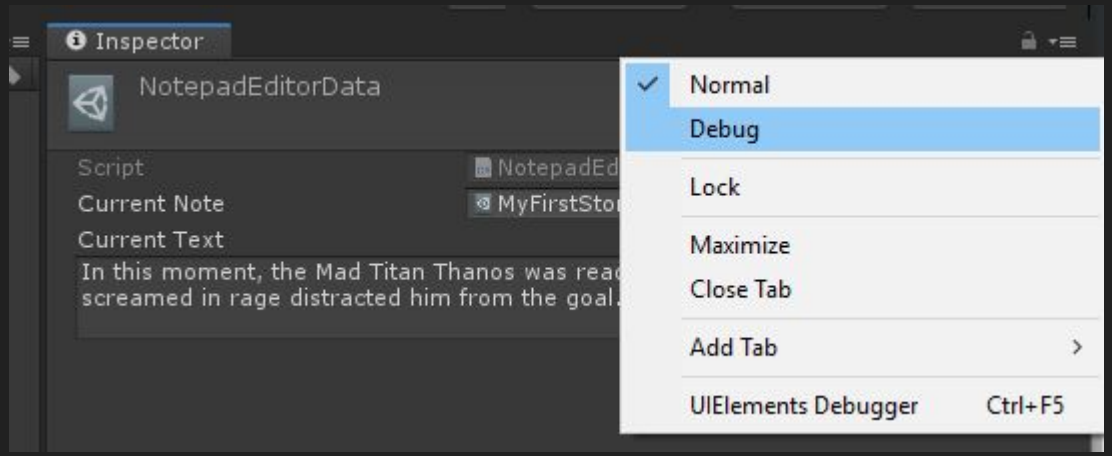
Separate your editor code from runtime



It is required for **Assembly Definitions!**

Tips & Tricks

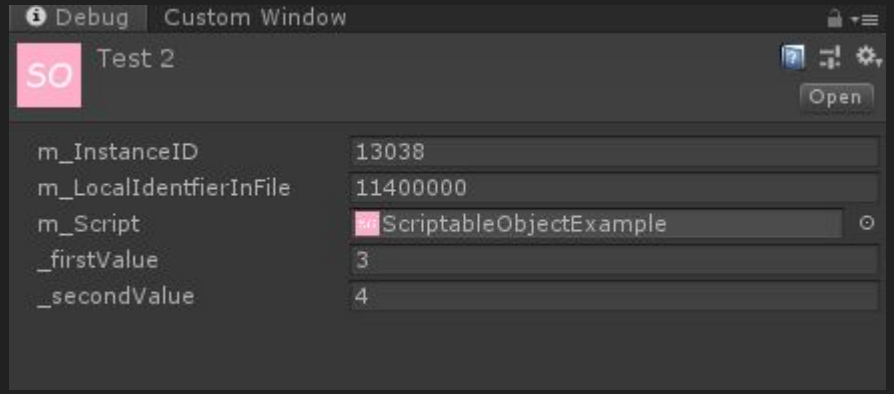
Use Debug View of Unity Editor



Tips & Tricks

Preview property names in Debug View.

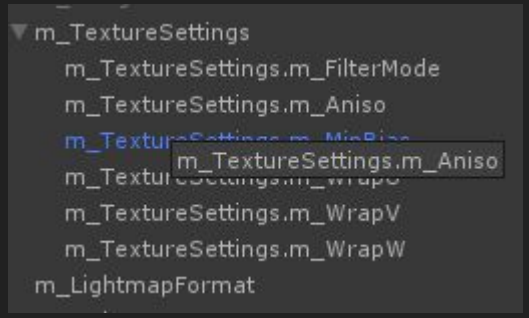
Press ALT + LMB on field in Inspector.



Tips & Tricks

Use full property path for complex structures.

```
serializedObject.FindProperty("m_TextureSettings.m_Aniso");
```



Tips & Tricks

Decompile Unity for examples.

https://github.com/Unity-Technologies/UnityCsReference

Tips & Tricks

Use nameof if you can.

```
serializedObject.FindProperty(nameof(_firstValue));
```

Tips & Tricks

Use Singleton SO for EditorWindow persistent data.

```
public static NotepadEditorData Instance {
    get {
        if (_instance == null) {
            var assets = AssetDatabase.FindAssets("t:NotepadEditorData");

            if (assets != null && assets.Length > 0) {
                var guid = assets[0];
                var path = AssetDatabase.GUIDToAssetPath(guid);

                _instance = AssetDatabase.LoadAssetAtPath<NotepadEditorData>(path);
            }
        }

        if (_instance == null) {
            _instance = CreateInstance<NotepadEditorData>();

            AssetDatabase.CreateAsset(_instance, "Assets/NotepadEditorData.asset");
            AssetDatabase.SaveAssets();
            AssetDatabase.Refresh();
        }

        return _instance;
    }
}
```

Noteworthy
Materials & Assets

Talks

Unite LA talk about UIElements

https://www.youtube.com/watch?v=MNNURw0LeoQ&t=838s

Unite talks Scriptable Objects

https://www.youtube.com/watch?v=6vmRwLYWNRo

https://www.youtube.com/watch?v=raQ3iHhE_Kk

Unite Berlin talk about Editor Scripts in Scene View

https://www.youtube.com/watch?v=Ah9CuzGa2vw

Odin

Inspector & Serializer

https://assetstore.unity.com/packages/tools/utilities/odin-inspector-and-serializer-89041

Serializer

https://github.com/TeamSirenix/odin-serializer

Summary

- You can use built-in attributes for some basic styling.
- Scriptable Objects are cool!
- Unity Serialization is based on YAML
- IMGUI can be used to create Custom Editors, Windows, Property Drawers, etc.
- UIElements are the new way that’s worth to dive into.

This talk and project

Q&A

We’re hiring!
Senior Unity Developer
jobs@robotgentleman.com

mateusz@robotgentleman.com

@mateuszpusty