CSC

Convertible Scene Creator

User Manual

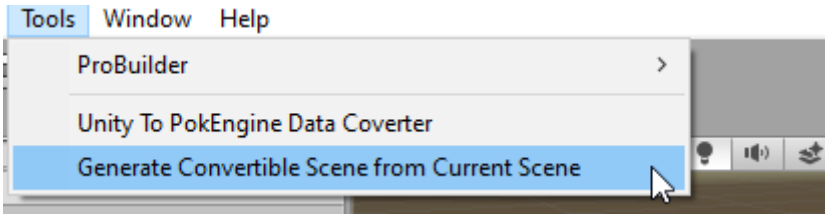
Oleg Loshkin

December 18, 2019

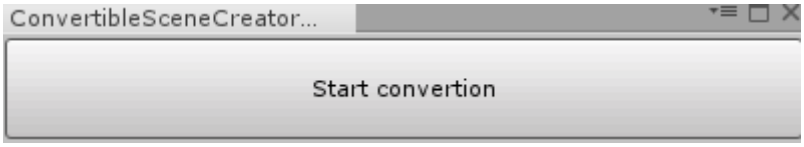
1 Introduction

The **Convertible Scene Creator, or CSC for short,** is a Unity tool used to **create .asset ScriptableObjects files from .unity Scenes** for conversion by UPDC. These .asset files can then be **used by the Unity to PokEngine Data Converter (UPDC)** to generate .pokconvertiblescene files readable by the PokEngine parser.

2 Accessing the Tool

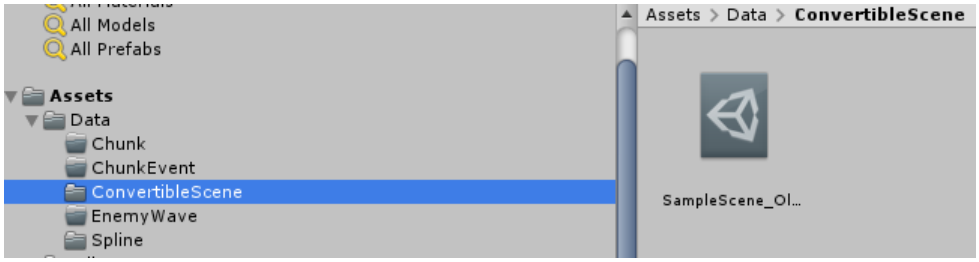


3 UI



The tool has only one button that when pressed will **launch the generation** of a .asset file **for the current scene**. **The generation may take some time**. Upon completion, the editor will close automatically.

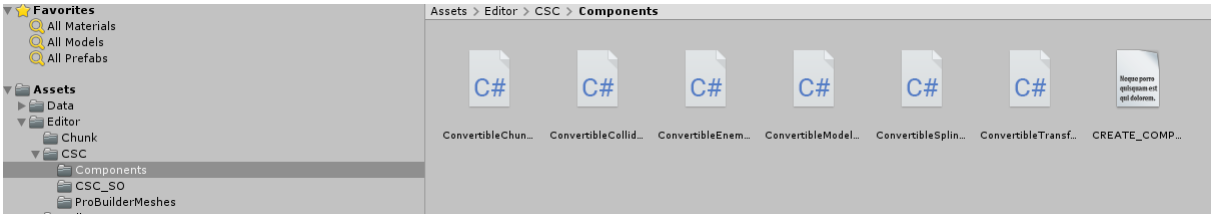
4 Output File Location



5 Adding a new Convertible Component

CSC allows you to implement the exportation of components it finds on the GameObjects in the Scene. **To implement** the exportation of a new component we will name "MyComponent" for example's sake **you must**:

- **Create a new .cs file under "Assets/Editor/CSC/Components"** that will define the convertible component:



- **Inside, you must define the struct that will be serialized. It must be marked as "[System.Serializable]" and must inherit from ConvertibleComponent.**
  Inside, you must also define a creator for your ConvertibleComponent. **It must implement a "CreateConvertibleComponents()" method**. See the image below.

```
3  using System;
4  using System.Collections.Generic;
5  using UnityEngine;
6  using UnityEngine.SceneManagement;
7
8  namespace ConvertibleSceneCreator
9  {
10     [Serializable]
11     public class MyComponent : ConvertibleComponent
12     {
13         [SerializeField] SomeData myData;
14
15         public SomeData MyData
16         {
17             get
18             {
19                 return myData;
20             }
21             set
22             {
23                 myData = value;
24             }
25         }
26     }
27
28     public class ConvertibleMyComponentCreator : ConvertibleComponentCreator<MyComponent>
29     {
30         public override InstanceIdToConvertibleComponent CreateConvertibleComponents(Scene scene)
31         {
32             var returnValue = new InstanceIdToConvertibleComponent();
33             var pairs = ConvertibleSceneCreatorUtility.GetAllIdComponentPairsOfType<MyComponent>(scene);
34             foreach (var pair in pairs)
35             {
36                 returnValue.instanceIds.Add(pair.Key);
37
38                 MyComponent component = new MyComponent();
39                 component.myData = pair.value.myData;
40                 returnValue.convertibleComponents.Add(component);
41             }
42             return returnValue;
43         }
44     }
45 }
```

See implementations of existing ConvertibleComponents for more examples of implementations.

- Next, **edit the "Components.cs" file** located under "Assets/Editor/CSC":

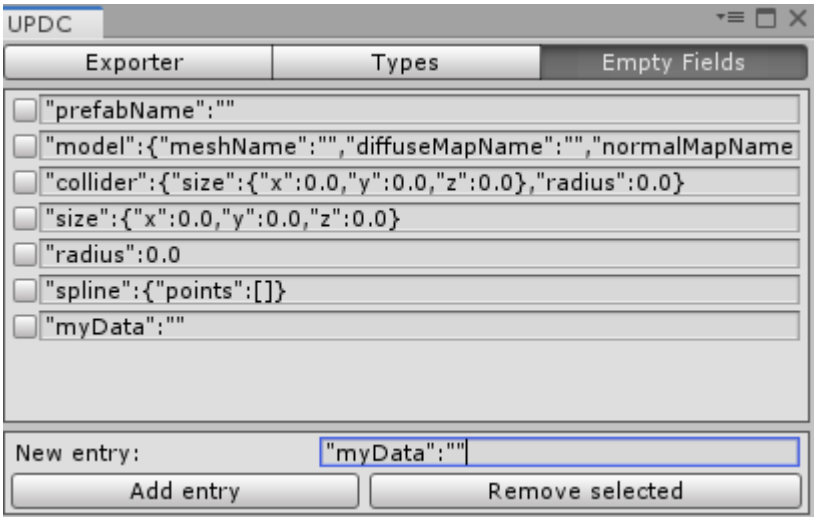
```
71      /// <summary>
72      /// Serializable object that is actually written to the .pokconvertiblescene .
73      /// </summary>
74      [Serializable]
75      public struct ConvertibleGameObject
76      {
77          // Add your component here.
78          public int instanceId; // Unity's GameObject's instanceID.
79          public bool isActive;
80          public string prefabName; // Name of the prefab the Unity's GameObject is linked to.
81          public ConvertibleTransform transform;
82          public ConvertibleModel model;
83          public ConvertibleCollider collider;
84          // public MyComponent component;
85      }
86
87      /// <summary>
88      /// Main class that actually does the converting.
89      /// </summary>
90      partial class ConvertibleSceneCreator : EditorWindow
91      {
92          void CreateConvertibleComponents(Scene scene)
93          {
94              convertibleComponents = new List<InstanceIdToConvertibleComponent>();
95
96              // Call your creator's CreateConvertibleComponents() implementation here and add the return value to the
97              convertibleComponents.Add(new ConvertibleTransformCreator().CreateConvertibleComponents(scene));
98              convertibleComponents.Add(new ConvertibleModelCreator().CreateConvertibleComponents(scene));
99              convertibleComponents.Add(new ConvertibleColliderCreator().CreateConvertibleComponents(scene));
100             // convertibleComponents.Add(new ConvertibleMyComponentCreator().CreateConvertibleComponents(scene));
101         }
```

Here, you will **add your new ConvertibleComponent to the declaration of a ConvertibleGameObject**. You will then **add a line to ConvertibleSceneCreator's CreateConvertibleComponents() method to add the components to ConvertibleSceneCreator's components list.**

Finally, **assign your MyComponent to the ConvertibleGameObject by adding a few lines to ConvertibleSceneCreator's GenerateConvertibleGameObject() method** like demonstrated below:

```
103     void GenerateConvertibleGameObject(out ConvertibleGameObject convertibleGameObject, int instanceId)
104     {
105         convertibleGameObject = new ConvertibleGameObject();
106         GameObject sceneGameObject = EditorUtility.InstanceIDToObject(instanceId) as GameObject;
107         convertibleGameObject.instanceId = instanceId;
108         convertibleGameObject.isActive = sceneGameObject.activeSelf;
109         convertibleGameObject.prefabName = PrefabUtility.IsAnyPrefabInstanceRoot(sceneGameObject) ?
110             new Path(PrefabUtility.GetPrefabAssetPathOfNearestInstanceRoot(sceneGameObject)).Get(PathOptions.NAME_ONLY, PathOptions.NO_EXTENSION) :
111             "";
112
113         // Set your convertible component's field in the convertibleGameObject being generated here.
114         convertibleGameObject.transform = convertibleComponents[0].GetByInstanceId(instanceId) as ConvertibleTransform;
115
116         ConvertibleModel model = convertibleComponents[1].GetByInstanceId(instanceId) as ConvertibleModel;
117         if (model != null)
118         {
119             convertibleGameObject.model = convertibleComponents[1].GetByInstanceId(instanceId) as ConvertibleModel;
120         }
121
122         ConvertibleCollider collider = convertibleComponents[2].GetByInstanceId(instanceId) as ConvertibleCollider;
123         if (collider != null)
124         {
125             convertibleGameObject.collider = convertibleComponents[2].GetByInstanceId(instanceId) as ConvertibleCollider;
126         }
127
128         /*
129         MyComponent component = convertibleComponents[x].GetByInstanceId(instanceId) as MyComponent;
130         if (component != null)
131         {
132             convertibleGameObject.component = convertibleComponents[x].GetByInstanceId(instanceId) as MyComponent;
133         }
134         */
135     }
```

- You will also **need to add a string to UPDC's "Empty Fields" tab that defines what an empty MyComponent looks like in JSON format**. This will make sure that the output file will not contain any empty instances of your new component.



You should now be able to see the values of your new component in any new .asset file's inspector and upon exporting it, the resulting JSON file should not have any uninitialized instances of your component.