CSC
Convertible Scene Creator
User Manual

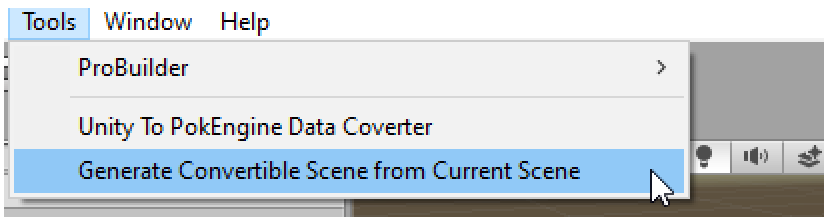
Oleg Loshkin

December 19, 2019

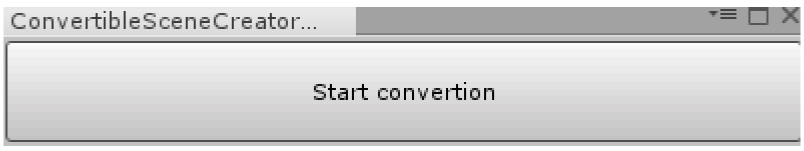
1 Introduction

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


Tools
Window
Help
ProBuilder
Unity To PokEngine Data Coverter
Generate Convertible Scene from Current Scene

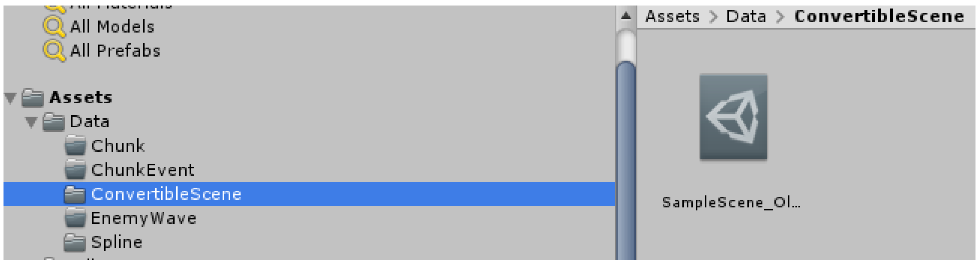
3 UI



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 and **.obj files for any meshes created using ProBuilder will be generated under** "`Assets/OBJ/CSC_Generated`".

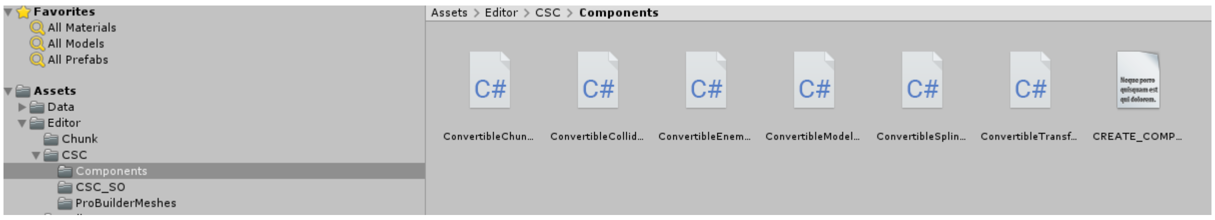
4 Output File Location



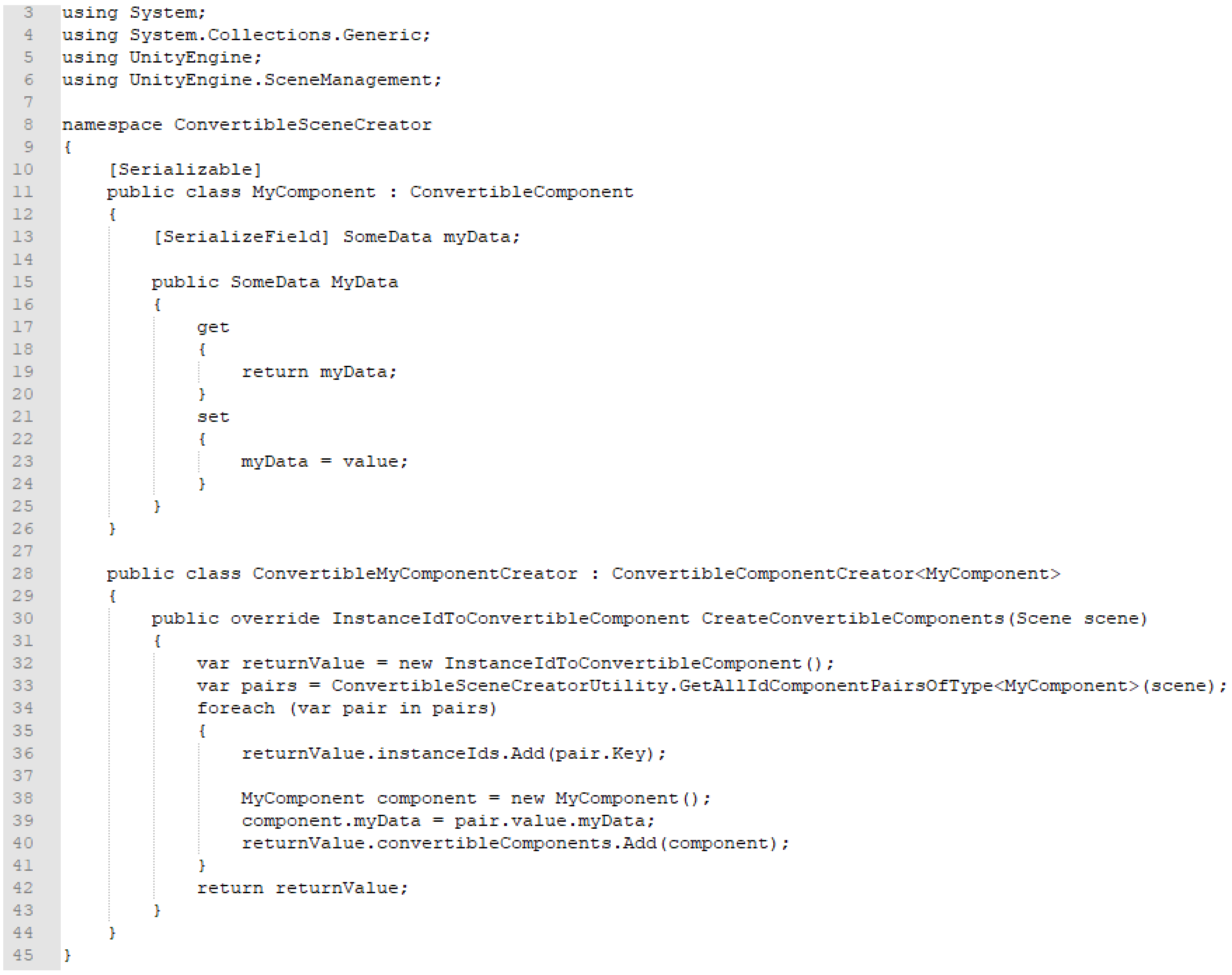
5 Adding a new Convertible Component

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:



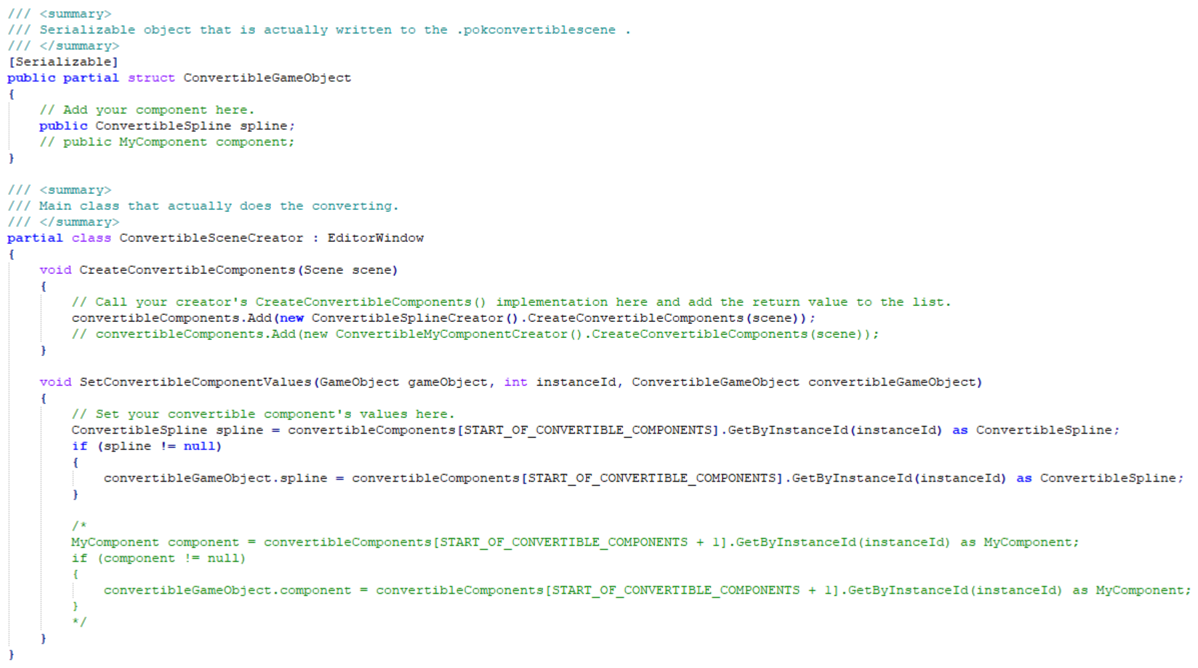
- **Inside, you must define the struct that will be serialized. It must be marked as "[System.Serializable]" and must inherit from ConvertibleComponent.**
  Inside, you must also define a factory for your ConvertibleComponent. **It must implement a "CreateConvertibleComponents()" method**. See the image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



```
using System;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

namespace ConvertibleSceneCreator
{
    [Serializable]
    public class MyComponent : ConvertibleComponent
    {
        [SerializeField] SomeData myData;

        public SomeData MyData
        {
            get
            {
                return myData;
            }
            set
            {
                myData = value;
            }
        }
    }

    public class ConvertibleMyComponentCreator : ConvertibleComponentCreator<MyComponent>
    {
        public override InstanceIdToConvertibleComponent CreateConvertibleComponents(Scene scene)
        {
            var returnValue = new InstanceIdToConvertibleComponent();
            var pairs = ConvertibleSceneCreatorUtility.GetAllIdComponentPairsOfType<MyComponent>(scene);
            foreach (var pair in pairs)
            {
                returnValue.instanceIds.Add(pair.Key);

                MyComponent component = new MyComponent();
                component.myData = pair.value.myData;
                returnValue.convertibleComponents.Add(component);
            }
            return returnValue;
        }
    }
}
```

See implementations of existing ConvertibleComponents for more examples of implementations.

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

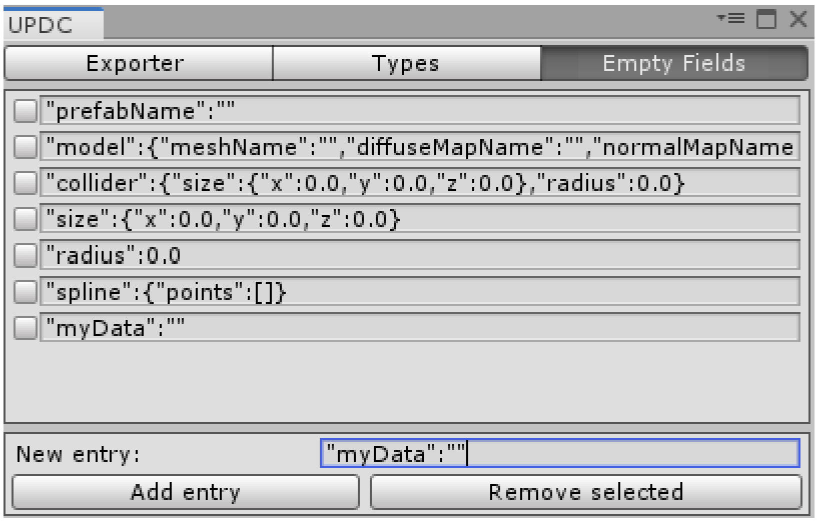


```
/// <summary>
/// Serializable object that is actually written to the .pokconvertiblescene .
/// </summary>
[Serializable]
public partial struct ConvertibleGameObject
{
    // Add your component here.
    public ConvertibleSpline spline;
    // public MyComponent component;
}

/// <summary>
/// Main class that actually does the converting.
/// </summary>
partial class ConvertibleSceneCreator : EditorWindow
{
    void CreateConvertibleComponents(Scene scene)
    {
        // Call your creator's CreateConvertibleComponents() implementation here and add the return value to the list.
        convertibleComponents.Add(new ConvertibleSplineCreator().CreateConvertibleComponents(scene));
        // convertibleComponents.Add(new ConvertibleMyComponentCreator().CreateConvertibleComponents(scene));
    }

    void SetConvertibleComponentValues(GameObject gameObject, int instanceId, ConvertibleGameObject convertibleGameObject)
    {
        // Set your convertible component's values here.
        ConvertibleSpline spline = convertibleComponents[START_OF_CONVERTIBLE_COMPONENTS].GetByInstanceId(instanceId) as ConvertibleSpline;
        if (spline != null)
        {
            convertibleGameObject.spline = convertibleComponents[START_OF_CONVERTIBLE_COMPONENTS].GetByInstanceId(instanceId) as ConvertibleSpline;
        }

        /*
        MyComponent component = convertibleComponents[START_OF_CONVERTIBLE_COMPONENTS + 1].GetByInstanceId(instanceId) as MyComponent;
        if (component != null)
        {
            convertibleGameObject.component = convertibleComponents[START_OF_CONVERTIBLE_COMPONENTS + 1].GetByInstanceId(instanceId) as MyComponent;
        }
        */
    }
}
```

  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**.

- 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.



"prefabName":""
"model":{"meshName":"","diffuseMapName":"","normalMapName
"collider":{"size":{"x":0.0,"y":0.0,"z":0.0},"radius":0.0}
"size":{"x":0.0,"y":0.0,"z":0.0}
"radius":0.0
"spline":{"points":[]}
"myData":""

New entry: "myData":""

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.