CSC
Convertible Scene Creator
User Manual

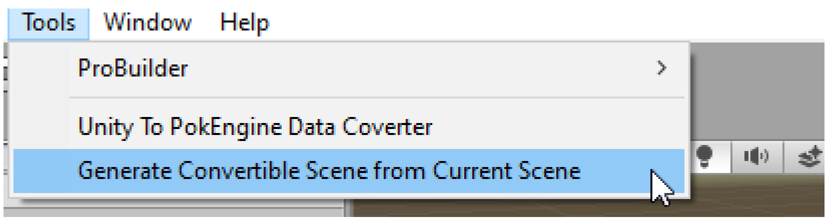
Oleg Loshkin

December 16, 2019

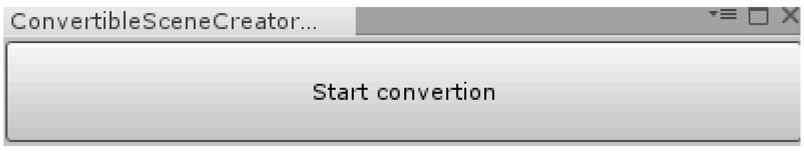
1 Introduction

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

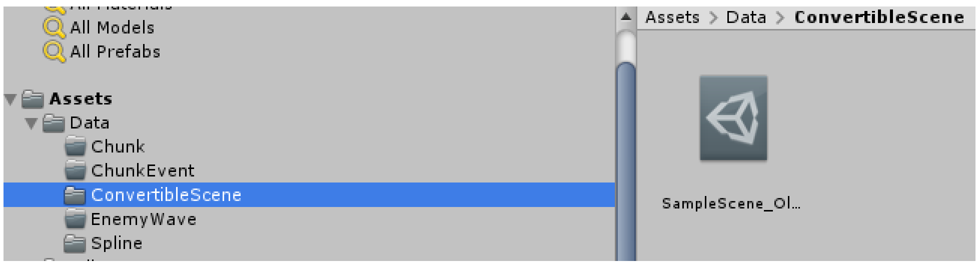

Tools
Window
Help
ProBuilder
Unity To PokEngine Data Coverter
Generate Convertible Scene from Current Scene

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.

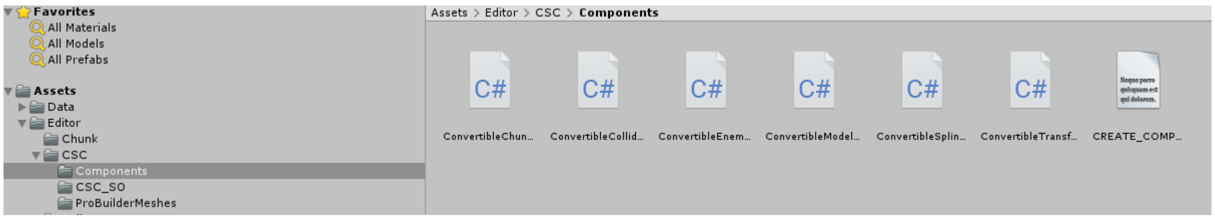
4 Output File Location



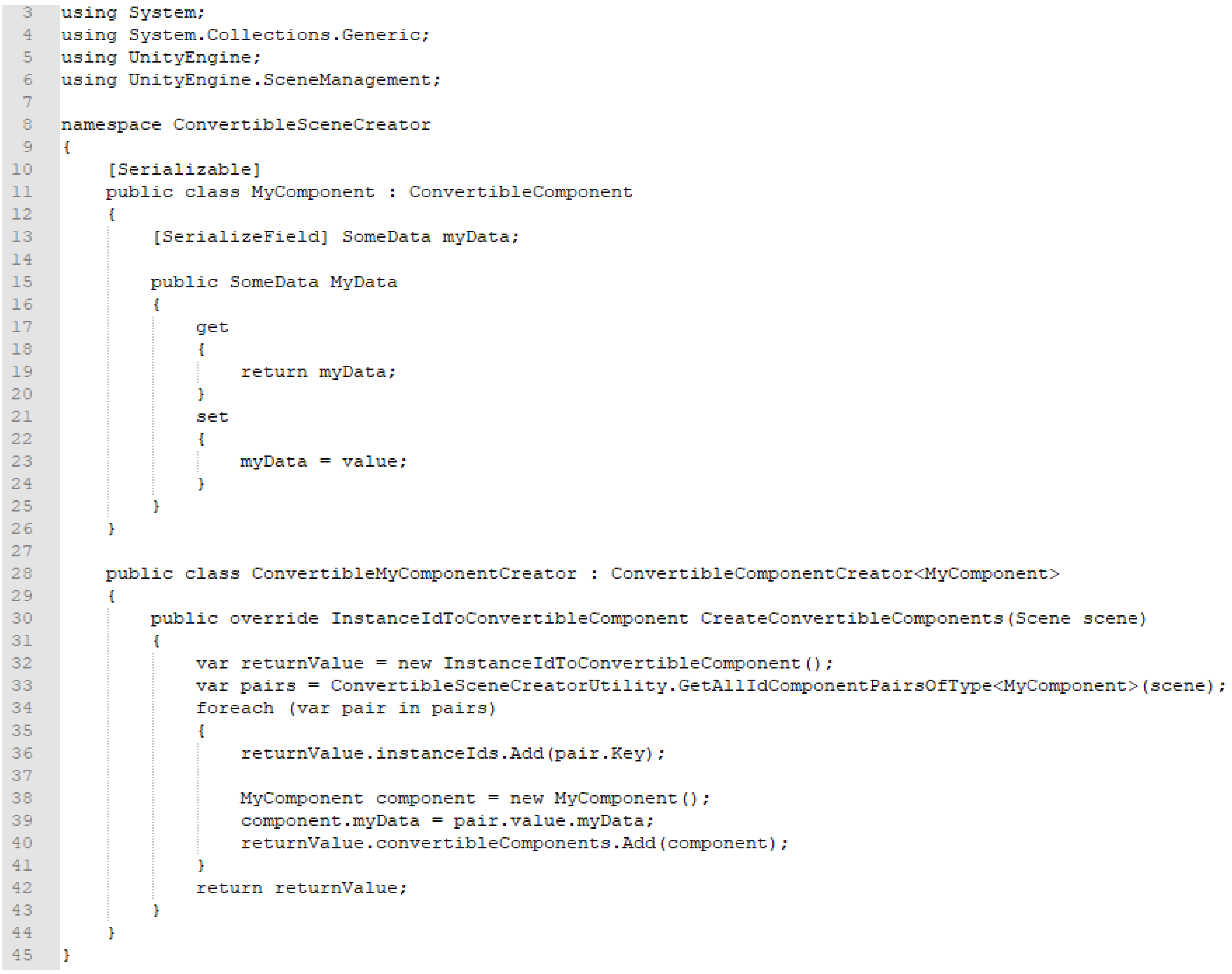
5 Adding a new Convertible Component

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:



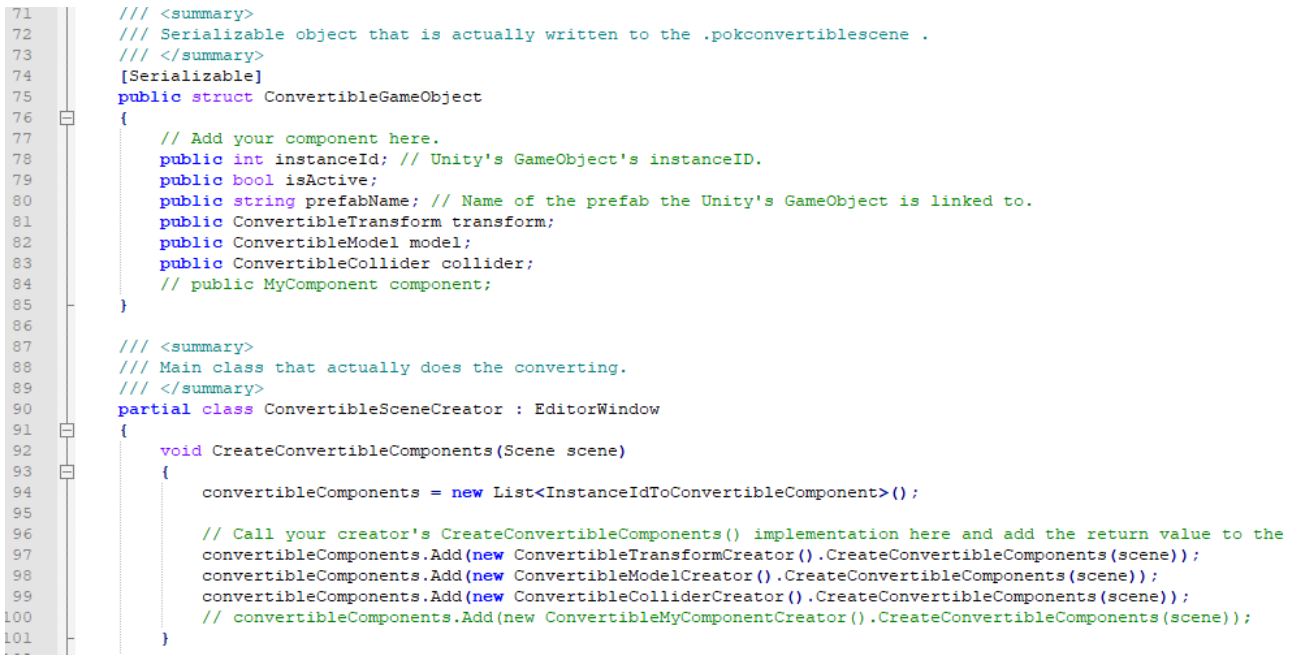
- **Inside, you must define the struct that will be serialized**. **It must be marked as "**[System.Serializable]**" and must inherit from ConvertibleComponent**.
  Inside, you must also define a creator for your ConvertibleComponent. **It must implement a "CreateConvertibleComponents()" method**. See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



```
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.

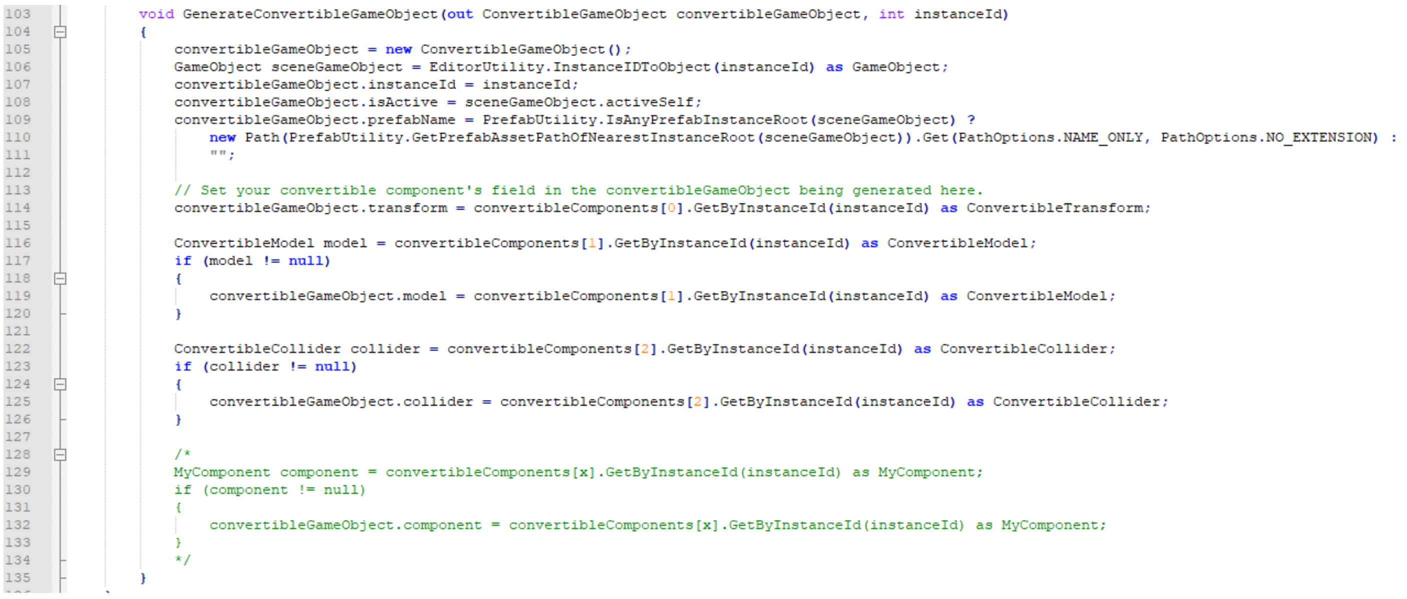
- Next, **edit the "Components.cs" file** located under "Assets/Editor/CSC":



```
    /// <summary>
    /// Serializable object that is actually written to the .pokconvertiblescene .
    /// </summary>
    [Serializable]
    public struct ConvertibleGameObject
    {
        // Add your component here.
        public int instanceId; // Unity's GameObject's instanceID.
        public bool isActive;
        public string prefabName; // Name of the prefab the Unity's GameObject is linked to.
        public ConvertibleTransform transform;
        public ConvertibleModel model;
        public ConvertibleCollider collider;
        // public MyComponent component;
    }

    /// <summary>
    /// Main class that actually does the converting.
    /// </summary>
    partial class ConvertibleSceneCreator : EditorWindow
    {
        void CreateConvertibleComponents(Scene scene)
        {
            convertibleComponents = new List<InstanceIdToConvertibleComponent>();

            // Call your creator's CreateConvertibleComponents() implementation here and add the return value to the
            convertibleComponents.Add(new ConvertibleTransformCreator().CreateConvertibleComponents(scene));
            convertibleComponents.Add(new ConvertibleModelCreator().CreateConvertibleComponents(scene));
            convertibleComponents.Add(new ConvertibleColliderCreator().CreateConvertibleComponents(scene));
            // convertibleComponents.Add(new ConvertibleMyComponentCreator().CreateConvertibleComponents(scene));
        }
```

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:



```
        void GenerateConvertibleGameObject(out ConvertibleGameObject convertibleGameObject, int instanceId)
        {
            convertibleGameObject = new ConvertibleGameObject();
            GameObject sceneGameObject = EditorUtility.InstanceIDToObject(instanceId) as GameObject;
            convertibleGameObject.instanceId = instanceId;
            convertibleGameObject.isActive = sceneGameObject.activeSelf;
            convertibleGameObject.prefabName = PrefabUtility.IsAnyPrefabInstanceRoot(sceneGameObject) ?
                new Path(PrefabUtility.GetPrefabAssetPathOfNearestInstanceRoot(sceneGameObject)).Get(PathOptions.NAME_ONLY, PathOptions.NO_EXTENSION) :
                "";

            // Set your convertible component's field in the convertibleGameObject being generated here.
            convertibleGameObject.transform = convertibleComponents[0].GetByInstanceId(instanceId) as ConvertibleTransform;

            ConvertibleModel model = convertibleComponents[1].GetByInstanceId(instanceId) as ConvertibleModel;
            if (model != null)
            {
                convertibleGameObject.model = convertibleComponents[1].GetByInstanceId(instanceId) as ConvertibleModel;
            }

            ConvertibleCollider collider = convertibleComponents[2].GetByInstanceId(instanceId) as ConvertibleCollider;
            if (collider != null)
            {
                convertibleGameObject.collider = convertibleComponents[2].GetByInstanceId(instanceId) as ConvertibleCollider;
            }

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

You should now be able to see the values of your new component in any new .asset file's inspector.