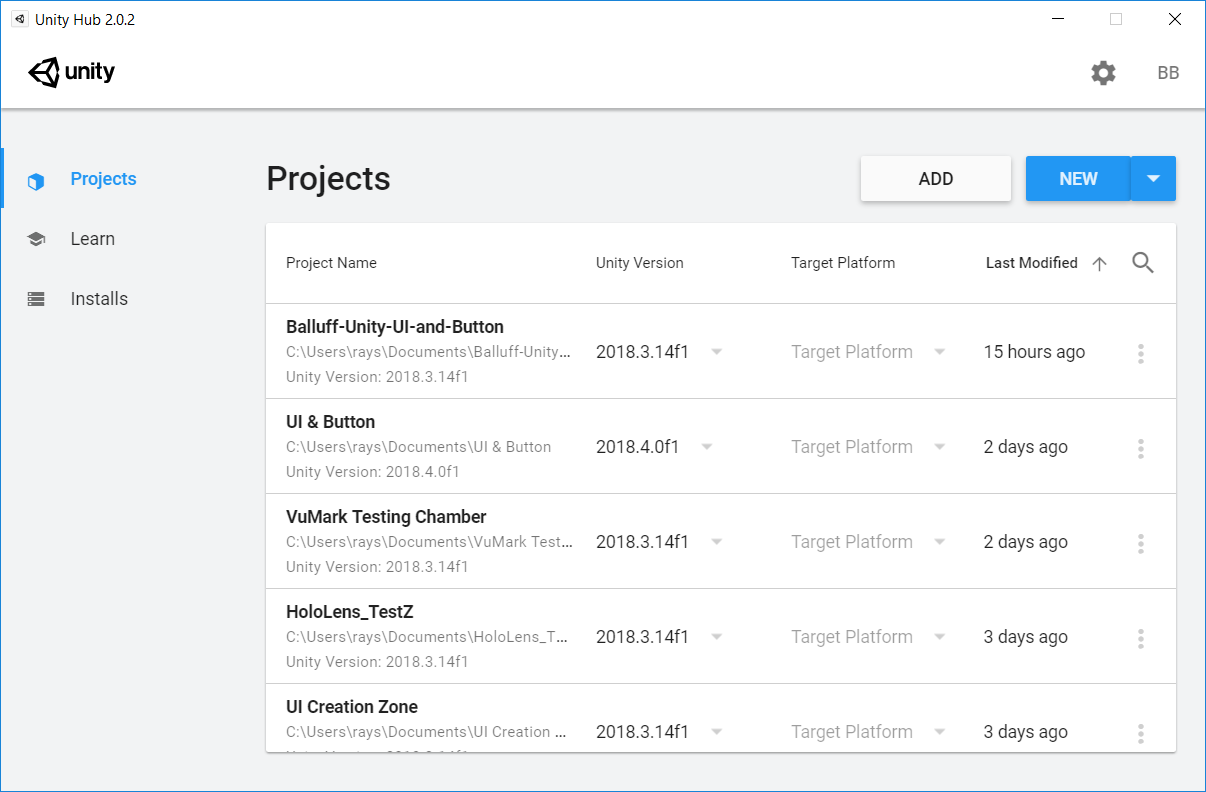
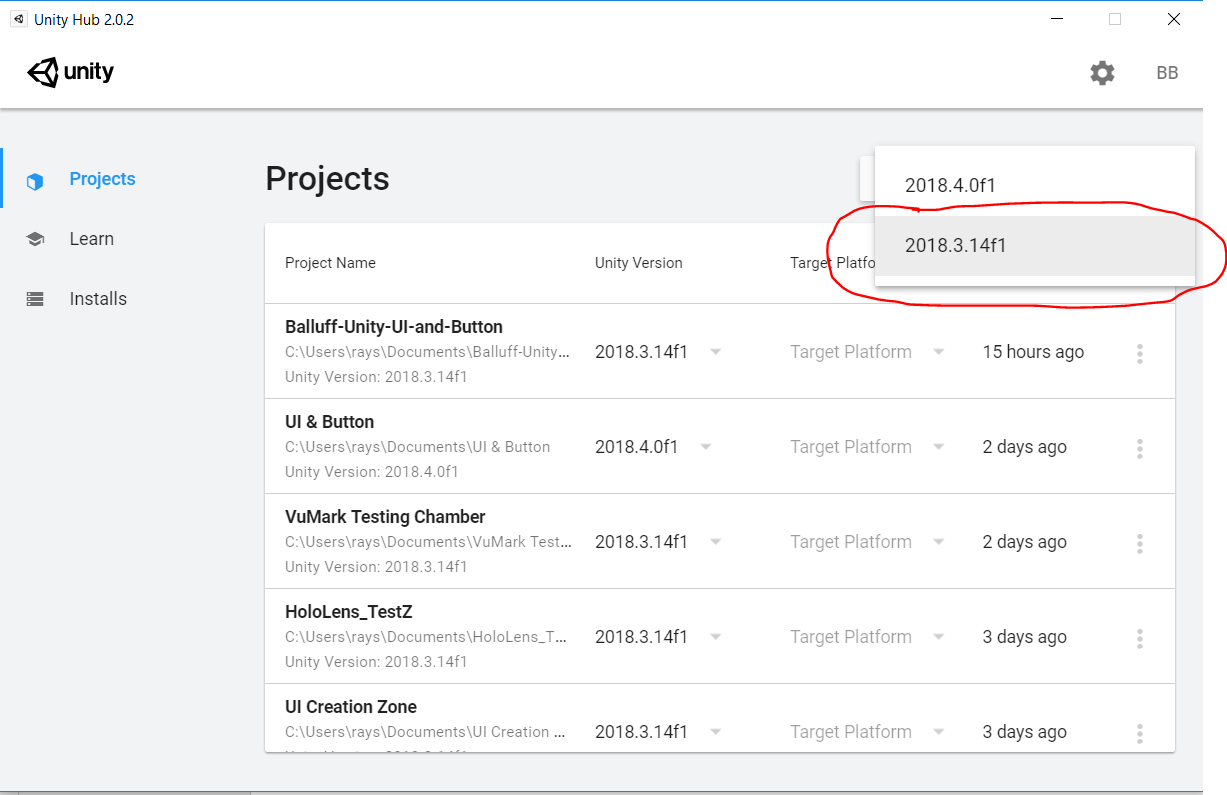
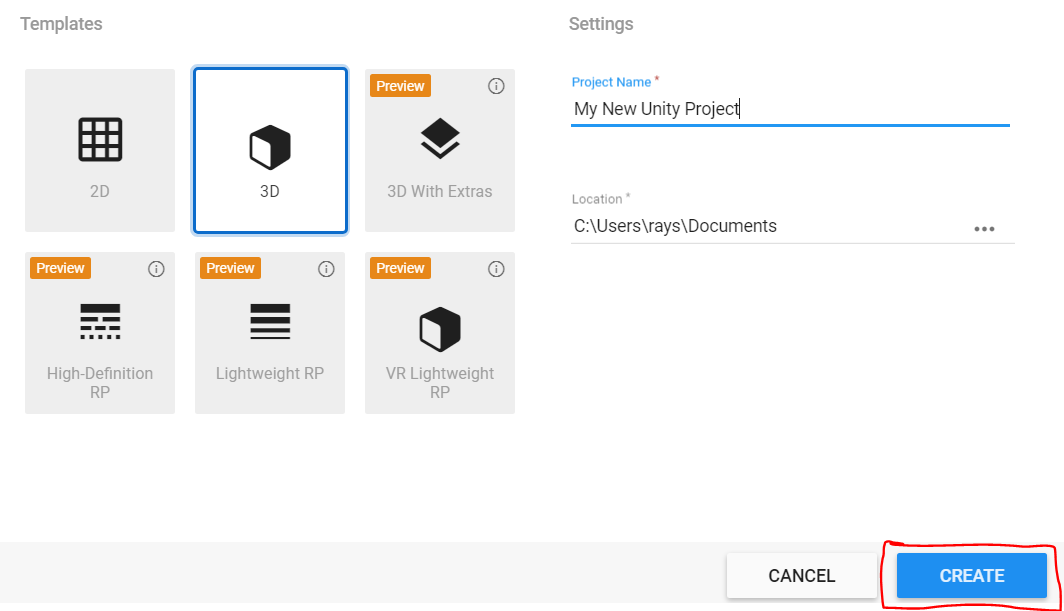
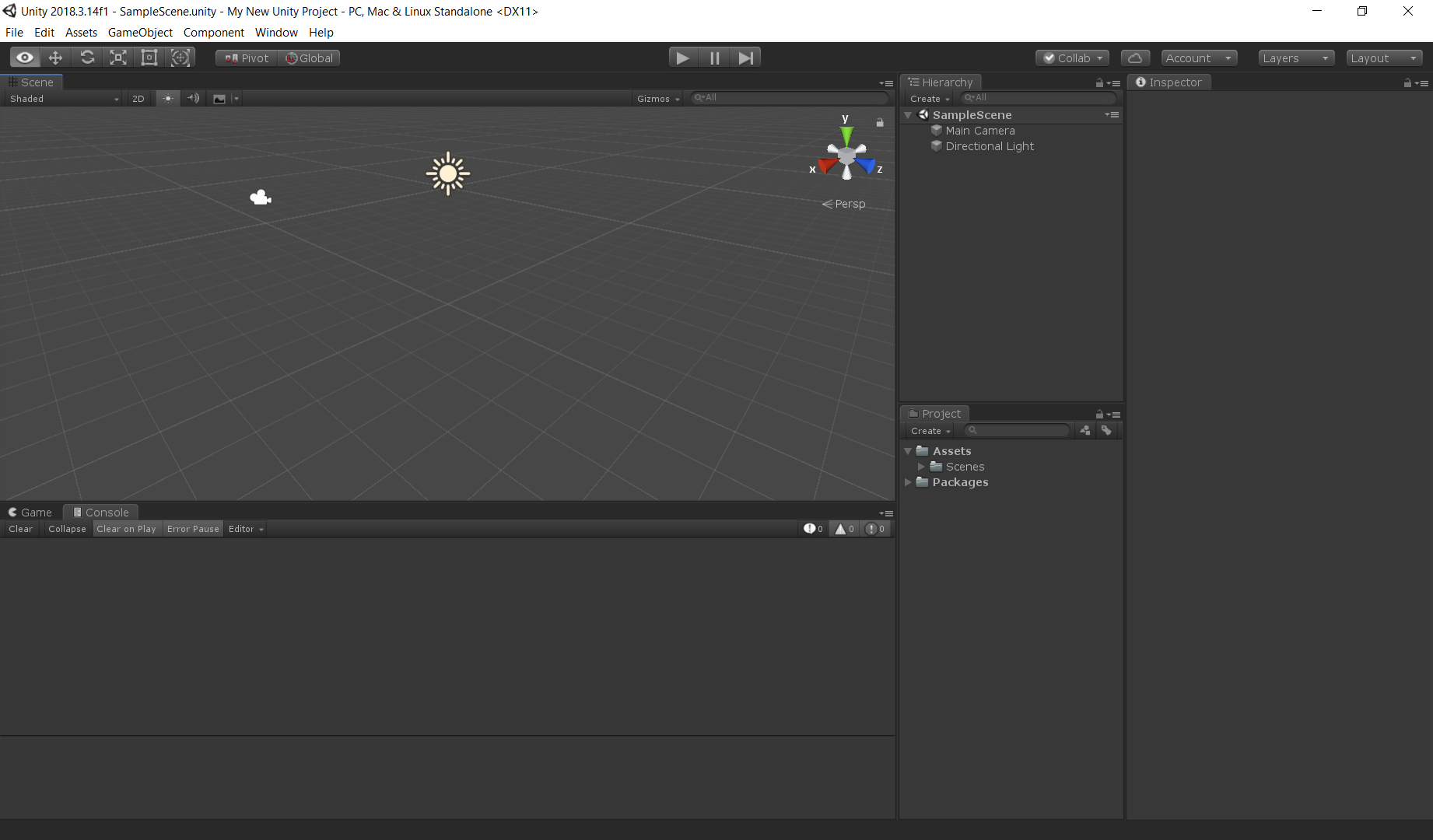
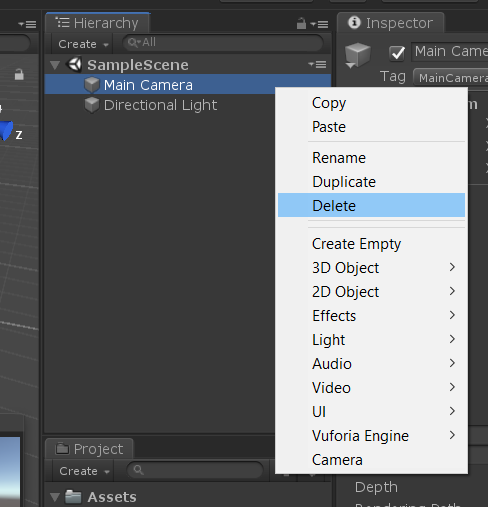
# Creating Unity Project with Newtonsoft.js

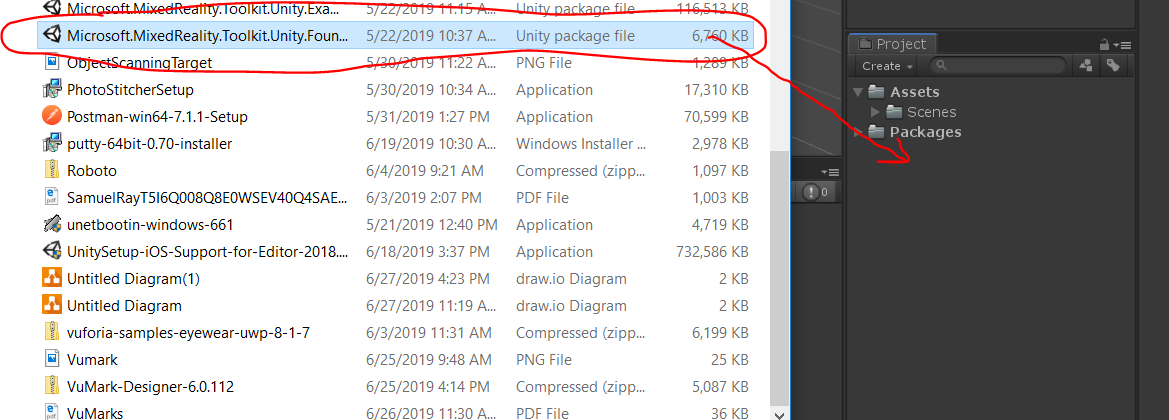
Make sure you have Unity 2018.3.14 (some of the later versions make new functions unusable)

Open a new Project with Unity Hub

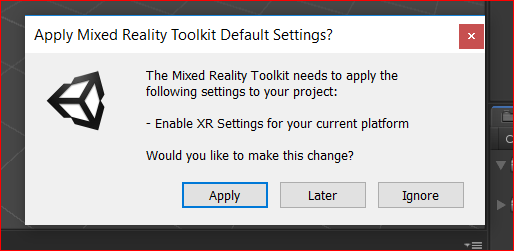
  

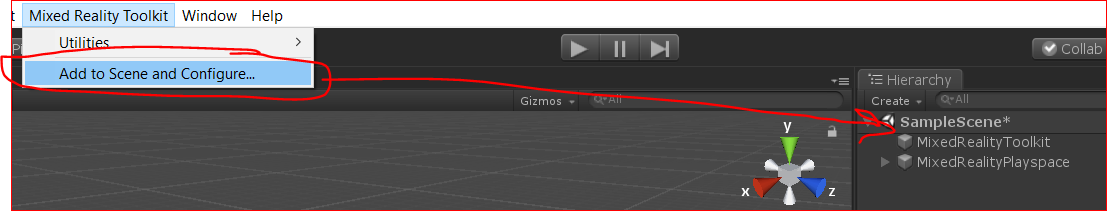
After Creating the new UnityProject, delete the Camera and Directional Light



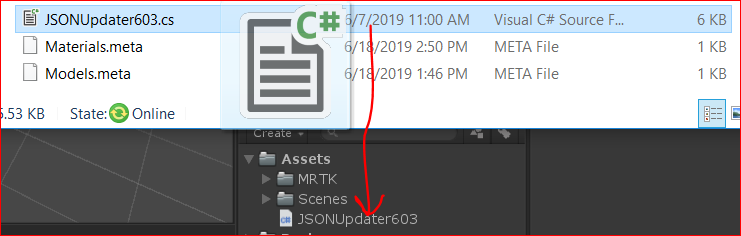
Drag your MRTK .unitypackage file into Unity

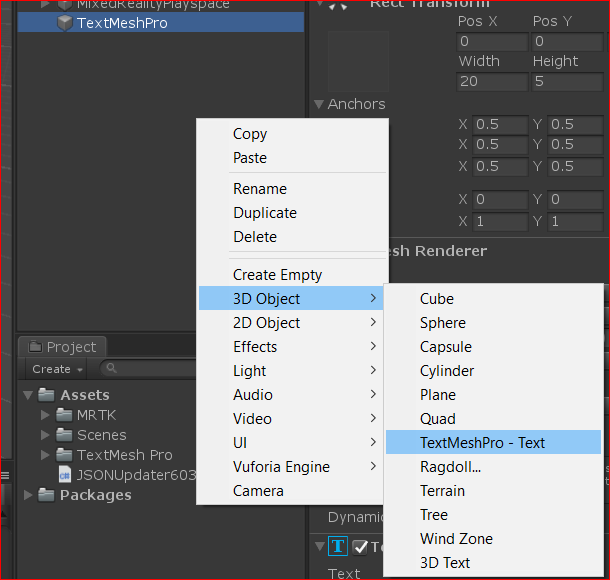
Apply XR Settings to Current Platform



Click your new Mixed Reality Toolkit Tab and select Add to Scene and Configure. It will bring up two new objects for you to play with.

Create a new TextMeshPro 3D Object and drag your JSONUpdater.cs Script into Unity

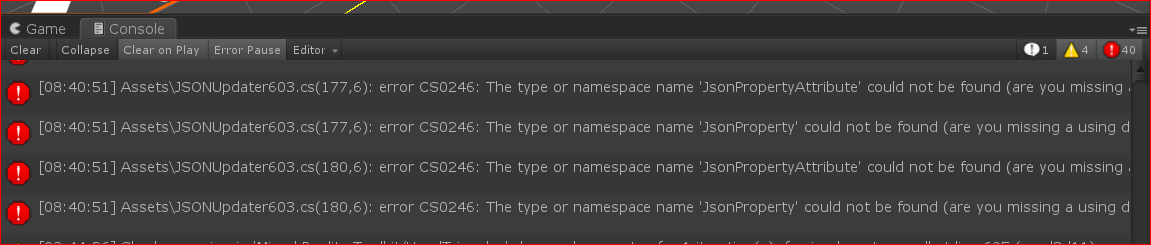




Import TMP Essentials

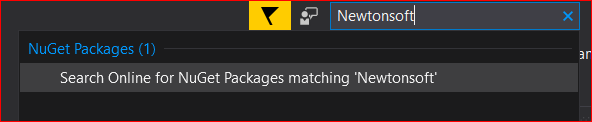


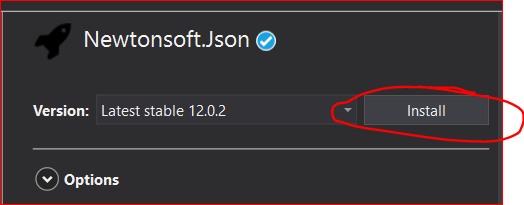
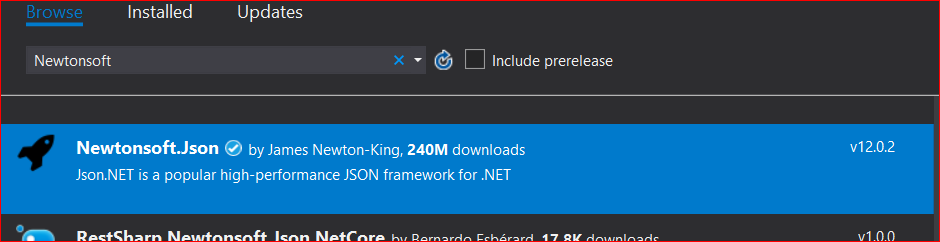
See these errors?:

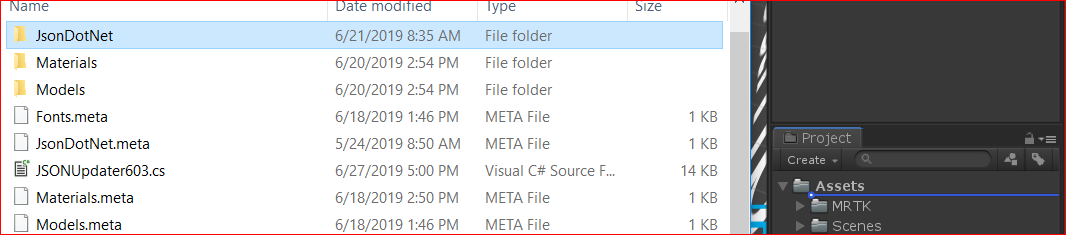


Let’s fix them. Click on the JSONUpdater Script VSCode 2017 should open up for you

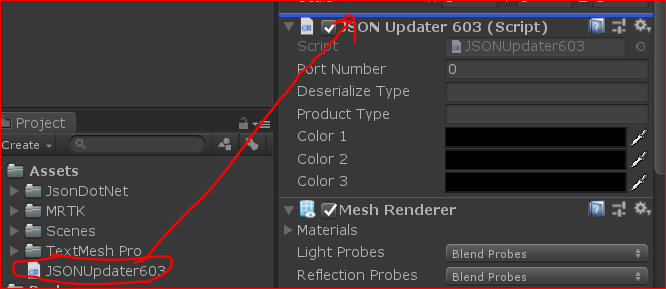
Search for Newtonsoft





Find the JsonDotNet import tool and add that to your Project Hierarchy. This should get rid of the errors 

Now click on your TextMeshPro object in Hierarchy and drag the JSON Updater into the Inspector



You’re done! Now you can use your script however you want with the Editor.

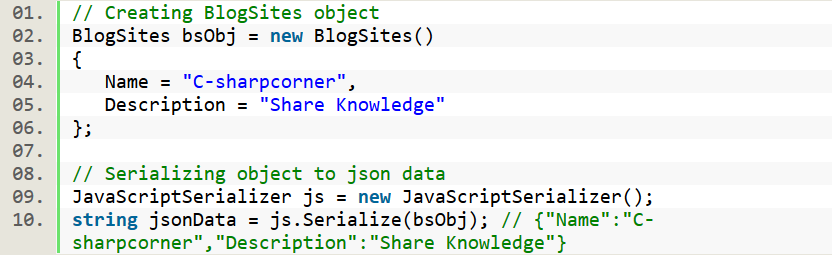
# De/Serialization JSON to C#

<https://www.c-sharpcorner.com/article/json-serialization-and-deserialization-in-c-sharp/>

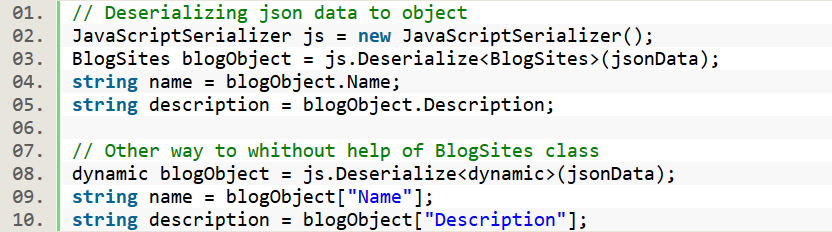
## What is this?

With JavaScriptSerializer:

In Serialization, it converts a custom .Net object to a JSON string. In the following code, it creates an instance of BlogSiteclass and assigns values to its properties. Then we create an instance of DataContractJsonSerializer class by passing the parameter BlogSite class and create an instance of MemoryStream class to write object(BlogSite). Lastly it creates an instance of StreamReader class to read JSON data from MemorySteam object.Why should we use this?



In Deserialization, it does the opposite of Serialization which means it converts JSON string to custom .Net object. In the following code, it creates JavaScriptSerializer instance and calls Deserialize() by passing JSON data. It returns custom object (BlogSites) from JSON data.



With Json.NET:

Json.NET is a third party library which helps conversion between JSON text and .NET object using the JsonSerializer. The JsonSerializer converts .NET objects into their JSON equivalent text and back again by mapping the .NET object property names to the JSON property names. It is open source software and free for commercial purposes.

The following are some awesome features,

* Flexible JSON serializer for converting between .NET objects and JSON.
* LINQ to JSON for manually reading and writing JSON.
* High performance, faster than .NET's built-in JSON serializers.
* Easy to read JSON.
* Convert JSON to and from XML.
* Supports .NET 2, .NET 3.5, .NET 4, Silverlight and Windows Phone.

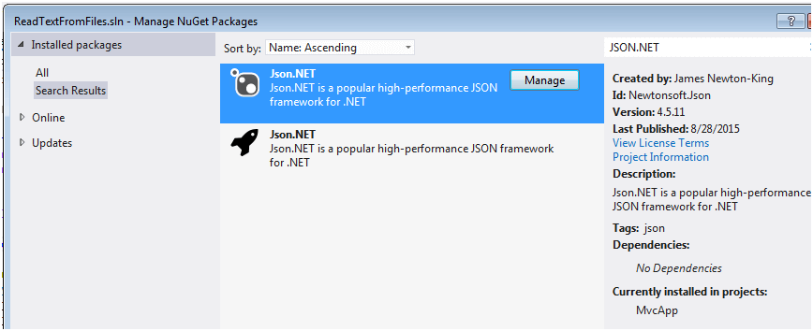
Let’s start learning how to install and implement:

In Visual Studio, go to Tools Menu -> Choose Library Package Manager -> Package Manager Console. It opens a command window where we need to put the following command to install Newtonsoft.Json.

Install-Package Newtonsoft.Json

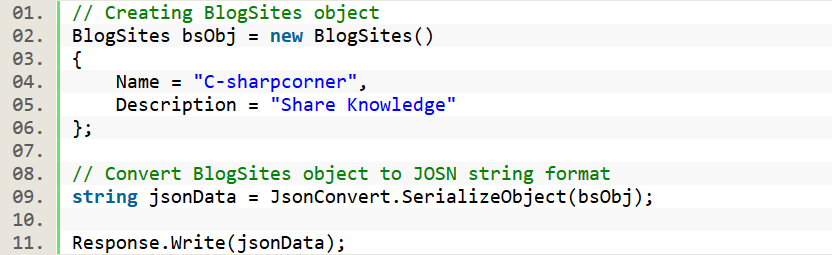
OR

In Visual Studio, Tools menu -> Manage Nuget Package Manager Solution and type “JSON.NET” to search it online. Here's the figure,



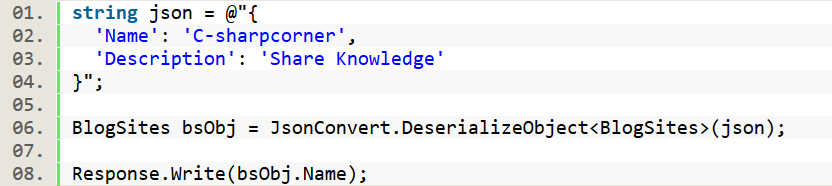
**Serialization**

In Serialization, it converts a custom .Net object to a Json string. In the following code, it creates an instance of BlogSiteclass and assigns some values to its properties. Then it calls static method SerializeObject() of JsonConvert class by passing object(BlogSites). It returns JSON data in string format.



**Deserialization**

In Deserialization, it does the opposite of Serialization which means it converts JSON string to custom .Net object. In the following code, it calls static method DeserializeObject() of JsonConvert class by passing JSON data. It returns custom object (BlogSites) from JSON data.



## Where can I use this?

DataContractJsonSerializer class helps to serialize and deserialize JSON. It is present in namespace System.Runtime.Serialization.Json which is available in assembly System.Runtime.Serialization.dll. Using the class we can serialize an object into JSON data and deserialize JSON data into an object.

## What can be done with it?

Now we can use the script found in the JSON code coming from the Masterblock. Then, we get it into the C# script to be used by a world anchor and access the data.

## Using Deserialized Text in Unity

<https://theburningmonk.com/2010/02/converting-hex-to-int-in-csharp/>

<https://www.youtube.com/watch?v=UUQydC0IimI>

Text Through a Windows Form to convert JSON to C#:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Newtonsoft.Json;

using System.Windows.Forms;

namespace WindowsFormsApp4

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

#region UI Events

private void cmdDeserialize\_Click(object sender, EventArgs e)

{

//debugOutput(txtRawJSON.Text);

deserializeJSON(txtRawJSON.Text);

}

private void cmdClearDebug\_Click(object sender, EventArgs e)

{

txtDebugOutput.Text = string.Empty;

}

#endregion

#region Debug Output

private void debugOutput(string strDebugText)

{

try

{

System.Diagnostics.Debug.Write(strDebugText + Environment.NewLine);

txtDebugOutput.Text = txtDebugOutput.Text + strDebugText + Environment.NewLine;

txtDebugOutput.SelectionStart = txtDebugOutput.TextLength;

txtDebugOutput.ScrollToCaret();

}

catch(Exception ex)

{

System.Diagnostics.Debug.Write(ex.Message.ToString() + Environment.NewLine);

}

}

#endregion

#region json functions

private void deserializeJSON(string strJSON)

{

try

{

var jPerson = JsonConvert.DeserializeObject<dynamic>(strJSON, new JsonSerializerSettings { NullValueHandling = NullValueHandling.Ignore });

for(int i = 0; i < 16; i++)

debugOutput("Product ID for Port#" + (i+1) + ": " + jPerson[i].ProcessInputs);

//debugOutput("Here is our JSON object: " + jPerson.ToString());

}

catch (Exception ex)

{

debugOutput("We had a problem: " + ex.Message.ToString());

}

}

#endregion

}

}

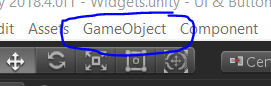
To Test this Code:

1. Create a new WebForms App
2. Click on the ‘Form1’ box (it will open another window with your script.
3. Input the above script the newly formed .cs script.

# Using JSONUpdater.cs

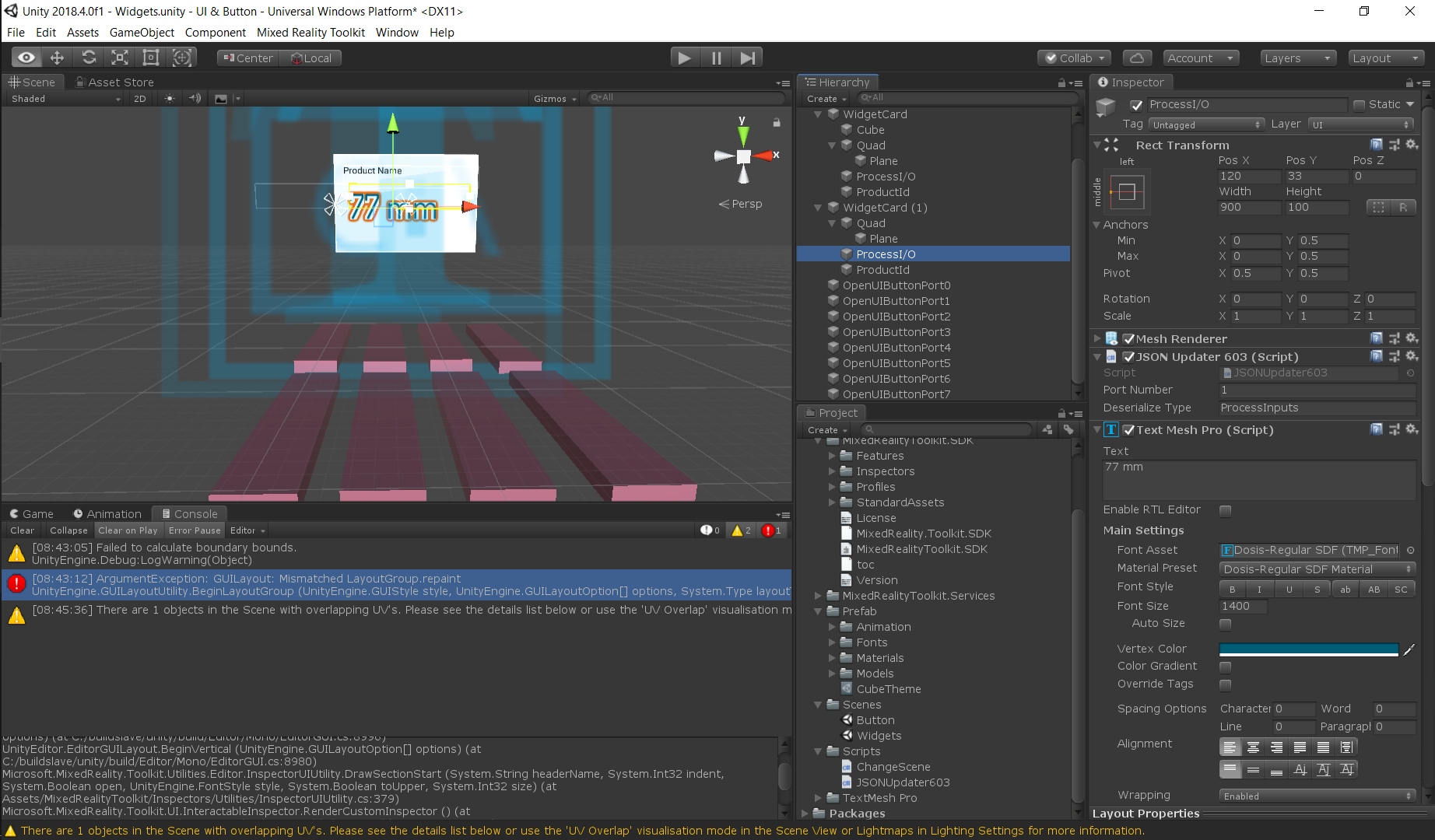
There is a lot of documentation within the Updater, but the main thing you need to know is this updates your text wherever you put it in Unity.

1. Create a game object



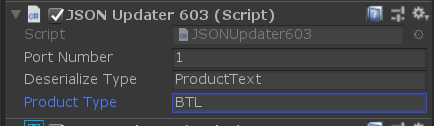
2. Click on Game Object in Hierarchy

3. Using your windows explorer, drag your JSONUpdater603.cs from Project to the Inspector

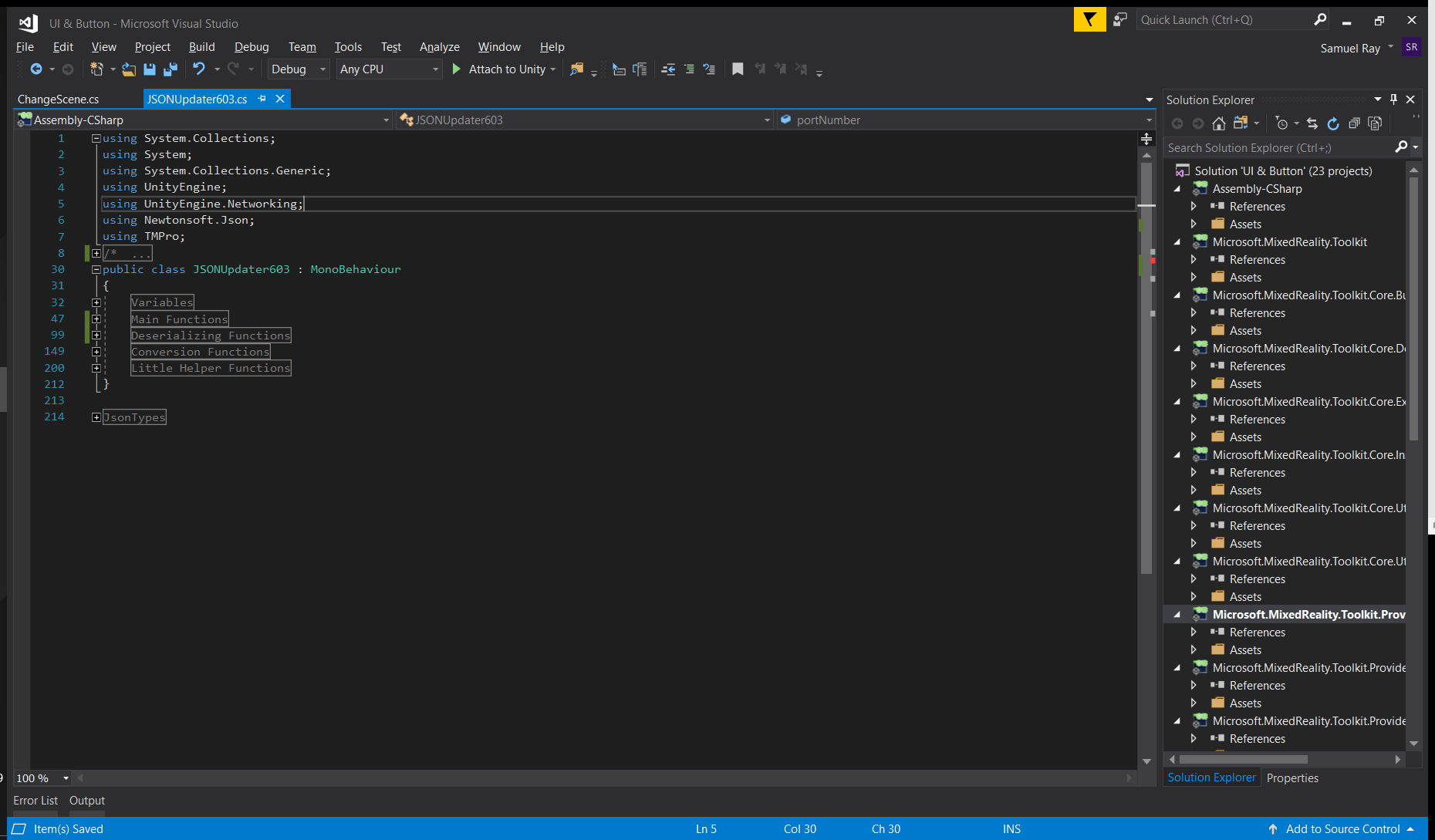


Port Number: the port you have selected for your masterblock

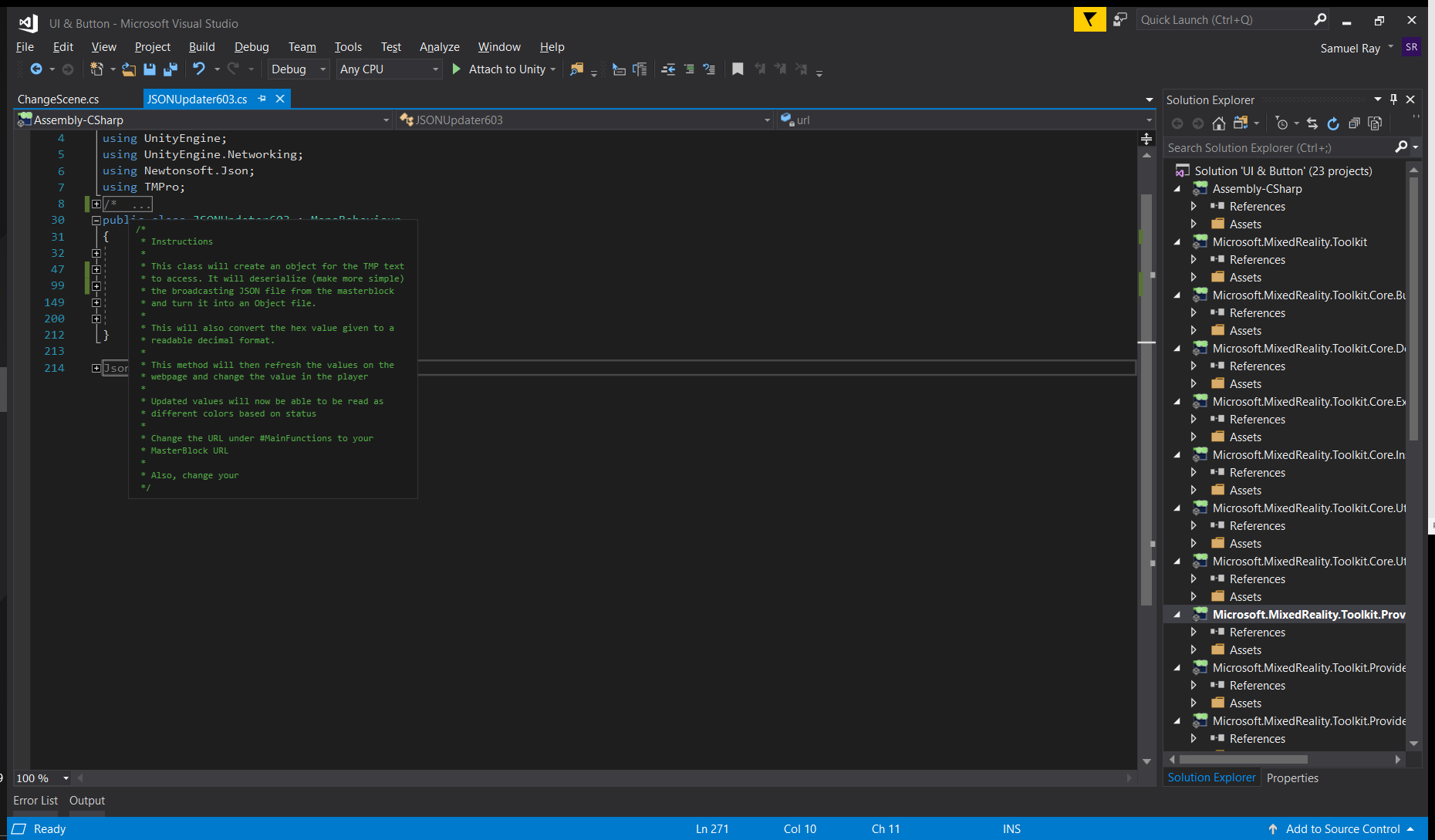
Deserialize (verb) Type: Which object from dprop.jsn you want to see in text



4. Click on the script. A new program (VS Code) will open it up for editing



5. This is what it should look like more or less. There is tons of comments that will help you read the code. First thing to click is the little box right here:



These are your instructions. Everything here will tell you how to use the JSONUpdater in Unity

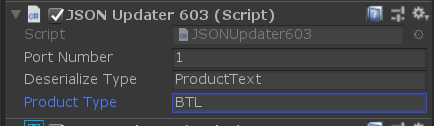
6. Click on Main Functions



7. Change your url to your [masterblock url]/dprop.jsn



8. Remember this? Well now use this to change your port numbers on your MasterBlock to your desire:



Currently working types: ProcessInputs (2), ProcessOuputs (2), ProductText  
Currently working sensors: BUS, BTL, BSP