7/30/2018

Documentation

Development & Setup

Thibaud Vegreville
FOR: REQUEA

Table of Contents

Setup the environment:

Software:

- Visual Studio 2017
- Unity 2018
- Windows 10 (you couldn't develop for HoloLens under any other OS)
- Mongo DB & SQL
- Tomcat

Setup:

- First, install Unity Hub and install the latest version of Unity 2018(2f at this this time)
- Install Visual Studio with all .NET, Game dev and Unity related dependences. It's really important but could be edited later if you miss the point.
- When the installation of Unity 2018 is done, open the git repo with "Open Project" and select the "QRCodeRebrand" folder. It will open it in Unity.
- Once open, well. It's all done! All the setup is already done in, and should not be edited (scripting backend, c# project, vr support, etc…). The project also contain the whole HoloLens Example library for easy extend and manipulation.
- In the case you have to resetup a whole new project, this is few things to do :
  - Edit Player settings in "Build Settings", and put the project in VR support (XR category)
  - Put the project on scripting backend .NET 4.*, supporting .NET and not I2LCPP
  - Put the project in Build settings on "UWP", and select C# project. DON'T TOUCH ANY OTHER OPTION! (Copy reference made a hell of a mess, and remote control is useless and will stuck the app in 2D if you don't do other manipulation with.)
- For other setup, just follow setup instruction on MSDN official documentation or on Unity support, it's all well explain. Tricky part will be already explain in the dedicated chapter.

Code logic and extensibility:

The QR app was at first conceived in a MVC pattern. Due to the lack of content (after all it’s just around 1K500 lines of codes) and the absence of any DB, it ended up in a slightly different way.

The core code is mainly separated in 3 files:

- UIDisplayerManager which was originally designed to be the UI manager in an MVC pattern. In fact, it also handle the data side because all data are volatile. The data structure is define elsewhere, however the UIDisplayerManager handle all the data flow. The reason behind is that, for such a little code, it was not really necessary to separate in first time. But in case of need, the code is written in a way that is easy to modify in a true MVC model. The data are not bind at all to the UI.
- ScannerManager, which handle all the QR code scanning part. Some code was reused from several project, git and documentation because it’s kind of hard to understand fully the .NET framework about media frame. So only few parts was adapt. At the moment, the QR code work perfectly fine but lack of performance (scanning distance and precision are quite poor). The class is pretty autonomous and you have nothing to do, except start or stop the scanner. When the scanner have a result for an image, it throw an event that could be listen and be register (delegate).
- REQUEA_LIB which is the API worker designed for working with the online API (com.requea.iot.api). This class handle all the connection request stuff and display few public function that allow you to do different type of request on. All function are asynchronous, except the Unity ones (for reasons explain here).The class is standard and could be easily extend. Also, it does not contain the data structure that is retrieve from the API point: the interest is that, in case of rework on the API, you don’t have to modify the API worker, but just update the data structure stored in GlobalDefinitionScripted file. Obviously, the server where the lib make its request has been hardcoded while it was just a prototype. So be aware that you ***HAVE TO*** change the API url. For others parameters, it’s should be standard and no need to update it.

There’s few other classes that handle interaction and provide easy-to-use delegate interface in order to listen on object (e.g. GlobalEventProvider).

Tricks and tips:

- Unity is not compatible with the async function in .NET framework. In fact, Unity is a monothread application using a handmade kind-of asynchronisation via the “yield” mechanism. When you have a coroutine (that cannot return anything and must have a callback if you want a result), you can use the yield keyword in order to tell to Unity to look at this function at each frame, and keep up the good work. To put it short, Unity will enter the pseudo-async function at each frame, and continue the work before rendering the frame. So, when you implement a function, you have to be aware of :
    - You can’t use any Unity call outside of the main thread. So, if you use async-await from .NET, you can’t use any Unity API’s function in! (But you can still use it for .NET work only)
    - You can’t mix Yield and async-await : it just doesn’t work. That’s it.
- If you have to import an external component in Unity, there is few things to know :
    - Unity is divided into the player, the editor, and the “runtime”. The editor provide the capability to play your scene into the Unity Engine and test it in. The player expose all the method globally related to Unity, and the “runtime” is the resulting build of your unity project.
    - When you try to add DLL or plugin to Unity, you have to be aware of where you can use it. Some are not compatible with the Editor and so will prompt error in the Engine. You have to enclose the code using those DLL into pre-comp instruction like #if !UNITY_EDITOR to avoid it. e.g.: It’s the case for all Nuget package so far.
    - Unity’s bundle are usually well imported, but could be not reckognize or compiled in VStudio.
- When you have to test your code, do it on the HoloLens. As said previously, some external package (here, zxing for example) don’t work under Unity editor. By the way, the dll zxing that is in plugin folder should be delete when you compile the project. This dll avoid Unity to spread error all over the project when you’re in editor mods. When you compile and open the resulting project in VStudio, go in Reference list and delete zxing, then import the Zxgin Nuget package. You will have to do this little manipulation each time you compile the projet (For the nugget part, if you just compile over the same project (choosing the same folder each time), the nugget plugins will not disappear. So, just delete the zxing dll from Unity and you will be fine).