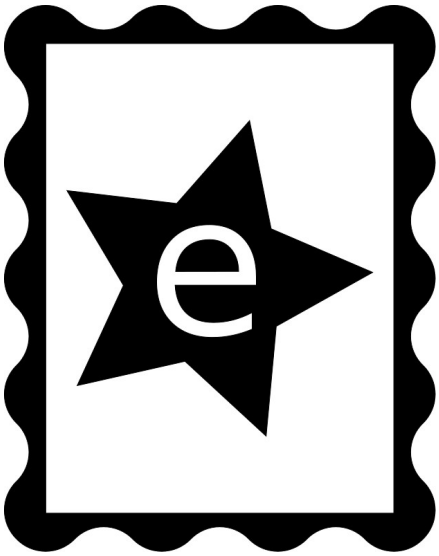

Connecting C++ native DLL to C# for Unity

(eToile 2016)

Introduction

This package includes everything you need to know in order to deploy x86 and x64 DLLs for Unity integration. The included DLL project compiles under Visual Studio Community 2015, which can be downloaded for free from here:
https://www.visualstudio.com/en-us/products/visual-studio-community-vs.aspx

You'll need to install the "C++ language" to be able to compile DLLs in Visual Studio.
In any case, eToile recommends to make a complete installation in order to avoid problems with the DLL example project and any further product development you may want to make.

As you may expect, this DLL integration may also run on other development platforms because there is not any "Unity-Only" feature. This feature is not covered in this document.

Unity project structure

The project includes two versions of a single C# file for Unity. They are attached to the main camera (DllAccess.cs or MFCDllAccess.cs) and you can enable them separately.
The "DllAccess.cs" source code shows how to wrap the retrieved data from the DLL. You can encapsulate it for your own projects if you need.
Please note that in some data types a "Marshaling" is needed in order to retrieve the data correctly.

Example:

```
[DllImport("UnityTestLibrary")] static extern IntPtr GetWString();
string data = Marshal.PtrToStringUni(GetWString());
```

The `[DllImport]` attribute loads the DLL into memory for further use. You can add the ".dll" extension to the file name if you want.
The `GetWString()` function returns a pointer to a wide char C++ string. After that, the string is correctly wrapped and copied into the C# `string data` variable for further use.
To use `Marshal.PtrToStringUni()` you must include:

```
using System.Runtime.InteropServices;
```

at the header of your C# file.

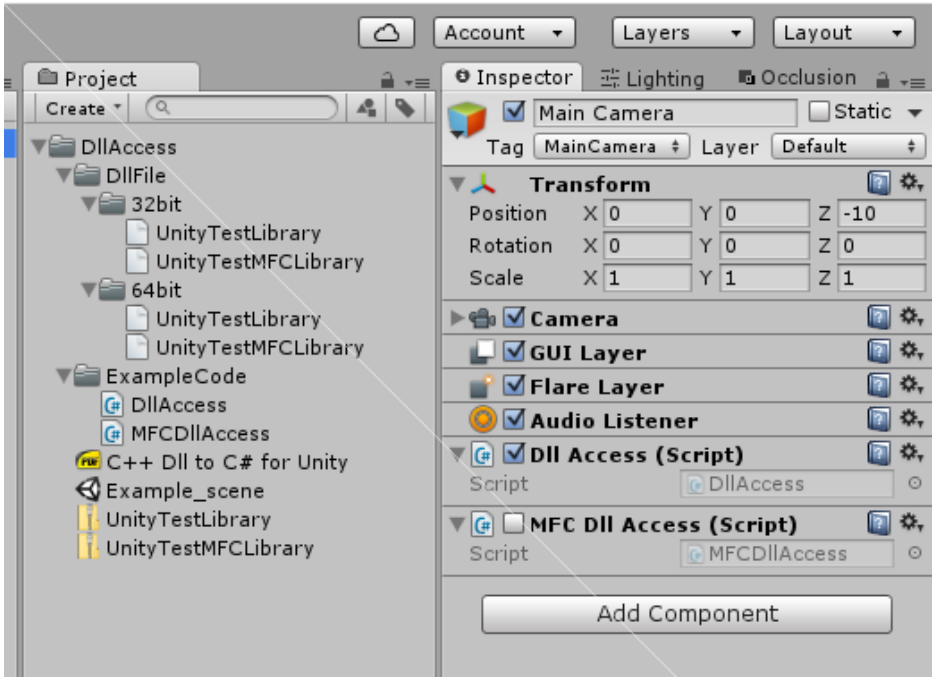

Fig 1: Complete structure of the Unity project

Visual Studio project structure

The Visual Studio project (UnityTestLibrary) is included as a ZIP file in the "DllTestProject / Assets / DllAccess / DllFile" folder. Decompress it and open it in Visual Studio double-clicking the "UnityTestLibrary.sln" file.

Once opened, you will see the two important files of the project in the "Solution Explorer" window:

- "UnityTestLibrary.h"
- "UnityTestLibrary.cpp"

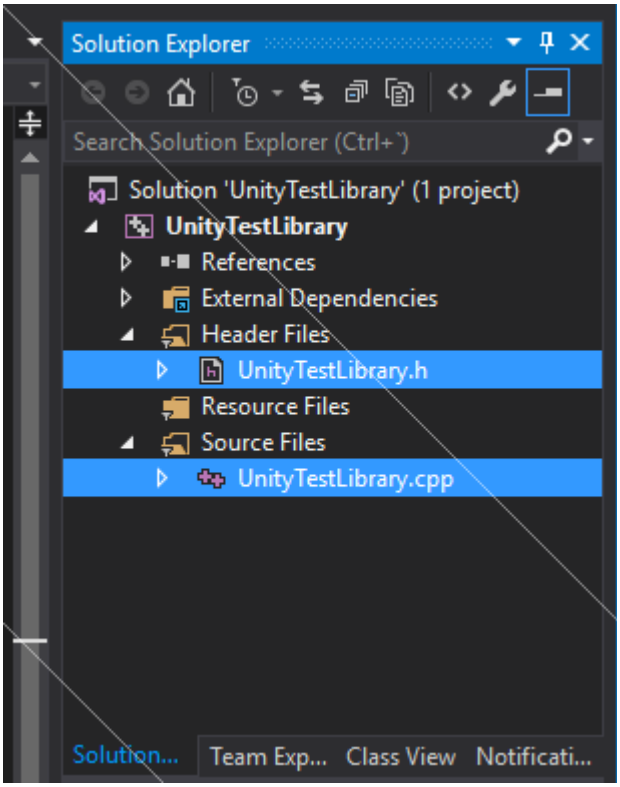

Fig 2: Relevant files in the VS project

Note that the "External Dependencies" in this project are there due to the use of `#include <string>` and `#include <codecvt>` for string manipulation.

The “UnityTestLibrary.cpp” file contains the function definitions. Here is where your code goes.

```
// Function definitions:
extern "C"

// Common types:
int __stdcall GetInt(int n)
{
    return 2 + n;
}
float __stdcall GetFloat(float n)
{
    return 0.5f + n;
}
bool __stdcall GetBool(bool n)
{
    return !n;
}
```

Fig 3: Example of definitions

The “UnityTestLibrary.h” file contains the function declarations. This file declares the entry points for your DLL. Note that the declarations doesn’t includes the variable names in the parameter parentheses but only the variable types.

```
// Custom function declarations goes here:
#ifdef __cplusplus
extern "C"
{
#endif
    // Just argument type of functions must be declared (no names):
    int __stdcall GetInt(int);
    float __stdcall GetFloat(float);
    bool __stdcall GetBool(bool);

    const wchar_t* __stdcall GetConstWString();
    const wchar_t* __stdcall GetWString(char*);
    const char* __stdcall GetString(char*);

    byte* __stdcall GetByteArray();

#ifdef __cplusplus
}
#endif
```

Fig 4: Example of declarations

Ending

If you find errors in the package or document, please contact me at jmonsuarez@gmail.com and the product will be updated. Any comments, critics or improvements are also very welcome.

Hope you enjoyed the tutorial.

eToile, 2016