Research Findings: Unity Plugins

Date 8 August 2015
Researcher Paul Lee

---

1. Background

Our client has asked us to create a GPS plugin as a first part of the 'proof of concept' application they would like us to create.

---

2. Objectives

Get an understanding of what a plugin is, how to write a plugin, what makes a good plugin.

---

3. Approach

Look in to Unity tutorials, read articles and online resources.

---

4. Findings

4.1. From Unity – "Plugins allow us to access native code (either managed or unmanaged) within Unity. This gives us access to powerful hardware not available on all devices, such as trackpads, cameras, an Oculus Rift, etc. (Unity3d.com, 2015)

4.2. A plugin also could be known as an add-in, extension or add-on. A plugin is a software component that adds a specific feature to an existing software application. When software allows plugins to be added it lets the application have lots of customisation.

4.2.1. Common software where plugins are found are in web browsers, new features such as virus scanners, ad blockers, script writers etc. are all examples of plugins. A very well-known plugin is the Java plugin which lets the browser launch java apps.

4.2.2. It encourages 3rd part developers to create more exciting features for an application.

4.2.3. It reduces the size of an application as users can choose which feature they want or need and do not have to install ones they don't need.

4.2.4. The application is easily able to support new features.

4.3. In Unity you normally use scripts to create functionality but you can also use plugins that have been created outside of Unity. There are two kinds of plugins that you can use in Unity: *Managed Plugins* and *Native Plugins*.

4.3.1. *Managed Plugins* are managed in .NET assemblies created with tools like visual studio. They only contain .NET code and can't access features that are not supported by the .NET library. Managed code is able to access the standard .NET tools that Unity uses to compile scripts, there is little difference between a managed plugin and a Unity script, expect for the fact it is developed outside of Unity.

4.3.2. *Native Plugins* are platform-specific code libraries, they can access features like OS calls and third-part code libraries that would otherwise not be available in Unity. But these libraries are not accessible to Unity's tools in the way that managed libraries are. You will get standard compiler error messages if you forget to add a managed plugin file to the project but only an error report if you do the same with a native plugin.

---

5. Further Investigation

5.1. Creating a test platform so that we can test the plugins that we have created.

5.2. How do we make sure that it can be used in any application/platform?

5.3. Coding standards for plugins

6. Recommendations

6.1. Make sure that we are not creating any ‘god’ classes. This is a ‘one-class-fits-all’ kind of mentality. Creating multiple classes will produce much simpler, easier to read code.

6.2. Even just a separate class for a simple operation is nicer than sticking it somewhere it doesn’t belong.

7. References

Unity3d.com,. (2015). *Unity - Writing Plugins*. Retrieved 8 August 2015, from https://unity3d.com/learn/tutorials/modules/beginner/live-training-archive/writing-plugins

Wikipedia,. (2015). *Plug-in (computing)*. Retrieved 8 August 2015, from https://en.wikipedia.org/wiki/Plug-in_%28computing%29

Technologies, U. (2015). *Unity - Manual: Plugins*. *Docs.unity3d.com*. Retrieved 8 August 2015, from http://docs.unity3d.com/Manual/Plugins.html