UNIVERSE

TECHNICAL DOCUMENTATION (v2.0)

Contents

Introduction

One of the first problem someone encounter while developing with Unity is what could be called "level of persistence". At the root, you have the Unity Player which is never unloaded; it can be seen as being always there. On top of the Player, there's a collection of Unity managers, such as the Sound Manager and the Input Manager. They are also persistent and are never unloaded as long as the game is running.

The developer doesn't have access to that Player layer. Instead, he has two "virtual" levels which are the scenes and the GameObject. You usually load and unload a scene as the context of the game change. As for GameObject, you create and destroy them all the time following the current gameplay.

However, it happens quite often you would need a way to have a fully persistent manager on your own. Sadly, Unity doesn't give you access at creating your own manager the same way they did for their own. This package is our solution to this issue.

Universe

In other game engine we have used, the layers of persistence were separated as followed;

- **GameObject**; The lowest item, with a very short life span.
- **World**; Scene that holds GameObject or other object. Are loaded and unloaded once in a while when the context of the game change. It has a medium life span.
- **Universe**; Is loaded when the game start, and stays as long as the game is running. It has the longest life span.

In Unity, the concept of Universe is not accessible. However, it is possible to build one for our needs. Usually, one will go towards the singleton or the static pattern. Both may work, but they each have their flaws and shortcoming.

A static class will lack any parameters and proper serialization.

A singleton pattern requires maintenance and to be setup in scene to work properly.

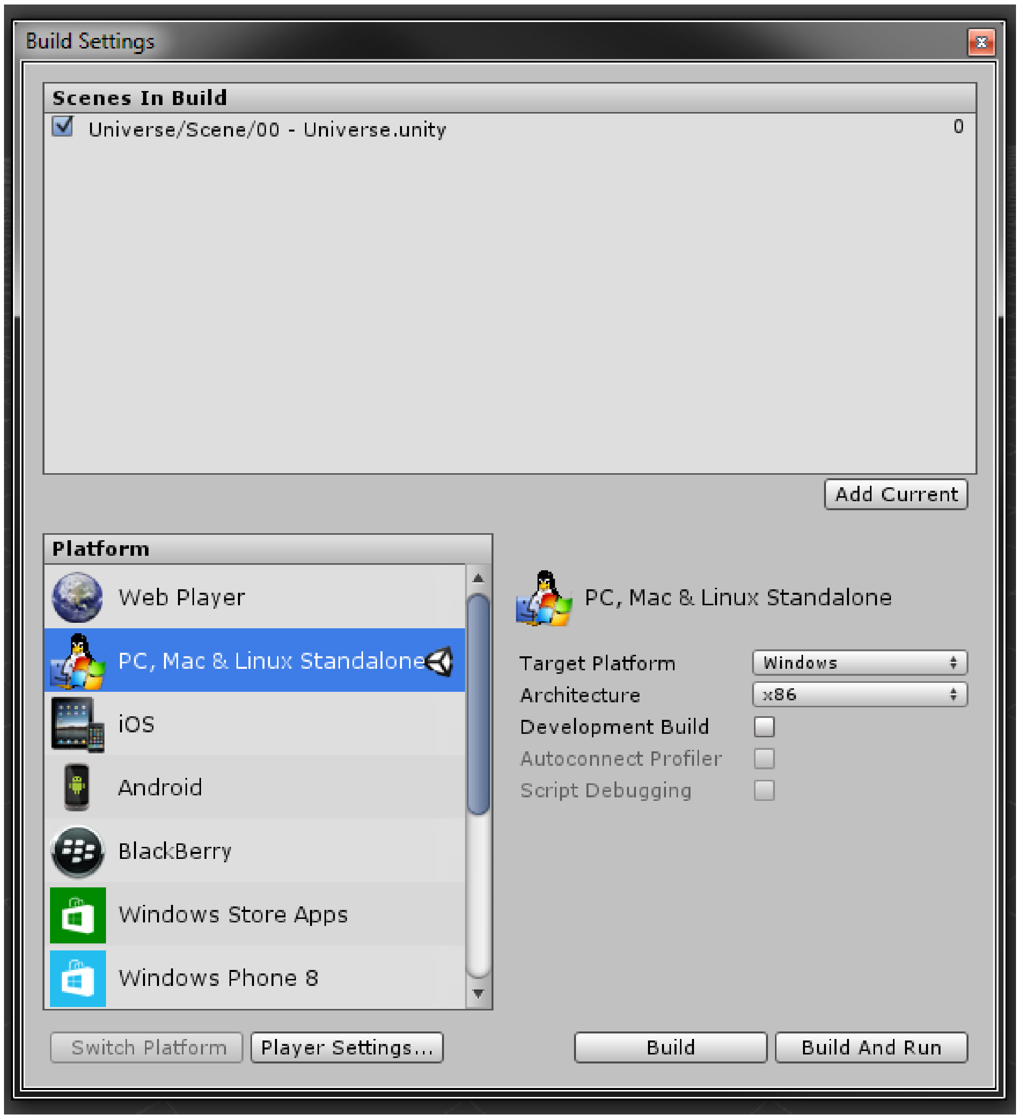
The Universe in this package works with a self-referencing singleton pattern, coupled to a serialization-by-reflection. There's many advantages to this;

- No maintenance; no need to add or update script, GameObject or prefab in all your scene. No need to create any prefab, the package does it for you without bloating your scene with extra data.
- All Scenes work; create an empty scene, press play, and it works! Managers are automatically loaded by an editor tool which detected that you're in the editor and need your managers.
- Accessibility; Each manager has its own prefab to hold its data in the Resource folder. They are all at the same place, and easy to edit.
- Message; a static class cannot receive external Message, which is a must for platform like iOS/Android. Being on a prefab, every manager is ready to receive your messages.
- Auto-implementation; the singleton pattern is automatically implemented, you don't need to do anything. All your managers are accessible with their data at all time.

How To Use

Step 1

Using the Universe package is extremely easy. First, you need to set the first loaded scene of your build as being "00 - Universe", a scene that is provided in this package. This scene has a single GameObject with the Universe script on it. Its only task is to load all the prefabs of all the managers and keep them alive. It must be at index 0.


Build Settings

Scenes In Build

☑ Universe/Scene/00 - Universe.unity 0

PC, Mac & Linux Standalone

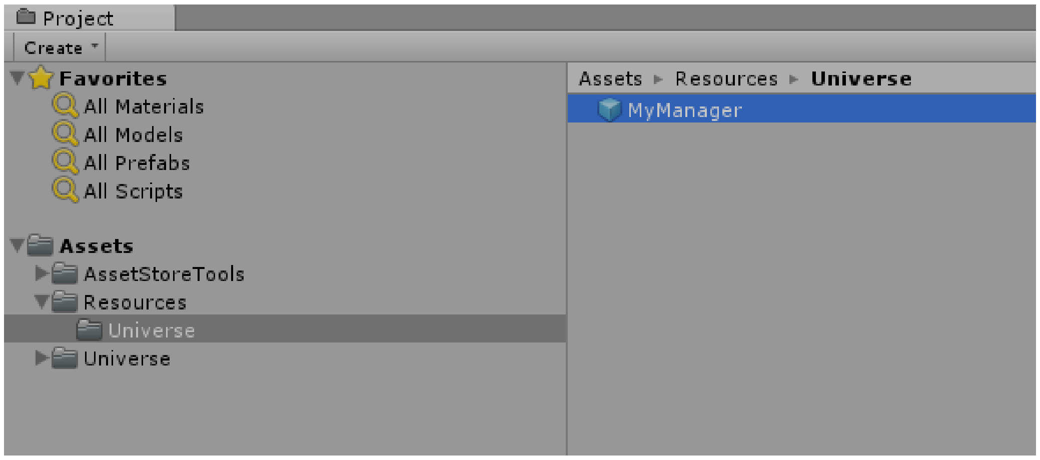
Target Platform	Windows
Architecture	x86
Development Build	☐
Autoconnect Profiler	☐
Script Debugging	☐

Step 2

Second, have your manager you wish to exist for the lifespan of the whole game to derive from the Manager<> class;

```
public class MyManager : Manager<MyManager>
{
    public float myValue;
}
```

The next time the code reload, a new prefab will be created;



This prefab is named the same way as your manager class. Please do no rename it, otherwise a new prefab will be created.

That's it! All your managers will always be loaded, no matter which scene you use, in the editor or in your build. Added a new manager is as easy as creating the class and pressing play.

Thing to Consider

When starting from a random scene, if you try to access your manager on the OnEnable, Awake or Start method, you will get a NullException. This is because the object in your scene are awaken 1 frame before the Universe editor tool can do its job. The workaround is this kind of pattern;

```
        private void Start()
        {
            StartCoroutine(WaitForManager());
        }

        private IEnumerator WaitForManager()
        {
            while (MyManager.Instance == null)
                yield return null

            // do manager related initialization
        }
```

Contact

While the Universe is a fairly simple tool, it surely still has room for improvement. To send us any bug report or feature request, contact us at;

Email : admin@lightstrikersoftware.com

Website : www.lightstrikersoftware.com

Package Revision

1.0

- Release

2.0:

For some reason I never actually update this package on the store while over the last two years the version we use internally changed drastically.
We shipped 6 games using this tech. However, the version that was on the Store was flawed and had a few issues that had been fixed a while ago.

We simply didn't update the store package.

I deeply apologize for not updating this asset on the store earlier!

[CHANGES]

- Manager prefabs are created on the code reload, not when pressing play in the editor.
- The tools relies on the index of the scene (0) instead of checking for the "00" in the scene name. That was dumb.
- Namespaced all the classes under the Universe namespace. Class collisions are bad.
- Replaced the ManagerBase class by a IManager interface.
- Removed the whole concept of "rogue" universe.

[FIXES]

- Managers now load properly once deployed in a build. What a terrible bug - my apologize again. (Why nobody told me this?)
- The tools now relies on the types instead of object names, which is far more reliable. Way less GameObject.Find... like none.
- Added support for Unity 5.3 SceneManager.
- Fixed the "Missing Behaviour" warnings.