On The Wonderfulness of Scriptable Objects

There are already a vast number of online articles about Unity's Scriptable Objects. Most of them tell you the same things:

- They're assets and can't be attached to GameObjects.
- Affect the asset in the Project and the S.O. is also affected.
- You can't really have stateful runtime code in them.

[Note: some of the following is simplified for readability]

But that's misleading. While it's true that the S.O. class definition resides in the Project folder, so does every other piece of code, be it a component, a static class, whatever.

One big difference is that a component or any sort of class is just a text file. A S.O. asset is inspectable and changes to it are saved in the Asset Database (ADB).

So it is true that changes to a S.O. at runtime affect the asset in the ADB.

Change your view

Think of a S.O. as a GameObject without a transform and missing a few events. And treat it like a prefab.

I first 'discovered' this scheme when trying to figure out how to have editable persistent Tilemap tiles. I wanted to be able to have editable fields in Tiles right in a scene rather than using Tiled (etc) and importing.

In Editor-edit mode, it turns out that if you merely clone (Instantiate) a Tile S.O. (a tile is a S.O.) and then add it to a Tilemap via SetTile (or any other method) then that clone is:

- Independent of the asset in the ADB.
- Saved with the scene and loaded when the scene is loaded by Unity.

And these tiles can have any sort of code and data that you want; have state, respond to events and so on.

There are a host of other issues to solve for using tiles this way, such as how to inspect tiles containing fields not in the Tile class, but that's talked about in the TilePlus Toolkit docs elsewhere on this site.

This approach can be extended to other uses:

- As DLLs, that is, as dynamically loaded services and/or singletons etc.
- As loadable code blocks created on-demand at runtime.

The 'DLL' approach is used in the TilePlus system for Services (Runtime Scriptable Object Singleton).

The 'loadable blocks' approach is used in the Tilemap Layout subsystem where blocks of tiles on multiple maps are added and deleted as the camera moves. This uses SOs called "Zone Managers" which are created and destroyed on demand depending on how many different Layouts are running.

Although upcoming Unity versions are going to 'fix' the reload time in-editor, currently these two approaches save reload time by avoiding static class initialization and so on. At runtime, lazily cloning SOs as needed improves startup time and can save memory/reduce GC issues.

But, Update?

And you can have Update, too. Using a modified GameLoop allows you to inject your own Update or other events wherever you want in the GameLoop.

Since the state of data in a S.O. that's part of a scene, such as a cloned Tile instance, is frozen at the time the scene is saved then the same issues arise as you have with any other runtime state changes. Changes in data can preserved with your other game data and restored the next time that a level is loaded. Just Json-ize it.

If you're curious about how this works and want to see some examples, check out the other documentation on this website. The Toolkit asset itself is free.

Revision #11
Created 2026-05-01 14:22:52 UTC by Vonchor
Updated 2026-05-01 15:43:51 UTC by Vonchor