Lifetime of a TilePlus tile

TilePlus lets you treat a Tile script much like a script attached to a GameObject: but Tiles are not GameObjects. It's easy to forget that a Tile is based on the ScriptableObject class. Here's part of what the Unity manual says about Scriptable Objects:

```
Just like MonoBehaviours, ScriptableObjects derive from the base Unity object but,
unlike MonoBehaviours, you can not attach a ScriptableObject to a GameObject.

Instead, you need to save them as Assets in your Project.
```

What's left out of that statement is that you cannot attach a ScriptableObject to a GameObject as a Component. But you can attach it as a reference via a field in a script. When you do that, the reference is to the asset in the project folder. Clearly, you can create an actual instance of the Scriptable Object in memory, and it can be placed in the reference 'slot'. That's essentially how TilePlus tiles work:

- You paint the tile (or place it programmatically). It's in the ASSET state at that time.
- The tile's StartUp method sees that the state is ASSET and queues a cloning request in TpLib.
- The tile is cloned at the next Update and the clone is placed at the same location, replacing the tile asset reference in the Tilemap.

The cloning only happens once: when the tile is placed by an editor tool like the UTE or Tile+Painter or by code.

- Move the tile from one place to another (Cut/Paste): no cloning
- Copy/Paste: the new tile is cloned in TpLib.

Since the clone is referenced in the Tilemap now, it's saved with the scene.

But it's still not a GameObject: most of the events are missing. The only really useful events are OnEnable and OnDisable. Fortunately, Tiles have a StartUp method where it is passed a reference to the parent Tilemap and the position. Follow along by examining the StartUp method of TilePlusBase.cs (not every line is discussed):

- A reference to the parent Tilemap for the tile is cached.
- The tile's position is cached.
- A flag is set if the position has changed (for example, if you'd moved it using the Cut/Paste function of TpLib). From that point there are two code branches depending on whether the tile is already a clone.
- Clone: check for a proper GUID and register the tile with TpLib.

- Asset: queue for cloning in TpLib. The clone replaces the asset reference in the Tilemap via Tilemap.SetTile. This causes StartUp to be executed again.

From this point in time the tile is essentially passive. When the Tilemap calls GetTileData and GetTileAnimationData the information returned from those methods are copied into the Tilemap data structures.

If you message the TPT tile it may perform other actions such as messaging other tiles, tweening, or even deleting itself. But aside from your code causing such actions the tile can't do anything since it doesn't get any events such as Update.

Events

- OnEnable, which generally will execute before StartUp, lets you set up initial conditions. Examples of this can be seen in the animated tile classes.
- OnDisable, can be used for cleanup.
- OnDestroy, while theoretically available, is not useful since it’s not called at any predictable time unless one were to Destroy tile instances programmatically. You’ll not see it used in TPT tiles at all.

Note that the TpLib.MaxNumClonesPerUpdate property controls how many cloning operations are executed on each Update (default is 16). This allows performance optimization. See this.

TPT Tiles are always cloned when painted in the Editor or when added programmatically during the application’s execution. When a Tilemap is Bundled, archived TPT tiles are ‘Locked’ assets. Cloning also occurs when loading TileFabs or TpTileBundle contents to your scene. For efficiency, this is done inline, within the loading process; and all at once.

In other words, TPT tiles will only request cloning when painted in-editor or at runtime, in code. If you want to paint numerous TPT tiles at runtime, consider using TileFabs and Bundles.

This should factor into your setting value for TpLib.MaxNumClonesPerUpdate.

For example, with the default value of 16 for TpLib.MaxNumClonesPerUpdate, painting 1024 TPT tiles will cause 1024 cloning operations. If only 16 are added during each Update, then assuming a 60 Hz Update frequency this would take about 1 second.

Knowing this, you might want to dynamically change this property’s value when you load a scene or instantiate a prefab.

If you’re not painting huge numbers of TPT tiles at runtime, then you don’t need to worry about the value of TpLib.MaxNumClonesPerUpdate.

InvokeRepeatingUntil

If you really need a repeated `Update` you can use TpLib's InvokeRepeatingUntil. This will automatically terminate under your control or when the tile is deleted.

Revision #10
Created 2025-07-14 11:18:09 UTC by Vonchor
Updated 2026-07-10 14:50:11 UTC by Vonchor