Chunkifying Details

When a Bundle is 'Chunkified' by the Layout/Chunking system the Bundle creates a cache of data in each TileBundle for rapid loading or re-loading of tiles using the `TileChangeData` and `Tilemap.SetTiles(TileChangeData[] tileChangeDataArray, bool ignoreLockFlags)`.

When the layout system wants to load a chunk of tiles to load in this fashion it uses the Bundle's `LoadCachedTileChangeData(RectInt locator, Tilemap map)' method.

That method combines the cached Unity tile `TileChangeData` asset references with the cached TPT `Locked` state `TileChangeData` for that locator.

During combining, any TPT tiles are cloned in-line. The resultant array of TileChange data is loaded to the map referenced in the method call.

After the data are loaded on the Tilemap, the new clones are deleted from the data.

To repeat: the cached TPT tile references are cloned prior to loading to the Tilemap and deleted right after the loading completes.

What this means is that *every* time that the chunk of tile data are retrieved in this fashion *new, fresh* TPT tile instances are created. Immediately after `SetTiles` is used the clones are deleted. But they're not gone. They're now referenced in only *one* place: the Tilemap itself.

Why do this, or Why Bother??

Since each TPT tile is an independent instance, once it's loaded to a Tilemap its internal state may change during the time it's actually in the Tilemap; i.e., until it's unloaded. Hence, we really don't want to re-used these modified tiles once they're unloaded.

If these were unloaded and re-loaded later then the modified fields will still be modified, which can cause all sort of bizarre errors that would be hard to troubleshoot. Obviously you'd like an unmodified clone of the corresponding locked tile in the Bundle asset.

The end result here is that the TPT instance which gets loaded to the Tilemap has no references anywhere else. That way, when the tile is eventually unloaded it becomes garbage collected properly rather than be a potential memory leak.

You Need to be Vigilant!

This can be affected by your own code. If you cache a TPT tile reference, it won't be garbage collected and it may not even be actually placed on a Tilemap anywhere when you try to access its fields and properties, some of which may be unexpectedly null.

- **This can occur with ANY cached UnityEngine.Object reference!**

One approach to follow is to use Unity's pools to re-use Lists of TilePlusBase when you need to provide a list to one of the TpLib query methods. When creating such a 'ListPool' you should ensure that the Release callback clears the list. Get used to the idea of wrapping these methods with `using` statements to ensure that the lists are returned to the pool.

This type of code block is peppered throughout TilePlus and TilePlusPainter. Examples of pools can be found in the TpLib source.

Revision #6
Created 2025-08-09 14:37:11 UTC by Vonchor
Updated 2025-09-21 18:04:28 UTC by Vonchor