TpZoneManager

TpZoneManager is a chunk management subsystem. Your code interacts with instances of the TpZoneManager class, which are Scriptable Object instances created at runtime by using TileFabLib. The Layout demo programs illustrate how to use the system.

TileFabLib implements some simple management features for creating, locating, and destroying TpZoneManager instances. Let’s call them ZMs to avoid me having to type TpZoneManager over and over.

To save memory, the data structures used for management are not initialized until you enable the subsystem by setting the TileFabLib.EnableZoneManagers property to true. This enables the subsystem and allocates memory to manage the ZMs, which you create with `TileFabLib.CreateZoneManagerInstance`.

For efficiency, the GUID remapping data tables discussed earlier are also maintained within TileFabLib.

```
public static bool CreateZoneManagerInstance(
  out TpZoneManager? instance,
  string iName,
  Dictionary<string, Tilemap> targetMap)
```

TileFabLib.CreateZoneManagerInstance requires that you provide a string as a unique identifier, and a mapping between Tilemap names and Tilemap instances. The ZM instance is placed in the out-parameter instance.

Don’t use the same Tilemaps with different ZMs.

Once created you can always get this specific ZM instance by using TileFabLib.GetNamedInstance.

Delete a named ZM with TileFabLib.DeleteNamedInstance. If you delete a named instance and there are no more, TileFabLib disables the subsystem and releases memory subject to GC.

Once you have a ZM, you initialize it with ZM.Initialize, providing the chunk size, the world origin coordinates (in the Tilemap address space), and an initial number of chunks expected.

For a single ZM instance, restrictions apply:

1. Chunks (TileFabs) all need to be all the same size: square, with even-number dimensions such as 64 x 64.
2. The smallest chunk size is 4x4.

3. Chunks are always aligned to the super-grid defined by the chunk size.
4. A Tilemap should not be used with more than one ZM. A ZM is unaware of Tilemap positions that may have had Zones filled or deleted by another ZM, and conflicting chunk sizes between different ZMs would naturally be problematic.

Each ZM can be set up completely differently. Just don’t cross the streams…. uh, share Tilemaps between ZMs.

When you provide a ZM instance to TileFabLib.LoadTileFab it sends the results of the load to the ZM, which creates a ZoneRegistration.

Why are Chunks Square?

Chunks must be even dimensioned due to integer math. If a Chunk had an odd size, division could introduce a positioning error. They don’t have to be square (they could be rectangles), but for this implementation they must be square with even dimensions because it makes the logic and math easier and much faster. It’s also more intuitive.

Revision #3
Created 2025-07-08 14:04:30 UTC by Vonchor
Updated 2025-07-22 17:50:14 UTC by Vonchor