TileFabLib

TileFabLib.LoadTileFab loads the Bundles referenced by a TileFab asset. The TileFab asset has references to one or more Bundle assets, which include all the information required to recreate the tiles. There are both Async and non-Async overloads for this method.

TileFabLib.LoadBundle loads the tiles from a single TpTileBundle asset. You normally don't need to use this, although you can if you just want to load tiles from a single Bundle (this is how Tile+Painter paints single Bundles). Otherwise, use LoadTileFab: a TileFab is created even if you only bundle one Tilemap and LoadTileFab is easier to use.

If you were only loading one Bundle to a Tilemap you'd intrinsically know which Tilemap to use as a target for placing the tiles extracted from a Bundle. This is why you can use the Tile+Painter to paint a Bundle anywhere you want to.

However, the normal use case, and the one supported by TpZoneManager, is to use a TileFab. That asset is a wrapper for any number of Bundles, along with the tags and/or names for the source Tilemaps that the Bundles were created from.

Why archive these two items? Because that information tells you what the Tilemap is for each Bundle; that is – where do you paint a Bundle that's part of a TileFab?

Using the original Bundle assets' names and tags, LoadTileFab tries to locate the target tilemap; first with its tag, then by name. For faster performance, a mapping between stored Tilemap names and specific Tilemap instances can be passed to LoadTileFab. This avoids searching for GameObject tags or names.

Each TpTileBundle asset is evaluated and the destination Tilemap must be found. The search is performed in the following order:

- Provided by the name-to-Tilemap instance mapping. (fastest)
- Provided by using the Tag to Find the parent GameObject of the Tilemap.
- Provided by using the Name to Find the parent GameObject of the Tilemap. (slowest)
- If a TPT tile reference is passed-in to LoadTileFab then the parent Grid of the tile is found, and the Tilemaps which are children of that Grid are examined for matching names (variable speed but faster than using Find).

If all those methods fail, then LoadTilefab silently fails, nothing happens, and no tiles are loaded from that TpTileBundle asset.

You can see this in action if you try to paint a TileFab containing multiple Bundle assets and one of the named Tilemaps isn't present. If no Tilemap is located, then there's no way to load the tiles and that Bundle asset is ignored: no preview or painting. If no Tilemaps are located for any of the Bundles, then nothing is painted at all, and no previews are possible.

If you’re using Tile+Painter to paint a single Bundle’s tiles, the destination Tilemap had already been selected in the leftmost column. Hence, you can paint a Bundle on any Tilemap quite easily.

If you want to ensure that loading TileFabs or Bundles recreate what you’re expecting, bear in mind that the Tilemap Component and Tilemap Renderer Component settings matter too:

- Tilemap: Frame rate, Color, Anchor, Orientation
- Tilemap Renderer: Sort Order, Sorting Layer and Order in Layer

If any of these are different than the setup when you originally archived the tiles, then the resulting visual appearance after the loading will be different.

Using LoadTileFab

LoadTileFab has several overloads for both synchronous and async use.

Let’s examine the method parameters for LoadTilefab:

- tileParent: The parent of the calling object (Only needed if calling from a TPT tile, otherwise ignored and can be null)
- tileFab: a TpTileFab asset reference
- offset: The Offset from tiles' stored positions – you use this to set the location where you want the tiles placed. For example, if you used Vector3Int(100,200,0) then the tiles will be placed relative to that location.
- rotation: optional rotation – unimplemented
- fabOrBundleLoadFlags: A set of control flags for LoadTileFab. See below.
- filter: a Func returning a bool. If non-null, this Func is used to filter out tiles that you don’t want.
- targetMap: A dictionary mapping tilemap names as found in the TileFab to actual Tilemap instances. Used to override the names from the Tilefab.
- zoneManagerInstance If using a ZoneManager, pass its instance.

the async version adds this parameter:

- intervalBetweenBundles For the Async version, a wait time between bundles.

Returns: Instance of TilefabLoadResults class. If null is returned, then there was an error of some kind.

fabOrBundleLoadFlags is a value from the FabOrBundleLoadFlags enumeration. These are Flag enums so they can be ORed. The combined value ‘Normal’ is the common case.

- None: usually not used
- Load Prefabs: Normally true
- Clear Prefabs: Normally false

- Clear Tilemap: Normally false
- Force Refresh: Normally false.
- New GUIDs for TPT tiles: Normally true
- FilterOnlyTilePlusTiles: Normally true
- NoClone: Normally false.
- MarkZoneRegAsImmortal: Normally false
- Chunkified: Normally false
- NormalWithFilter: LoadPrefabs | NewGuids | FilterOnlyTilePlusTiles
- Normal: LoadPrefabs | NewGuids
- ChunkifiedDefault: LoadPrefabs | Chunkified | FilterOnlyTilePlusTiles

The meanings:

- forceRefresh: Executes Tilemap.RefreshAllTiles after the tiles are loaded.
- loadPrefabs: Load any prefabs found in the TileFab
- clearPrefabs: Delete all prefabs attached to the target Tilemap's GameObject.
- clearTilemap: Clear all tiles on a target Tilemap prior to loading new tiles.
- newGuids: Provide new GUIDs for all TilePlus tiles. Use when placing these assets at runtime to avoid duplicate GUIDs. There's a discussion about this a bit farther down.
- filterOnlyTilePlusTiles: If the filter is provided then the filter is only applied to TilePlus tiles, which is often sufficient and saves much time.
- NoClone: Do not clone TilePlus tiles within the Bundle. The TPT tiles will clone themselves.
- MarkZoneRegAsImmortal: Used with the Chunking/Layout system. See Immortalizer tile
- Chunkified: The TileFab and its bundles are already Chunkified: See Chunkifying

Most of these are easy to understand and the Normal value for fabOrBundleLoadFlags is usually a good choice. You use the offset to set the origin for placement.

What happens is different depending on whether or not the TileFab is a square 'Chunk' as described earlier:

- Chunk: the 'offset' or placement position is the lower-left corner of where the TileFab's Bundles are placed.
- Not Chunk: the 'offset' is the lower-left corner of the bounds of the entire Archive.

This is automatic, and is mostly handled internally. You can see what happens differently for the two varieties with Painter: the tiles in a TileFab or Bundle assets are placed using LoadTileFab. The offset value is the mouse position converted into Tilemap (Grid) positions. You can see how that's done by examining the code in TpPainterSceneView.

When you paint a Chunk the lower-left corner of the Chunk will start at the mouse position.

When you paint a TileFab that *isn't* a Chunk (not captured with a Grid Selection) then the entire Tilemap had been archived. The mouse pointer position is the lower-left corner of the bounds of the Tilemap.

It sounds confusing but you can try this out for yourself to see the difference. In any case: if you do not want to load an entire archived set of Tiles you use Chunks. In practice they're way more useful.

TileParent is null unless this is being called from within a TPT tile, such as the TpAnimZoneLoader. It’s used when trying to find the painting Tilemap, as mentioned in the preceding section.

ZoneManagerInstance can be left null (the default) if you’re not using it. If not null, the results of the loading operation are archived in the TpZoneManager instance that you provided.

NewGuids: A new GUID is created for each TilePlus Tile in the Bundle. When the Bundle is created, the GUIDs of the TilePlus tiles are saved in the Bundle, and when the Bundle is painted by Tile+Painter or via code, the GUID of the painted TilePlus tiles are the same as when the tile was archived. But this isn’t always desirable. For example, if you wanted to use a TileFab repeatedly in a game you’d end up with the same GUID for multiple tiles.

That’s an issue because the GUIDs are supposed to be unique: the TilePlus system rejects TilePlus tiles with duplicate GUIDs and an error message is issued. If NewGuids is false, GUIDs are unchanged.

Filtering

What’s the filter for? There are several uses for this feature: limiting the number of tiles loaded to those within a certain area, extracting information from the loaded tiles, eliminating certain tiles, changing the Color or transform for a tile, or for replacing a tile with different one (like for seasonal events). You can also adjust the position of a Prefab.

As input, the filter is provided with:

- An object containing information about the Unity tile, TilePlus Tile, or the prefab asset with position information.
- A value from the FabOrBundleFilterType enum. This tells you what the object is.
- The BoundInt for the Bundle.

Note that for Prefab assets or Unity tiles: these are the actual project assets so don’t modify the asset. Changing to a different tile (not TPT) asset can be OK if it contextually makes sense.

When the filter receives a TPT tile it’s a clone so you can change its fields if you want to. Each of these objects has different types of information:

- Unity tiles: TileSetItem: a tile reference (as TileBase), position, color, transform, and tile flags Again, note that the tiles are assets.
- TilePlus tile: the TPT Tile reference and the position.
- Prefab: the prefab asset reference and the placement position.

If you want to change values for the actual objects such as the Unity Tile or the TPT tile, please be aware that:

- Don’t change from a TilePlus tile to a non-TilePlus tile or vice versa.
- Prefabs and Unity tiles are project assets and should not be altered.
- TilePlus tiles should be clones. If they are not, they auto-clone when painted on the Tilemap, which depends on the settings of TpLib.MaxNumClonesPerUpdate. To clone a TilePlus tile, call its Cloner method with the newGuid parameter = true.
- Since there are so many possibilities, this sort of use is unsupported aside from (our) bugs.

Loading from a List

Another way to load TileFabs is by providing a list of loading parameters and using `LoadTileFabsAsync`. The Layout system uses this approach. This method allows specifying the delay between loading TileFabs and a delay between loading each Bundle in each individual TileFab.

The delays allow you to spread out the loading over time. Since the loading happens outside the camera view this can be advantageous.

Revision #9
Created 2025-07-08 11:59:29 UTC by Vonchor
Updated 2025-08-13 16:36:15 UTC by Vonchor