TpLib

TpLib itself is a large class divided into eight partial classes:

- TpLib
- TpLibData
- TpLibDataAccess
- TpLibPools
- TpLibScene
- TpLibTasks
- TpLibTiming

An Editor folder has the final partial class:

- TpLibEditorUtils

If you're coding to the TpLib API, the parts you'd most likely be interested in are TpLibDataAccess, TpLibTasks, and TpLibTiming. Complete information can be found in the API reference (a zip file in the TilePlusExtras folder and here).

In this documentation, the dataset maintained in TpLib is generically called "Tilemap DataBase" or TMDB. It's not a database, but there's a set of query operations available which are tailored for use with TilePlus tiles and Unity Tilemaps. Data are added and removed from the TMDB automatically using features in the TilePlus tiles and the Unity Tilemap component.

TpLibDataAccess

The methods in TpLibDataAccess have pre-built 'queries' that allow you to extract information from the TMDB, which are data loaded into various structures in the TpLibData section of TpLib such as Types, Interfaces, Tags, GUIDs, etc.

There is also functionality for complex operations:

- Cut And Paste: Move a TPT tile from one position to another.
- Copy And Paste: Copy a TPT tile and place the copy elsewhere.
  - This should always be used for this sort of operation so that the cloned tiles are copied correctly.

Queries

Please consult the API reference for complete information. Not every variation is shown below.

- GetAllTilesInRegionForMap: load all TPT tiles on a specified Tilemap and within a RectInt region into a provided List.
- GetAllTiles<T>: load all TPT tiles of Type T in all Tilemaps into a provided List, with filtering callback.
- GetAllTilesOfType: load all TPT tiles of a particular Type from a specified Tilemap into a provided List, with filtering callback and a RectInt region. If the specified Tilemap is null, uses all Tilemaps.
- GetAllTilesWithInterface<T>: load all TPT tiles from a specified Tilemap into a provided List, with filtering. If the Tilemap is null, uses all Tilemaps.
  - An overload uses a provided list of Type T (saves casting later) and queries all Tilemaps. This includes a filter and a RectInt region.
- GetTilesWithTag: Get all tiles on a specified Tilemap with a particular tag into a provided List with filtering and a RectInt region. If the specified Tilemap is null then all Tilemaps are used.
- GetFirstTileWithTag: Convenience method, returns the first tile found from GetTilesWithTag.
- GetTilePlusBaseFromGuid: Find a TPT tile on any Tilemap that has a specified GUID.
  - Overloads allow use of a GUID string or byte array.
- GetTilePlusBaseOfTypeFromGuid<T>: Similar to GetTilePlusBaseFromGuid but returns null if the tile is not of Type T.

The methods that don't take a Tilemap reference or allow the reference to be null provide a Tilemap- and position-independent way to locate tiles without having any idea where they are actually located.

This is extremely useful!

When used with the TileFab loading and the Layout systems you can easily locate TPT tiles as they're loaded and you won't get null ref errors after the tiles are unloaded.

When used with the Messaging Service, you can send messages to tiles based on their Type, Interface, tags, etc., without having to prebuild a list of targets. It happens auto-magically.

GUIDs

GUIDs can be used as a persistent identifier for a specific tile. That's all they are used for.

But they're incredibly useful, especially for saving and restoring data from and to TPT tile instances. See Persistence.

Since you can retrieve any TilePlus tile by searching TpLib using its GUID, it's a truly Tilemap-independent means of locating a tile.

TpLibTiming and TpLibTasks

TpLib's core is built around static classes and Scriptable Objects, neither of which have any access to what we're all used to in a Monobehaviour update.

TilePlus Services use a class derived from ScriptableObject called ScriptableRuntimeServices. There's a chapter of this book devoted to them but briefly, Services such as the Tweener and the PositionDatabase require an Update method.

TpLibTiming

The classic solution to having an Update in a non-Monobehaviour class like a static class or a Scriptable Object is to have a dont-destroy-on-load GameObject spawned that vectors its Update method somewhere else. TilePlus used to do this.

But with the advent of the various domain-reload options in the editor, its not that easy to make that work flawlessly for this sort of an extension.

TPT version 5 uses one of two approaches.

- A modified Player Loop which updates at `PostLateUpdate`.
- An Awaitables-based Loop which updates at `EndOfFrame`.

If you're curious, check it out.

That update 'tick' is used in many ways, and that brings us to TpLibTasks.

TpLibTasks

TpLibTiming invokes TpLibUpdate in the TpLibTasks partial class. Here's a brief description of what happens in that method.

- If a target frame rate has been set the frame rate is calculated and maxima and minima are also calculated.
- TpServiceManager's Update is invoked.
  - The Service Manager sends an Update to all SRS that need it. This can change dynamically.
- Delayed Callbacks are evaluated (see next section).
- An internal cloning queue is examined and tiles waiting to be cloned are actually cloned at this time.

    - This handles cases where TPT tiles are 'woken up' by the Tilemap before TpLib is ready to register them; these requests are cached until the proper time. This is basically an edge case.

Delayed Callbacks

As a simple example: a tile wants to delete itself when its StartUp method is invoked (this is a real case). If you do something like:

`Tilemap.SetTile(position,null)`

from within that StartUp method Unity usually crashes.

In Editor windows, doing certain types of things during a GUI event cause GuiClip and other errors.

So lots of times you want to just wait till the end of the frame to perform these actions.

Or maybe a TPT tile wants to spawn a prefab 1 second after being sent a Message.

One way to do this is by using Awaitables and async methods. But that's actually way more complex than needed.

The DelayedCallback method is simple to use. It works within the TpLibUpdate method mentioned above to process these callbacks.

You can also provide a 'Condition' Func to test a condition, so if that condition isn't fulfilled by the end of the specified delay the delayed callback isn't actually invoked until the condition is met.

This method is used over 80 times in TPT (including the demos). It's very useful, and being removed from the scene hierarchy it isn't affected by scene loading or unloading. Null-checking is used throughout so if a TPT tile, an Editor Window, or some other caller which could possibly become null *does* become null is deleted prior to the timeout then the callback is not executed.

Repeated Invokes

The convenience method `InvokeRepeatingUntil` sets up a automatically-repeating callback, essentially, a timer.

It's essentially DelayedCallback with an empty `Callback` method. The parameter `invokedFunc` is the same as the `Condition` for DelayedCallback. The `invokedFunc` is a func taking a float [Time.deltaTime] and returning a bool.

Like DelayedCallback, the ID of the process is returned so you can terminate it if you want to. But it can be done by the repeatedly-invoked Func.

This Func is executed at a rate set with the `repeatInterval` parameter. If the Func returns true the timer continues running. When the Func returns false the timer terminates. In other words, you don't have to explicitly terminate the timer via its ID.

Unlike DelayedCallback, the `parent` parameter can't be null.

An example of this sort of use can be seen in the `CloudSpawnerTile`, which is part of the Topdown Layout demo.

Revision #26
Created 22 June 2025 17:00:02 by Vonchor
Updated 24 July 2025 16:17:00 by Vonchor