What is this ... THING

User Guide Introduction

Features

TilePlus Toolkit (TPT) is a unique way to work with Unity Tilemaps. It’s a Unity extension that can change the way you think about Tilemaps and how you use them.

REQUIRES UNITY 6 OR NEWER

Main Features:

- New Tile class which allows private instance data on a per-tile basis.
- New Brush for the Unity Tilemap Editor which supports editing these tiles’ data.
- New Brushless Painting/Editing tool: Tile+Painter.

Other Capabilities:

- High-level ‘Tile Scene’ subsystem.
- Archiving of single or multiple Tilemaps for fast loading and chunking.
- Fine-grained tile animation control including rewinding, looping and ping-pong looping.
- Tiles can control the animator component of a spawned prefab.
- Runtime `Service Manager` for Scriptable Object plugins
  - You can easily add your own.
- Bundled Services
  - Customized Tweener for tile sprites, including limited GameObject support.
    - Tween the tile sprite’s transform position, rotation, scale, matrix, or color.
    - Tween GameObject position, rotation, scale, or color
    - Tween GameObject position along a Bezier curve.
    - Sequences are supported
    - Convert Tweens and Sequences into Awaitables.
    - Custom tweens: tween anything.
  - Pooled Prefab and Tile spawner.
  - Position Database: collider alternative.
  - Messaging Service:
    - Tiles can message other tiles or send events to Monobehaviours.
    - Monobehaviours or static classes can message tiles.
- Simple save/restore systems for tiles’ data.
- Built-in Zone creation for setting trigger zones.
- Layout System for top-down or side-scroll orthographic views.
- Internal late-update-based scheduler you can use for timers inside tile code or elsewhere.
- Assortment of utility methods for Tilemaps.
- Several pre-created Tiles for common uses
- Use Tiles as UI.
  - Animated and static buttons
  - Ascii characters and strings (no editing)

- Hover zones for tooltips.
- Radio buttons
- Toggle buttons

And importantly, there is no interference with your existing project. No special dependencies, no special GameObject tags, no changes in how Tilemaps work: just a lot of C# code – and the source is included.

Where to Get Started

If you’re not into coding and want to play with some feature demos, head over to the TilePlus Extras folder. Each demo has its own documentation in text or markdown format.

The demo program doc files can also be found here.

If you want to use Tile+Painter: it has a short “Quick Guide” in the TilePlusExtras/Documentation folder. Clicking the ‘?’ button in Painter displays a quick summary right in same window.

All of the other documentation is online. The menu item ‘Tools/TilePlus/Docs’ opens a browser at the documentation website.

New Tiles!

New Tile Class?

The key component of TPT is a new Tile base class cleverly dubbed “TilePlusBase” (TPB). This tile clones itself when placed on a Tilemap in a scene.

Why would anyone care?

The One With No Instance Data

One of the issues that developers run into with Unity Tilemaps is that there’s no way to add fields (variables) to tiles and have the data serialized and saved in the scene just like the serialized fields of scripts used for components. This is because the Tilemap’s serialization is hard-wired to save data from the basic fields present in a Tile class.

For many (including your TPT developer) this is annoying, to say the least. Perhaps you want a configurable waypoint. Maybe you want to be able to paint a tile and set it up as a spawn zone. And you want to be able to edit fields as you usually do.

Using TilePlusBase and the supporting libraries you can have any sort of code and/or data in a tile. Since the tile is cloned, it is no longer connected to the asset in the project folder: it exists in the Scene, and its data is saved with the scene.

If you’ve used the Unity Tilemap Editor (UTE) you’ve used its Selection Inspector. That inspector is very different from the normal Unity inspector panel: the UTE Selection Inspector is hard-wired to support the fields of the Tile-class tile and that’s it.

The support libraries for TPT have an alternative Selection Inspector that’s available in the UTE as the “Tile+Brush” or by using the TPT’s Tilemap painting and editing tool: Tile+Painter (T+P).

Decorate your TPB-derived tiles with TPT’s custom attributes and this alternative Selection Inspector lets you view and edit those fields or display property values. See the Online Documentation for more information.

Basic Architecture

Libraries

TPT libraries are divided into Editor support and Runtime support, with separate assembly definitions.

The Editor Library

This includes support for the custom Selection Inspector, the Tile+Brush, several custom editor windows, diagnostic tools, and Tile+Painter. Painter is a Painting / Editing tool with separate documentation for you to read. It’s UI-Elements based and does away with the concept of brushes completely.

The Runtime library

The most important part of the Runtime library is generically called TpLib and comprises several different subsystems. Much of what you use it for can be loosely thought of as a “Tilemap Database” or TMDB.

By using the TMDB you generally don’t have to keep track of what Tilemap a tile is placed on nor its position. For example, you can query TpLib for all TPT tiles with a particular tag and send them all a message to activate animation with ONE method call.

Keeping track of tiles in this fashion is much easier and becomes very important when chunks of tiles are dynamically added and removed from Tilemaps as part of TPT’s Layout system. For more details see the online documentation.

Example: a Waypoint. Your player reaches the waypoint. You preserve its GUID in a save file so you can easily locate it again without knowing its position or even which Tilemap it’s on.

Organization

TPT is divided into two types of code: Static libraries and Scriptable Runtime Services (SRS).

Static Libraries

TpLib: The main enabling library for this system.

TileFabLib: This library handles loading Tilemap archives. It's also the underlying basis of the Layout subsystem.

TpEvents: Tiles can post events. A MonoBehaviour or other code can interpret these, or scriptable objects called "Actions" can be used to automatically handle the events.

TpTileUtils: A library of utilities related to Tilemaps and tiles.

TpServiceManager: Managers Scriptable Runtime Services (see below).

SRS

SRS is short for "Scriptable Runtime Service." These are Scriptable Objects which can be dynamically loaded at runtime (only) and have special features such as being able to receive Update events.

TPT uses SRS instead of static classes when possible.

1. Faster Domain reloading.
2. If you don't need certain SRS then they aren't in memory.
3. SRS can be unloaded from memory when not needed.

One loads and destroys Services with the TpServiceManager.

These basic SRS are provided with TilePlus Toolkit:

- TpMessaging
- TpSpawner
- TpTweener
- TpTilePositionDb

You can easily create SRS for your own use, see the online documentation. The Layout demo uses custom SRS for game state, file access, and the customized layout controller for the Layout system.

Using a SRS for game state is something that one shouldn't normally do as it's using the service like a Singleton which is generally frowned upon. Although a SRS makes a great singleton, there's no 'Instance' property, the only access is via the service locator.

More Tiles!

You'll also find some easy-to-use TPT tiles:

TpAnimatedTile: similar to Unity's AnimatedTile, but with looping support including Ping-Pong.

FlexAnimatedTile: animated tile where you can choose what to animate from an asset that contains a list of sprite animations. Looping, Ping-Pong, Start/Stop animation and changing animation sequence at runtime all supported. Animation preview in-editor when not in Editor-Play mode.

AnimatedSpawner: derived from FlexAnimatedTile, spawns a prefab or a tile when triggered.

SlideShow: Single-step animation though a list of sprites.

The TilePlusBase base-class tile has the data and UI for demarking a rectangular region of tiles. This is used in 'Zone' tiles:

AnimatedZoneLoader: Load a Tilemap archive when triggered.

AnimatedZoneSpawner: Similar to AnimatedSpawner, but uses a Zone.

ZoneAnimator: Control Animation of a Unity prefab when triggered.

"Triggered" refers to an action that's taken because of an inter-tile or Monobehaviour-to-tile communication. This may sound weird but it's very powerful.

Tweening is supported with some example tiles:

- TpTweenerTile
- TpTweenSpecTile
- TpTweenSpecSequenceTile

Read about these in the separate Tweener documentation.

Aaaand just for fun, how about Tilemap UI:

- UiButtonTile
- UiAnimButtonTile
- UiAsciiCharTile
- UiAsciiStringTile
- UiRadioButtonTile
- UiToggleButtonTile

You can read about these in the separate TilemapUi documentation. It’s not a replacement for UiElements, IMGUI, or UnityUi.

What Can I Do With It?

If all you were interested in was Tile+Painter, you'll find it can do 90% of that the Unity Tilemap Editor (UTE) can do and many things that it can't or doesn't do. Read all about it in Tile+Painter User Guide.

Let's talk about the new types of tiles.

One of the big limitations of Unity Tiles is that they're basically templates and can't store any data other than what's already set up in the Tile class: generally, just Color, transform, Collider, GameObject, and so on. Not that there's anything wrong with that! But there's no possibility of instance data: data specifically related to just one tile. There's no Update method and no simple way to have timed functions like delays.

This means that actual functionality – doing something – can't be done in a tile, or at least it's more difficult than it needs to be.

TPT tiles work differently: but simply, TPT tiles are promoted from Project assets to Scene objects. When you save the Scene, the TPT tiles are saved along with everything else in the Scene, along with any data in the tile.

TPT is based on new Tile asset classes and a support library. Unlike normal tiles, modifying fields after painting does not alter the original tile asset. This opens new possibilities for working with tiles and requires no changes to the standard Tilemap system. No modifying Scene files; none of that. Just code.

The sample tiles included with TPT add much more flexible animation capabilities, tiles with trigger zones for spawning prefabs (built-in pooling) or tiles, tiles with trigger zones that work with custom coding to load new sections of your tilemap, and more.

These tiles can have references to GameObjects, components, asset files, and so on.

At a higher level, TPT can bundle portions of, or entire Tilemaps into custom Assets and Prefabs. These can be painted directly on Tilemaps, dynamically loaded at runtime, and much more.

Caveat

As you may already know, UnityEngine.Objects in Unity Scenes can't be referred to by a Prefab. So, if you drag one or more Tilemaps into a prefab then TPT tiles will, in technical terms, get all messed

up.

No worries though, there’s a Tools/TilePlus menu function that lets you create compatible Prefabs of single or multiple Tilemaps at once. However, saving your Tilemaps inside Scenes is the most flexible workflow and “just works.”

Even if you want to use load one or more Tilemaps as Prefabs, the flexibility of these new Tile types can be extremely useful, even with the workflow changes. But it’s possible to re-think the organization of a project and not use Tilemap prefabs at all.

You might also explore the TileFab archiving system: save Tilemaps in an asset and load whenever and wherever you want.

Check out the Workflow section for more information.

The FAQs section at the end of this document may have answers to some of your questions.

Required Unity Version

Unity6 or newer is required. Earlier versions are no longer supported.

Unboxing, Setup, and Reinstalling

What's in the Box?

Installing the TilePlus Toolkit UnityPackage will add a few folders to your Assets folder:

- Assets/Gizmos/TilePlus
- Assets/Plugins/TilePlus
- Assets/TilePlus Extras

The Assets/Gizmos/TilePlus folder should never be removed unless you're deleting the entire TPT package: the image files within are used to provide custom icons for tiles and scripts.

Assets/Plugins/TilePlus contains Editor and Runtime code, Tile+Painter, a Chunking subsystem, a new Palette Brush called the "Tile+Brush," and several new Tiles that you can use or add features to if you're into Unity programming. This folder should not be moved.

The Assets/TilePlus Extras folder is optional, and you can remove it with no effect. Here you'll find documentation and demo projects. Demo projects are discussed in the Online Documentation. Documentation includes an API reference in zipped-website format.

Setup

Dependencies: *2D Tilemap Editor* and *2D Tilemap Extras* are required, the latter for Rule tile support.

- *Splines* is required for one of the demo programs.

Assemblies: There are two assemblies – *TilePlus.Editor* and *TilePlus*, neither of which should be deleted or modified.

Palette Brush: After installation, the *Tile+Brush* replaces the default Brush that comes with the 2D Tilemap Editor package.

If you want to keep the default brush available in the Palette window, you need to make a minor change to a file. Unfortunately, there's no other way to do this.

In the Project window, navigate to Plugins/TilePlus/Editor/Brush/TilePlusBrush.cs and open that file in your code editor. Right near the beginning of the file, just above the [CustomGridBrush] attribute, there are instructions on how to do this. It's easy, just comment out one line and uncomment the other.

The Tile+Brush remains available; it just won't be the default anymore.

Important: The Plugins/TilePlus folder should not be relocated.

Compiler #defines

The TpLog and TpLogWarning methods don't emit messages to the console in a build unless the Player Settings have TPT_DEBUG added as a scripting define symbol.

Reinstalling

It's best to totally delete the TilePlus folder prior to upgrading to a new release.

Please note that if you've changed any of the shortcuts (see Menu Items, below) then the original shortcuts may be reapplied, and you might need to manage any resulting conflicts in the Shortcuts manager.

Builds

See Preparing For Builds