Create Your Own

Create new TilePlus tiles

Basics

In most cases, the class that you want to subclass is TilePlusBase. But you might want to extend from one of the supplied tiles like TpFlexAnimatedTile. Many TilePlus Tiles' fields, properties, and methods are marked as 'virtual' so they can easily be overridden.

When creating subclasses of TilePlus tiles you should specify a namespace. Use Tools/TilePlus/Configuration Editor and add your namespace to the Namespaces field. Then click the Reload button. Namespaces are required for derived TilePlus classes. If the namespace isn't added to the system via the Configuration Editor, then the Selection and Brush inspectors will not display any fields or properties that you've decorated with TilePlus attributes.

Many of the properties in the sections demarked by #if UNITY_EDITOR are things that you can ignore. If you want fields or properties to display in the Tile+Brush Inspector, you can use attributes to display them. The TilePlusBase property "Description" can be used to show some text information about your tile in the Tile+Brush inspector.

It's helpful to examine the ITilePlus interface as it's not cluttered up with code, tooltips, attributes, conditional compilation directives, and so on.

Properties

Why so many properties?

Fewer serialized fields for things that are just basically boolean switches used by various parts of the system. For subclasses that don’t implement a particular functionality, specific fields aren’t needed, just a constant value. For those that do, serialized and non-serialized fields allow data to be provided via the properties, or a return value is computed for the property.

The TilePlusBase class implements all the items in the ITilePlus interface except those having to do with simulation: that has default values provided by the properties in the interface (C# 8 feature).

Most of what’s in ITilePlus are used internally and it’s unlikely that you’ll use them at all. But there are three which are especially useful: ParentTilemap, TileGridPosition, and TileWorldPosition.

- ParentTilemap always has a reference to the tilemap for the tile. This is useful for a lot of things, but beware: if a tile tries to erase itself by using the ParentTilemap reference to place a null tile at the TileGridPosition, Unity will crash.
- TileGridPosition always has the location of the tile in Grid coordinates.
- TileWorldPosition always has the location of the tile in World coordinates.

Methods

Simulate can be implemented to use the Editor update event to do something. In TpAnimatedTile and TpFlexAnimatedTile it’s used to show an animation preview. It's unlikely that you'd need to implement this yourself if you inherit from classes that already implement this feature.

TilePlusBase has two other methods that you probably want to override:

- StartUp
- GetTileData
- ResetState.

Overriding StartUp must be done a specific way: a simple example can be found in TpAnimatedTile. Basically, you must call the base method as usual, but pay attention to the return value: if it’s false your override must return false without doing anything else:

```
//this has to be the first thing to do.
if (!base.StartUp(position, tilemap, gameObject))
return false;
```

It would be an extremely bad idea to change any code in the Startup method in TilePlusBase. Worse than being eaten by the Sarlacc on Tatooine or looking into the atomic furnaces of the Forbidden Planet, Altair IV.

GetTileData is probably the most complicated override, aside from being sure to call the base class or duplicating its code, examine some of the examples for guidance. But it’s obviously extremely specific to what you’re trying to do. Be sure to use the tilemapIsPalette field to exit the method after the base call if tilemapIsPalette is true. Otherwise, your tile might animate in the Palette window. GetTileAnimationData code should also test tilemapIsPalette. See TpFlexAnimatedTile for examples.

ResetState is used in-editor when using the Bundle Tilemaps menu command or the TilePlus Bundler command in the hierarchy window’s context menu. It’s also used when you pick and re-paint a clone TPP tile or use TpLib.CopyAndPasteTile. The implementation must reset fields so that stale data isn’t persisted. Be sure to call the base method. This is a misleadingly simple method that you need to think about carefully. You don’t want to reset all fields, or you’ll be undoing changes made to your TPT tile. See the various implementations for examples.

Namespaces and Interfaces

Namespaces

The GUI formatter for the Brush and Selection inspectors displays information in class-hierarchical order. But it needs to know what not to display, otherwise it will breeze through the class hierarchy all the way to UnityEngine.Object.

Therefore, by default it ignores anything outside of specific namespaces. The TilePlus namespace is hard-coded in.

The configuration editor has a Namespaces text field where you can provide a comma-delimited list of namespaces to use. The default for that text field includes TilePlusDemo.

When creating your own tile classes, place the namespace that you're using in this list. Don't forget commas! Note that if you add a namespace, attributes are still required to display information.

For example, if you were to add the UnityEngine.Tilemaps.Tile namespace then the TilePlusBase' base class of Tile would not appear in a foldout.

Please click the Reload button in the configuration editor when you change this. Also, be aware that if you click "Reset To Defaults" that you'll need to re-add the namespaces!

Interfaces

ITilePlus specifies several properties and a few methods that are common to all tiles subclassed from TilePlusBase since that class implements everything in the interface.

Please note that any subclasses of TilePlusBase using ITilePlus properties with default members need to specify the interface to 'override' the defaults. This can be seen in the tiles which support simulation (TpSlideShow, TpAnimatedTile, and TpFlexAnimatedTile).

ITpPersistence specifies properties required for tiles using TpLib's save/restore framework.

ITpMessaging specifies properties required for tiles using TpLib's messaging framework.

ITpSpawnUtilClient specifies properties required for tiles which spawn prefabs or paint tiles when using the SpawningUtil library methods.

ITpMessaging and ITpPersistence are the interfaces you'll most likely implement if creating your own tiles and you want to use TpLib's messaging and save/restore frameworks. If you don't want to

use those then you can ignore those interfaces.