

# Table of Contents

[Intro](#)

[Articles](#)

[Intro](#)

[Tutorials](#)

[Getting Started](#)

[Big Tiles](#)

[3D Layouts](#)

[Path Constraints](#)

[Generating on Mesh Surfaces](#)

[Multiple Passes](#)

[Settings](#)

[Generator Settings](#)

[Tile Painting Controls](#)

[Mouse and Keyboard Shortcuts](#)

[Constraints](#)

[Index](#)

[Scene Constraints](#)

[Generator Constraints](#)

[Count Constraint](#)

[Mirror Constraint](#)

[Path Constraint](#)

[Separation Constraint](#)

[Border Constraint](#)

[Extras](#)

[Animation](#)

[Infinite Generator](#)

[Bolt Support](#)

[Paint Tiles From Mesh](#)

[Samples](#)

[Palettes](#)

[Cell Types](#)

[Models](#)

[Adjacent Paint Model](#)

[Overlapping Model](#)  
[Adjacent Model](#)  
[Mesh Edges Model](#)  
[Controlling Output](#)  
[Instantiate Output](#)  
[Mesh Output](#)  
[Tilemap Output](#)  
[Quality, Performance and Debugging](#)  
[FAQ](#)  
[Using the API](#)  
[Developing](#)  
[Upgrading to Tessera Pro](#)  
[Release Notes](#)  
[Api Documentation](#)

## Tessera

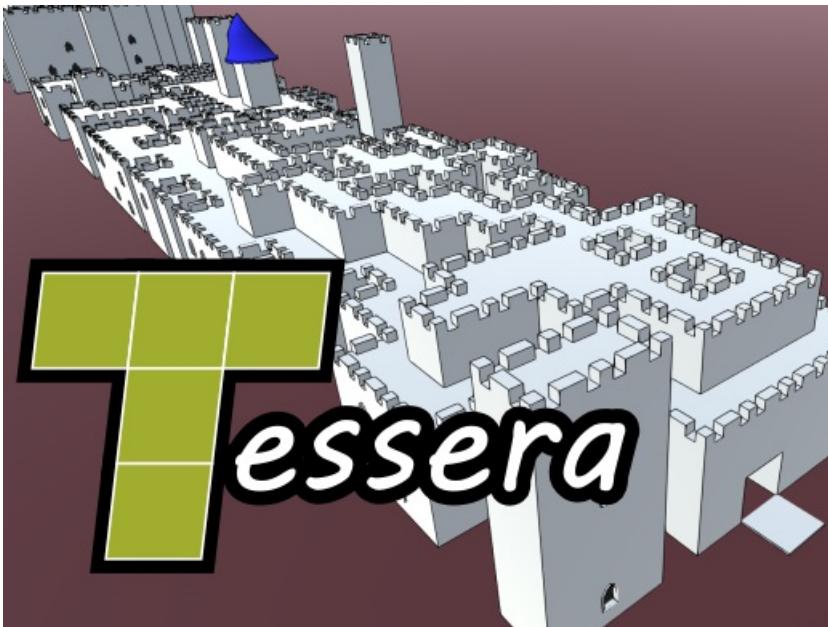
[AnimatedGenerator](#)  
[AnimatedGenerator.AnimatedGeneratorState](#)  
[BiMap<U, V>](#)  
[BorderConstraint](#)  
[BorderItem](#)  
[CellFaceDir](#)  
[CellRotation](#)  
[ChunkCleanupType](#)  
[CountConstraint](#)  
[CubeDirExtensions](#)  
[CubeFaceDir](#)  
[CubeFaceDirExtensions](#)  
[CubeRotation](#)  
[EdgeDetection](#)  
[EdgeDetection.Face](#)  
[EnumeratorWithResult<T>](#)  
[FaceDetails](#)  
[FaceType](#)  
[FailureMode](#)  
[HexGeometryUtils](#)  
[HexPrismDirExtensions](#)  
[HexPrismFaceDir](#)

HexPrismFaceDirExtensions  
HexRotation  
IEngineInterface  
ISubFaceInfo  
ITesseraInitialConstraint  
ITesseraTileOutput  
InfiniteGenerator  
InstantiateOutput  
MeshUtils  
MirrorConstraint  
MirrorConstraint.Axis  
ModelTile  
ModelType  
OrientedFace  
PaletteEntry  
PathConstraint  
PinType  
PrefixLookup<T>  
RotationGroupType  
SeparationConstraint  
SquareFaceDir  
SquareFaceDirExtensions  
SquareFaceExtensions  
SquareRotation  
SubFaceTuple  
SubFaceType  
SylvesConversions  
SylvesExtensions  
SylvesHexPrismCorner  
SylvesHexPrismDir  
SylvesOrientedFace  
SylvesTrianglePrismCorner  
SylvesTrianglePrismDir  
TRS  
TesseraBaseSceneConstraintComponent  
TesseraCompletion  
TesseraConstraint

TesseraGenerateOptions  
TesseraGenerator  
TesseraHexTile  
TesseraInitialConstraint  
TesseraInitialConstraintBuilder  
TesseraInstantiateBehaviour  
TesseraInstantiateEvent  
TesseraInstantiateOutput  
TesseraInstantiateOutputTileMapping  
TesseraMeshOutput  
TesseraMeshOutputCollider  
TesseraMeshOutputMaterialGrouping  
TesseraMultipassGenerator  
TesseraMultipassPass  
TesseraMultipassPassType  
TesseraPalette  
TesseraPinConstraint  
TesseraPinned  
TesseraSquareTile  
TesseraTile  
TesseraTileBase  
TesseraTileInstance  
TesseraTilemapOutput  
TesseraTransformedTile  
TesseraTrianglePrismTile  
TesseraUncertainty  
TesseraVolume  
TesseraVolumeFilter  
TesseraWfcAlgorithm  
TileEntry  
TileList  
TileSelectionType  
TrianglePrismDirExtensions  
TrianglePrismFaceDir  
TrianglePrismFaceDirExtensions  
TrianglePrismGeometryUtils  
TriangleRotation

UnityEngineInterface

VolumeType



**Tessera** is a Unity addon for procedurally generating 3d tile-based levels and builds. [Get it here.](#)

---

Tessera works by running the [Wave Function Collapse](#) algorithm. This algorithm is a powerful technique for generating maps from complicated tile sets with tight controls on behaviour.

For help, use [Discord](#) or post on the [Unity forums](#).

To get started with Tessera 6.3.1, see the [introduction](#).

# Introduction

**Tessera** is a Unity addon for procedurally generating 3d tile-based levels and builds.

Tessera works by running the [Wave Function Collapse](#) algorithm. This algorithm is a powerful technique for generating maps from complicated tile sets with tight controls on behaviour.

Tessera is available on the Unity asset store as two different packages. The [basic version](#) includes the fundamentals of generations and key features, while the [Pro version](#) contains more advanced features. These tutorials cover both, with pro-only features clearly marked.

Tessera has a lot of features. Try the following links for tutorials and an overview of what is available.

- [Getting Started Written Tutorial](#)
- [Getting Started Video Tutorial](#)
- [List of Samples](#)
- [Autogenerated API Reference](#)

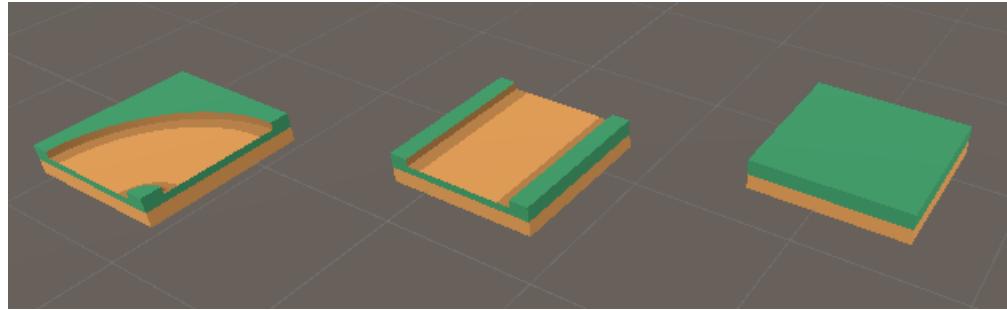
# Getting Started Tutorial

## Setup

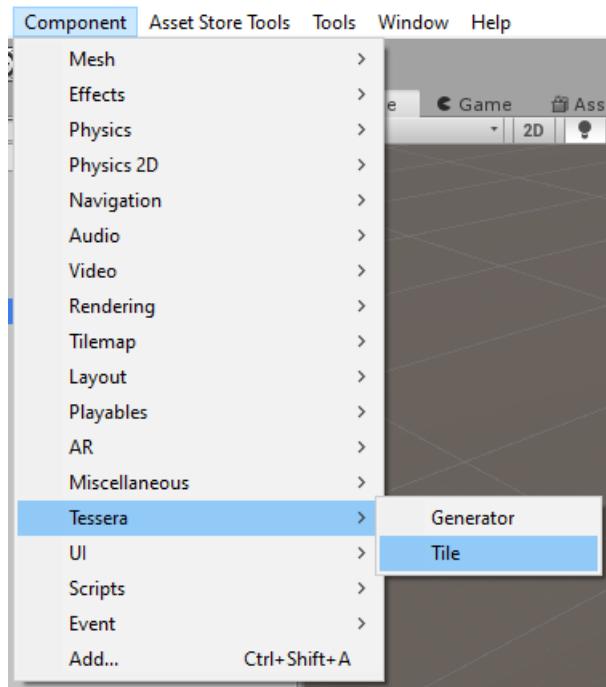
Start a new project. Then download [Tessera from the Unity Asset Store](#) and import it to your unity project. For this tutorial, we'll also use use [Kenney's Tower Defense Kit](#) so either download that or use the files included in `Samples/GrassPaths/Models`.

## Creating tiles

Lets create some tiles. From the tower defense assets, drag the prefabs for `tile`, `tile_cornerRound` and `tile_straight`. These tiles are a small selection of grass and path tiles.



Then, with those game objects selected, add the `TesseraTile` component from the menu.



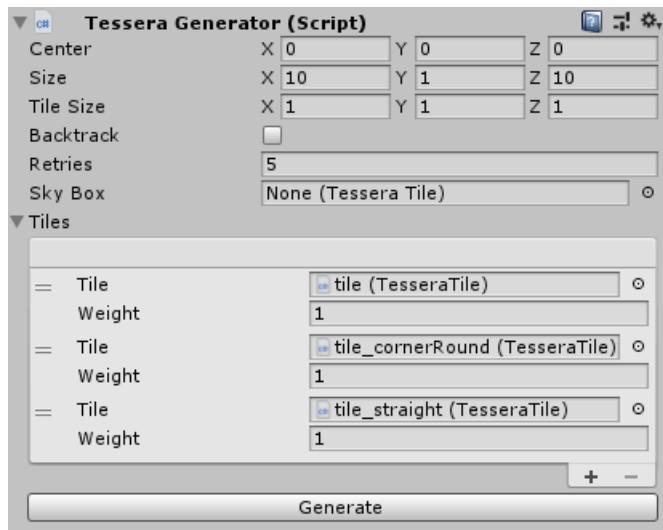
That's all for now, we'll configure the tiles later.

## Creating the generator

Next, create a new empty GameObject, and give it the `TesseraGenerator` component from the menu. Bring it up in the inspector. Add the tiles we created before to the list of tiles, either by dragging them from the hierarchy onto the Tiles section, or clicking the small plus button and selecting each tile.

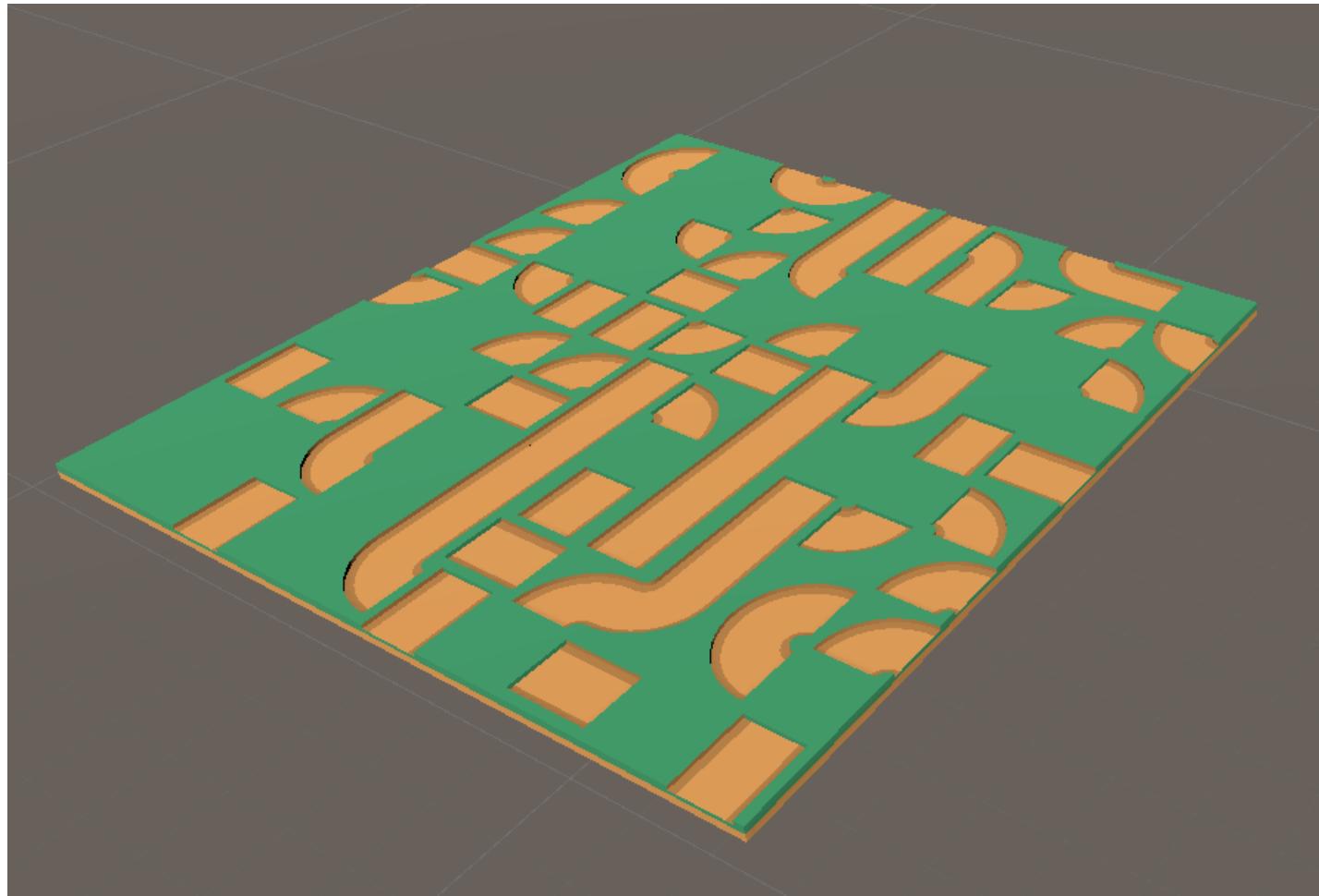
Position the generator so that it does not overlap the tiles you created.

Afterwards, your configuration should look like this:



There's more details on these settings [here](#).

Now press the "Generate" button to create a new arrangement of those three tiles. You should get something looking like this:



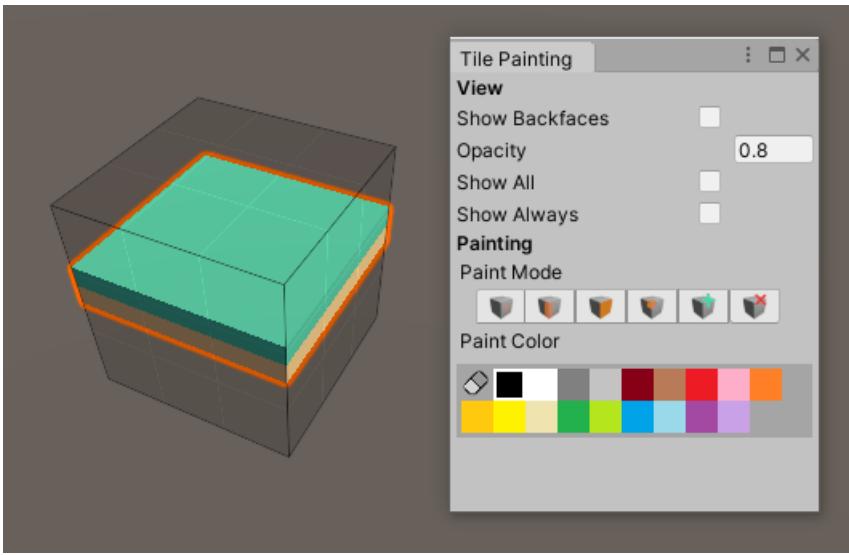
It's generated the tiles, but right now it doesn't know which tiles can be placed next to which other ones. So it has just placed them randomly. Hit undo to delete the created tiles.

To fix this, we need to paint the tiles.

## Tile painting

Select the first tile, called `tile`. In the inspector, click the "paint faces" button.

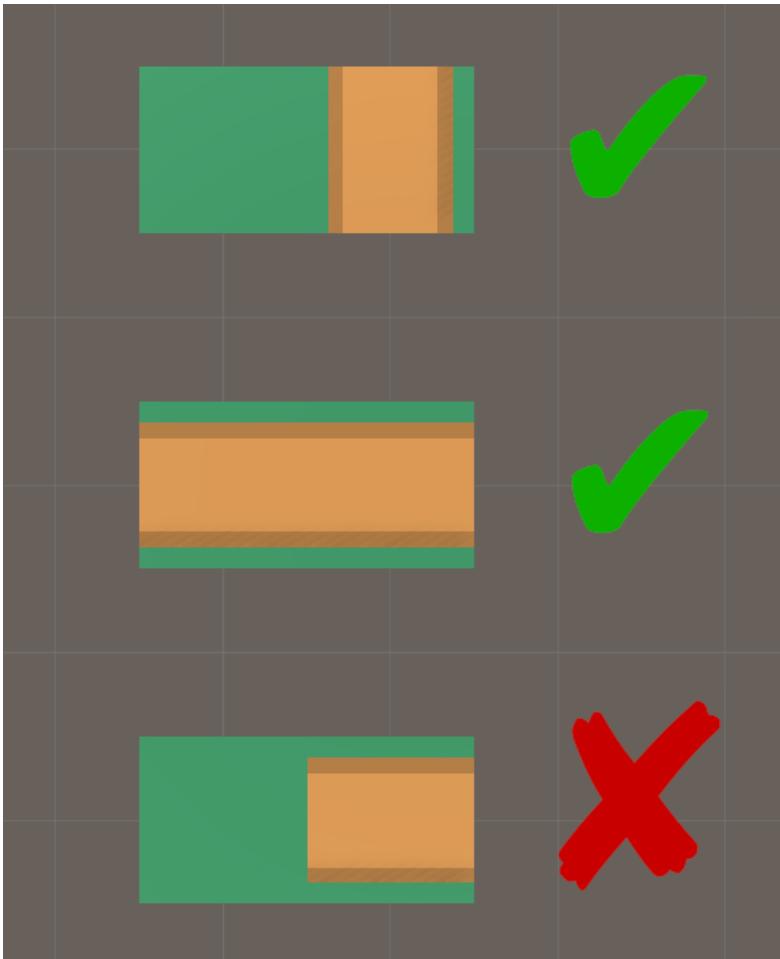
This should pop-up a "Tile Painting" window, and also show a semi-transparent cube around the tile.



You can use [these tools](#) to paint different colors onto the tile's cube. First, select a paint color from the palette, then click on the cube to apply that color. If you make a mistake, select the eraser from the palette and clear what has been painted.

**Tiles can only connect to each other if they have matching colors painted on their corresponding faces.** Specifically, each face is divided subdivided into 9 squares. A pair of adjacent tiles are compared by pairing up the squares on the opposing faces, and seeing if they match. Squares match if they are both the same color, or if either square is transparent, though this can be customized with a [Palette](#).

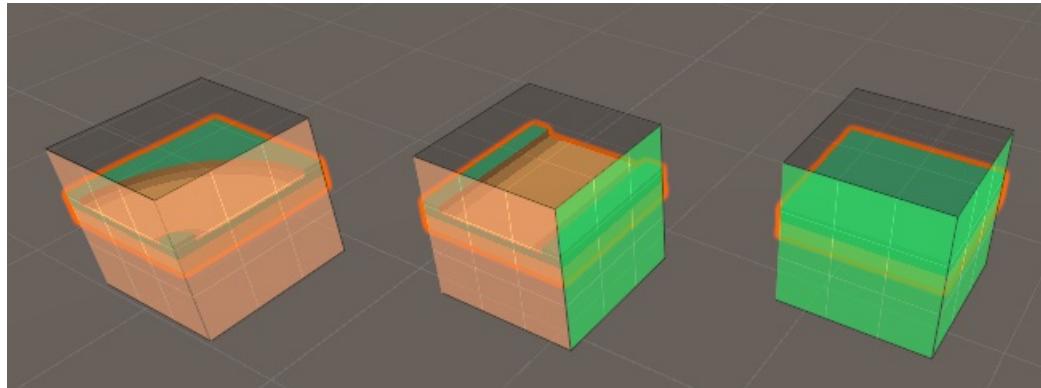
In this case, we want tiles to connect to each other if they are both grassy, or if they are both a path, but do not want paths to lead straight into grass.



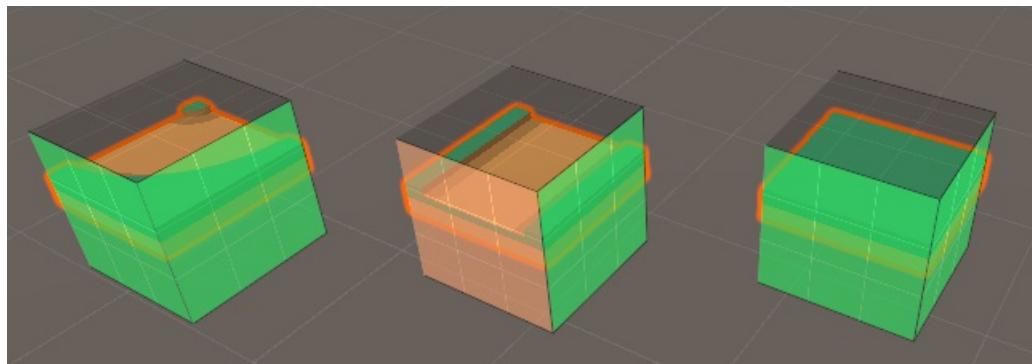
In order to achieve this, we will paint all the grassy faces of each tile green, and all the faces with paths brown. The top and bottom we will leave alone. That will connect grass to grass, paths to paths, and disallow grass connecting to paths.

Paint all 4 sides of the 3 tiles now. Afterwards, you should have three tiles that look like this.

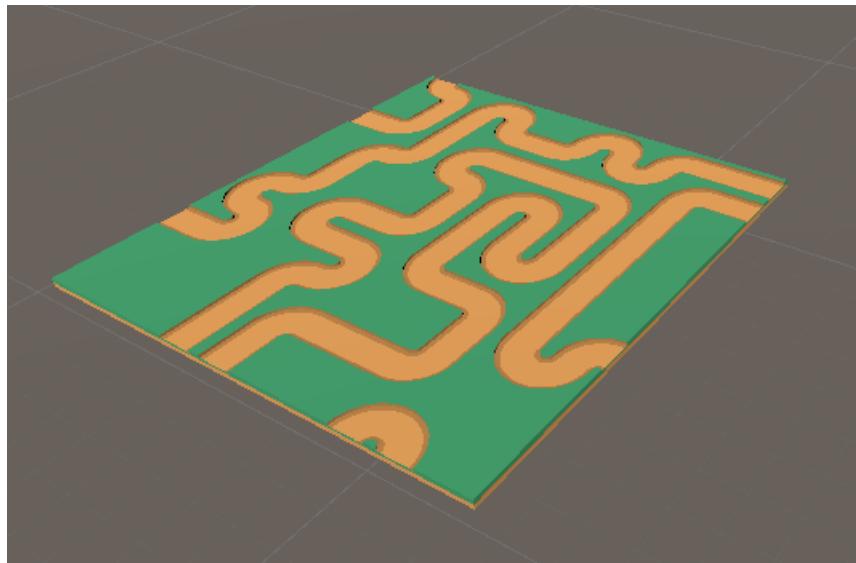
Front view:



Rotating each tile 180 degrees for a rear view:



Now we can go to the generator and press the "Generate" button again. Make sure you have deleted the tiles it created the first time around as it won't overwrite already generated tiles. If everything is set up correctly, it should look like this.



This concludes the tutorial. From here you can experiment with some of the settings in the inspector, try adding more tiles from the tower defense assets, or read the more advanced tutorials.

# Big Tiles Tutorial

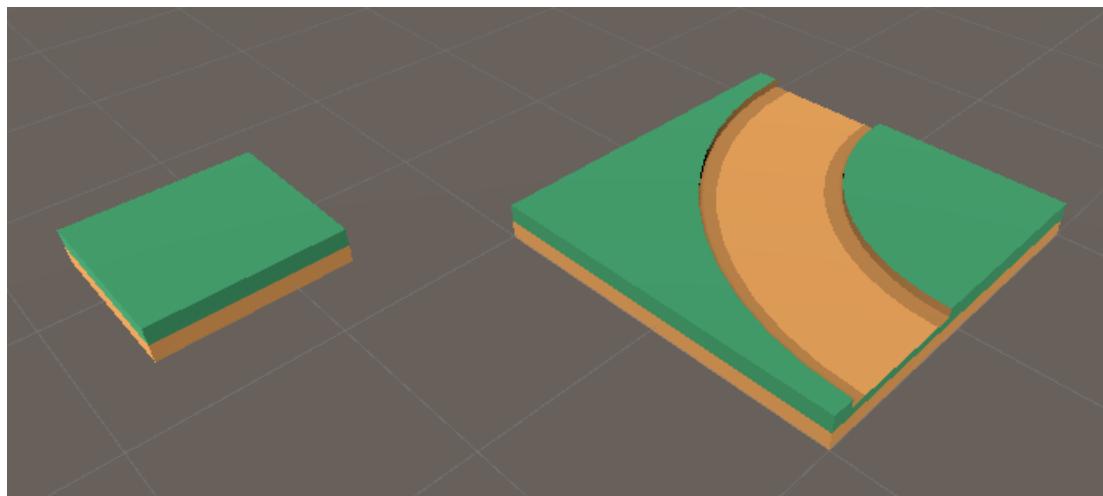
This tutorial continues from the [Getting Started](#) tutorial. It is recommended you complete that one before starting this.

So far we've looked at generating game objects that all have the same size. That is convenient, but the clear grid structure is not always desired. Here we look at one way of addressing that. If normal tiles only occupy one cell in the output, the big tiles can straddle several cells. That means you can design a larger set piece cohesively.

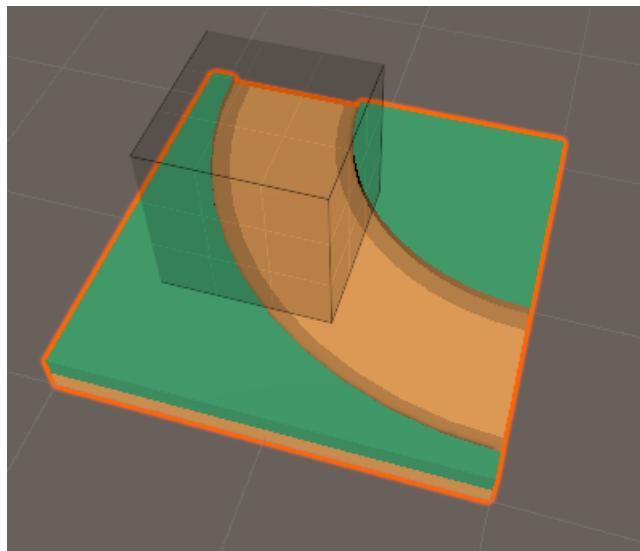
## NOTE

Big Tiles should only be used for tiles that are larger than your grid's tile size. If you are creating *every* tile using Big Tiles, you're better off just changing the grid.

Let's add a new tile from Kenney's Tower Defense Kit. This time, pick `tile_cornerLarge`. It is twice as bit as a regular tile in the X and Z dimensions, so it will occupy 4 cells in the final generation.



Lets set it up. As before, add the TesseraTile component. Then set the Center to (-0.5, 0, -0.5). This will place the paintable cube in one corner of the tile.

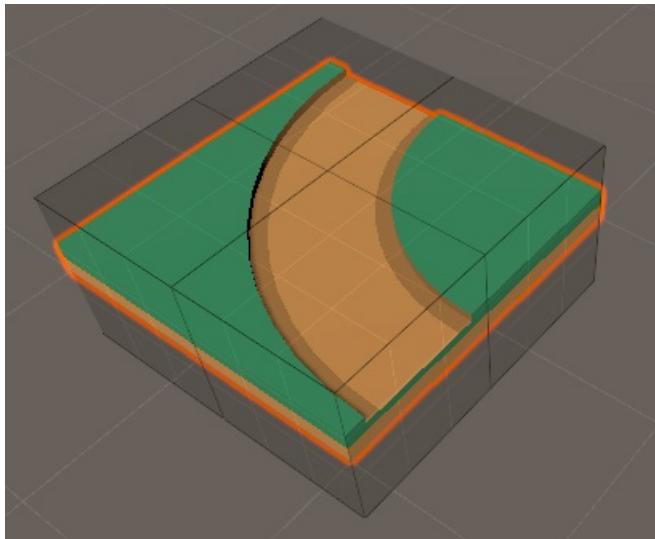


Then select the "Add Cell" tool from the paint menu . You can now click on faces of the paintable cube to make a tile with multiple cubes in it. If you make a mistake, you can use the "Remove Cell" tool to delete them.

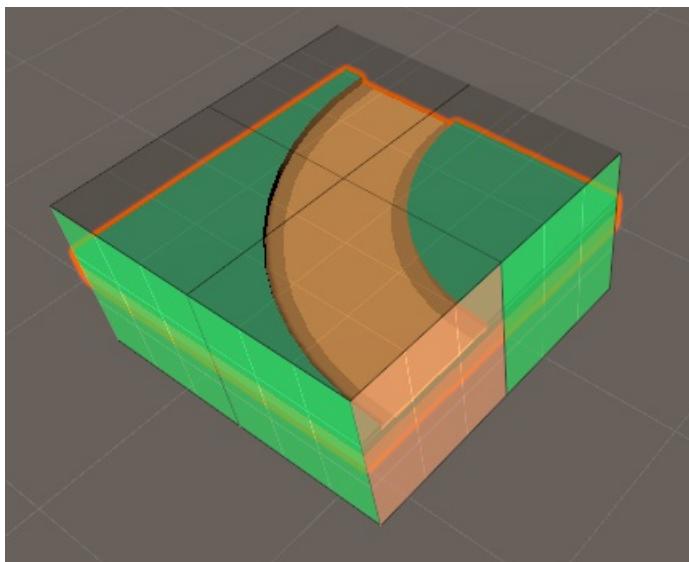
## NOTE

Big Tiles should have the same Cell Size as other tiles. You must use the Add Cell tool to make big tiles.

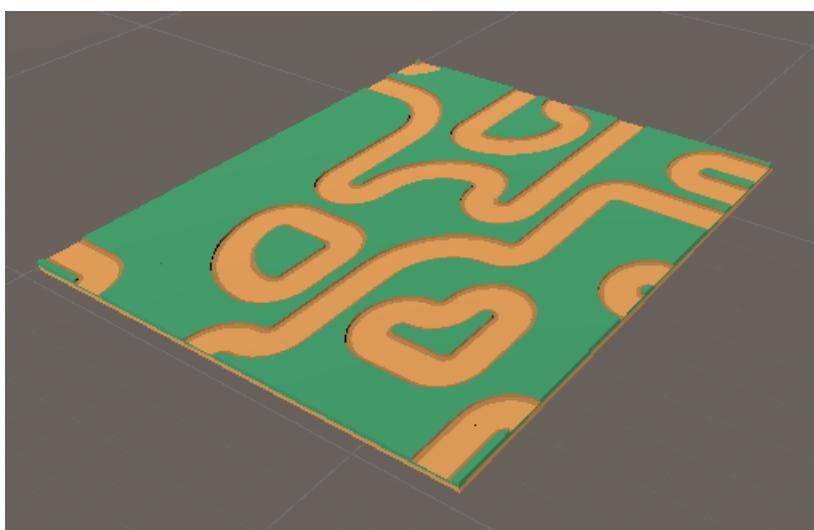
Add 3 extra cubes to the tile to cover it. This tile will now take up 4 cells worth of space, replacing 4 normal tiles.



Paint the sides of the new tile green and brown, like the original tiles, to indicate how it connects.



Now we're ready to add this cube to the list of tiles in the generator, and hit Generate. The new tile will be seamlessly mixed with the smaller tiles.



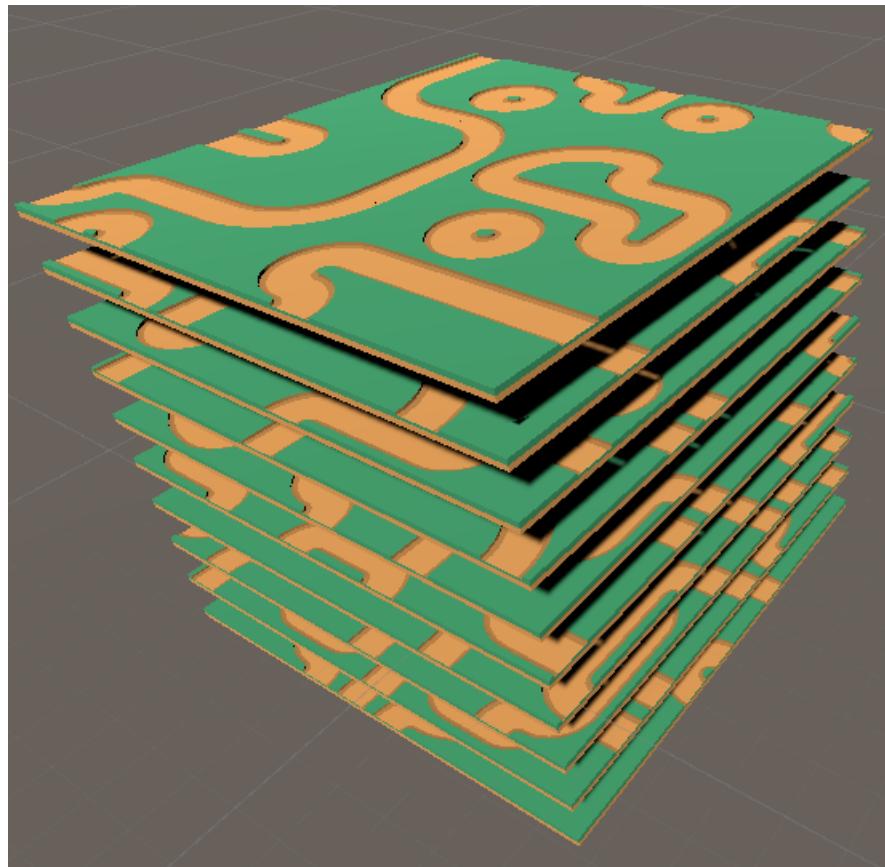
# Generating 3d layouts Tutorial

This tutorial continues from the [Big Tiles](#) tutorial. It is recommended you complete that one before starting this.

## Earth and air

So far we've only generated a single layer of tiles. But Tessera can work with 3d grids too. The principle is exactly the same - paint tiles to indicate how they connect and let Tessera do the rest.

Let's turn our previous example into a 3d example. First, go to the generator, and find the size setting and increase the Y size to 10. Now if we hit generate, we get something like the following:



There's two things to note:

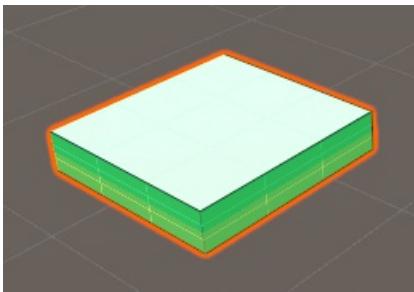
- Each layer is separated from the one above by 1 unit, even though our tiles aren't nearly that high.
- Just like in the first tutorial, Tessera doesn't know how things connect. We need to paint the tiles to indicate what can be put on top of what.

Let's fix those issues.

First, change the Tile Size property in the generator to (1, 0.2, 1). This is the size of the meshes we are using. Now do the same thing for the tiles. They should also have their center Y set to 0.1. All new tiles will want to share these same settings.

Second, let's paint tops and bottoms of the tiles. Paint the top faces of each tile as white, and the bottom ones as black. This will indicate above ground / below ground respectively.

*NB: You can use "Show Backfaces" to easily see the far sides of cubes so you don't need rotate all the time to paint everything.*

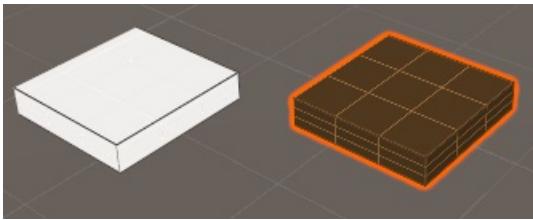


If we run the generation now, it will fail.

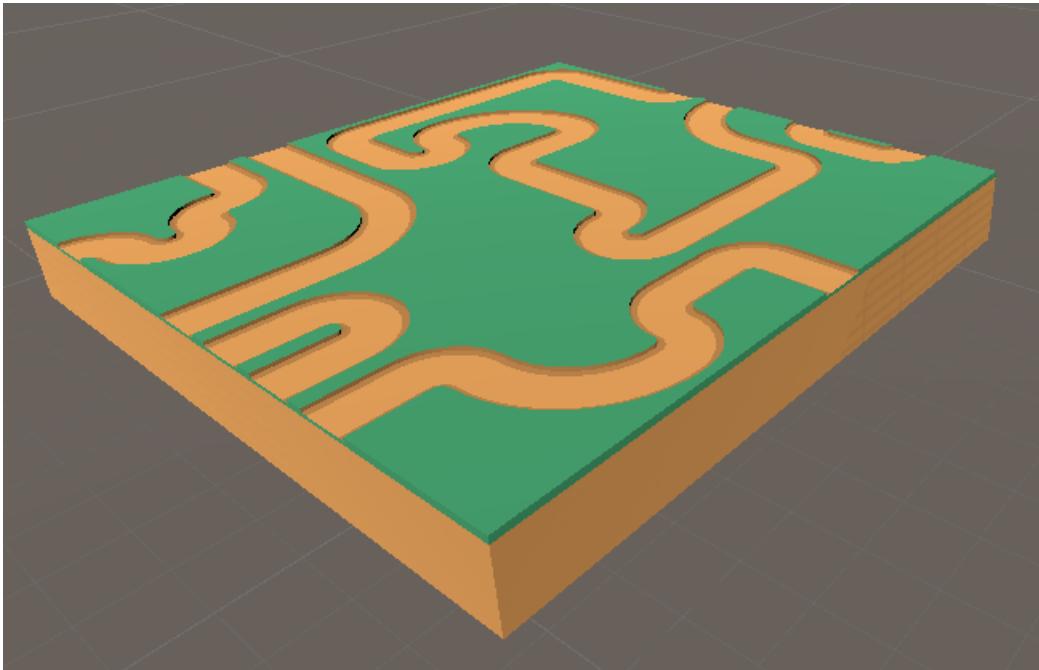
Failed to complete generation

This is because we've colored the tiles so that they no longer stack, but we haven't provided anything to stack above or below them. Tessera tries to fill the *entire* generator bounds with tiles, and fails if it cannot do so. So we need some more tiles.

Create an empty called `tile_air` and from the assets load `tile_dirtHigh`. Give both of these the TesseraTile component, add them to the Generator's tile list, and set their Tile Size and Center as with the other tiles. Now paint them. `tile_air` should have all 6 faces painted white, and `tile_dirtHigh` should have all 6 faces painted black.



Now generation should be working again, and you'll get a 3d, if flat, landscape.

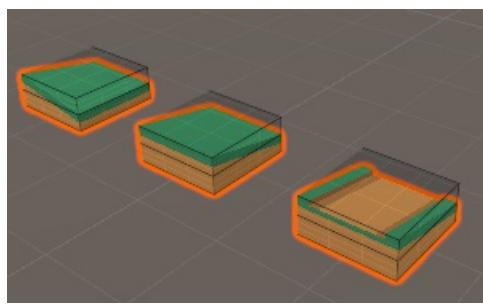


*NB: You may notice that instantiating this many tiles takes quite a bit of time. This can be undesirable for a game. If you select the air and dirt tile, and enable "Instantiate Children Only" then they will no longer be instantiated (as they have no children), speeding things up considerably.*

## Adding slopes

We need to add even more tiles before the surface can crinkle. Let's take 3 more from the tower defense assets: `tile_slope`, `tile_cornerOuter` and `tile_straightHill`. Again, TesseraTile component, Tile Size (1, 0.2, 1), Center (0, 0.1, 0).

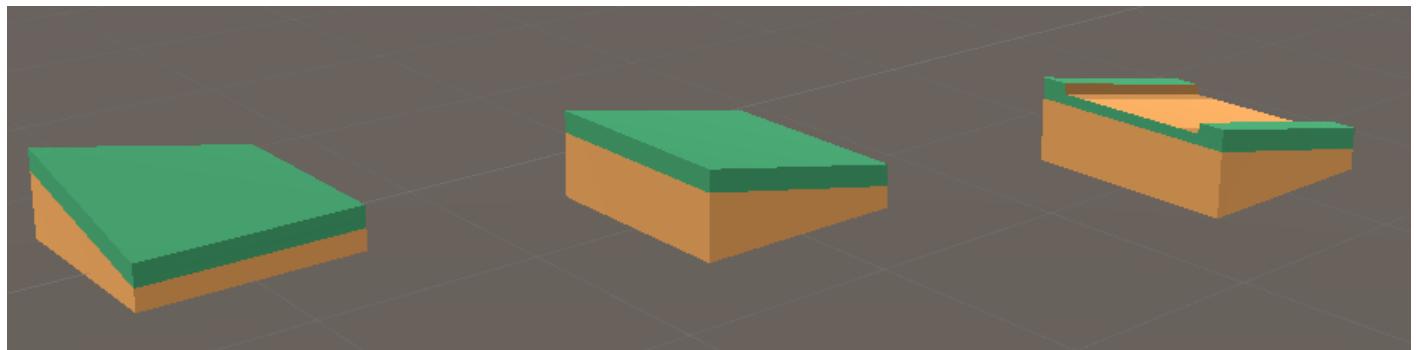
You'll notice that the tile meshes poke out the top of the paint cube. Use Add Cube to stack a second cube on top of the first so that the meshes are completely contained within. See the [big tiles tutorial](#) for details.



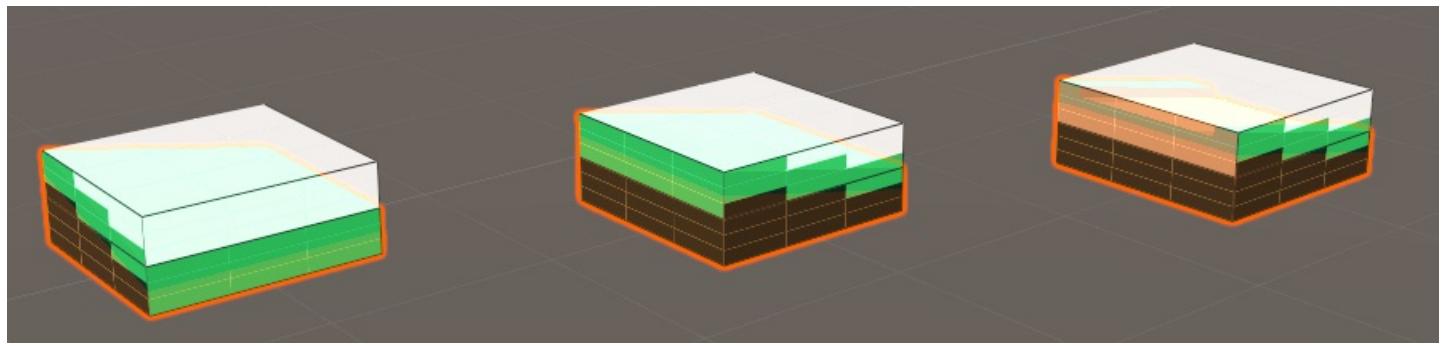
Now, we need to paint the cubes. We need to be careful when painting them. We want to ensure that all our existing tiles connect to these new tiles at the correct places. And we need to make sure that the sloped sides can connect only to each other.

Rather than give the sloped sides an extra color, we will paint them with a recognizable pattern. That will serve as a reminder. Sometimes we'll paint the pattern, and sometimes the reflection, indicating which direction the slope is running. Tessera will recognize this, and connect them appropriately. You can use the "Pencil" tool to paint patterns.

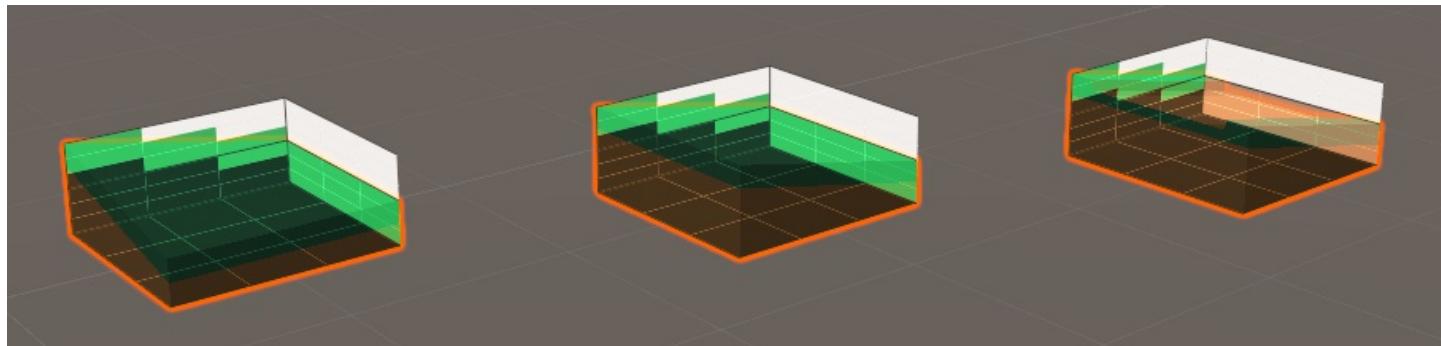
Paint the tiles. Before:



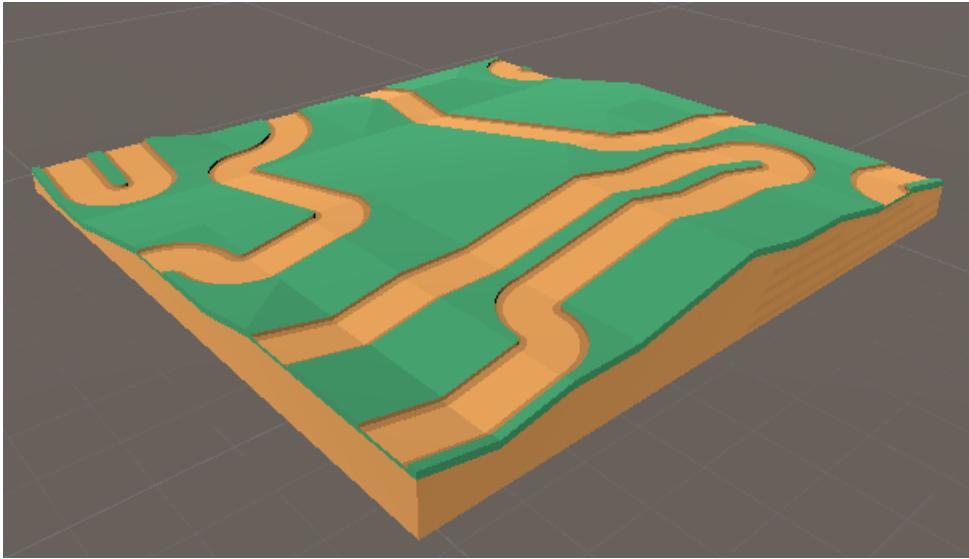
After painting:



After painting, showing backfaces:



Add the new tiles to the generator, and you should be rewarded with a undulating landscape:



## Adding a sky box

You may have noticed that sometimes the generator fills the entire volume with nothing but dirt, or nothing but air. There's nothing in what we've generated so far that prevents that. The generation algorithm will often surprise you out like that - anything that is a legal arrangement will occasionally be generated, and legal arrangements aren't always what you intended. One easy fix for this is adding a skybox to the generator. A skybox constrains what tiles can be placed on the boundary of the generated volume.

In this case, we want to force the top of the area to be air, and the bottom to be dirt, and we don't care about the sides. That will force there to be a surface somewhere between the top and bottom. Create an empty with the TesseraTile component, and paint the top face white and the bottom face black. Then assign it to the Skybox property of the generator. This will fix things as desired.

# Path Constraints Tutorial

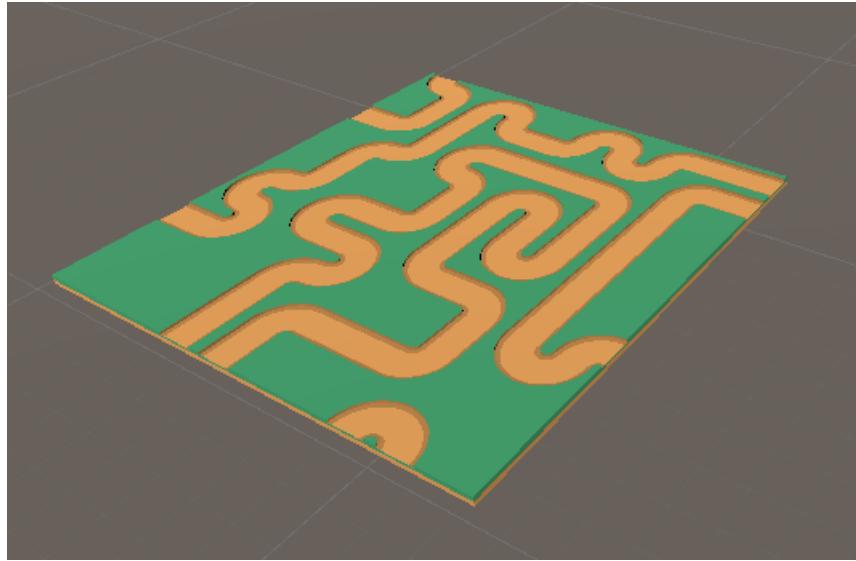
This tutorial continues from the [Getting Started](#) tutorial. It is recommended you complete that one before starting this.

The final result can be found in the sample scene *GrassPathsWithPathConstraint*.

## NOTE

Path constraints are only available in Tessera Pro

So far we succeeded in generating a level composed of grass and path tiles:



However, for many purposes, a level like the above is not acceptable. If the player is only able to walk along the path, then it's not possible for the player to walk over all the path tiles of the map, there is simply no route between them.

So far, all our generation has been *local* - that is, we've controlled what tiles are placed next to each other, without any regard for the overall structure. Now we are going to use the [PathConstraint](#) to assert some *global* behaviour on the map. The path constraint can be used in various ways, this is only an introduction.

## Setup

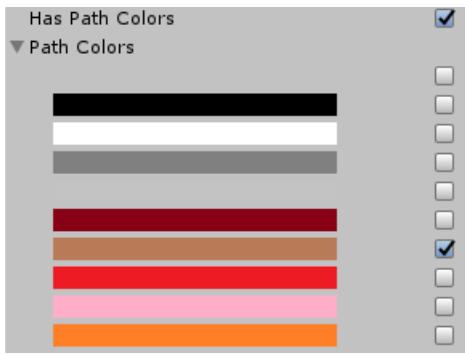
First select the generator object. Enable the `backtrack` option. Backtracking makes the generator try harder to find a viable solution. It's necessary when using the path constraint as the generator can otherwise get stuck trying to find a valid path.



Next, add a `Path Constraint` component to the generator. We need to configure the constraint, by telling it what we consider a path.

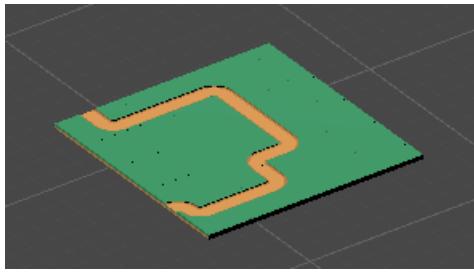
Recall that we colored the tile edge green if that edge was grassy, and brown if that edge had a path. We can give that information to the constraint.

Check the "Has Path Colors" checkbox, and then check the color corresponding to the path.



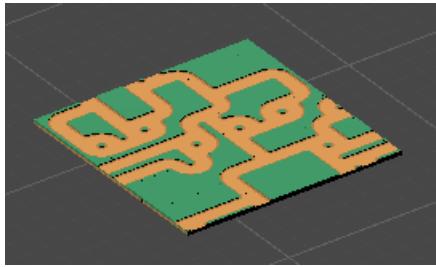
This tells the generator to search for all the sides of tiles that have that color painted in the center of the face. If two such sides connect together, then it considers there is a path between those two tiles. The constraint then ensures that all path tiles connect to each other.

If we run the generator now, it'll only ever generate a single path:



## Getting fancier

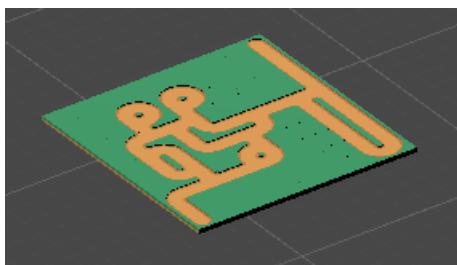
Let's add the `tile_split` tile to the generator, as you can get a lot more interesting paths once you allow junctions. The constraint will still ensure that it's always possible to walk across the map.

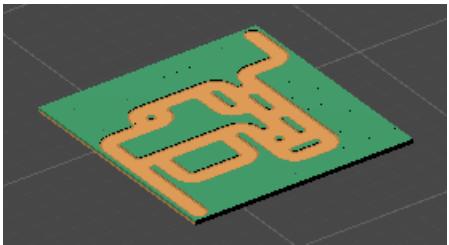
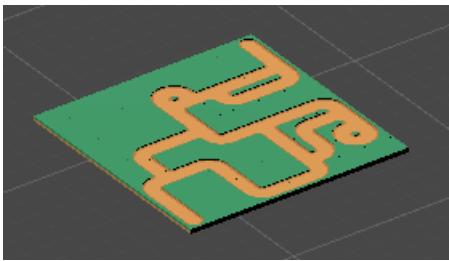


Add a `tile_endRound` to opposite corners of the generated area as a [pinned tile](#). Pinned tiles are ignored by constraints (unless using [PinType.Pin](#)), but the brown painting on these tiles forces adjacent tiles to be paths, and thus the path constraint must connect them together.

A few last tweaks: Set the skybox to the `tile` object to stop the path going off the edge of the map. And set the weight of the `tile` object to 10, increasing the ratio of grass tiles to path tiles.

Now we have a generator that makes self-contained full navigable maps. Here's a few results.





## Additional options

The Path constraint default to ensuring all tiles are connected, but has several other options too.

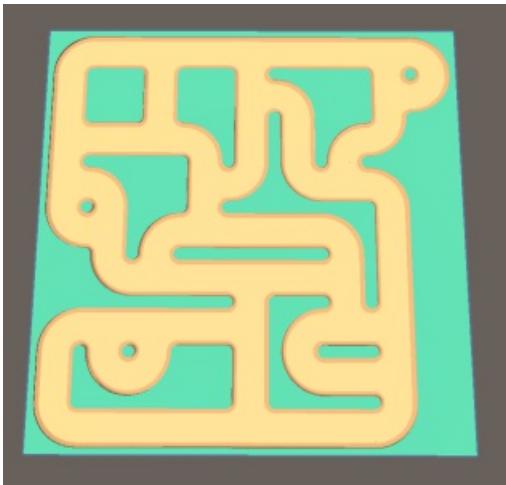
### Connected

When connected is true, then the constraint forces that all path tiles must have a contiguous path between them.

Connected is on by default.

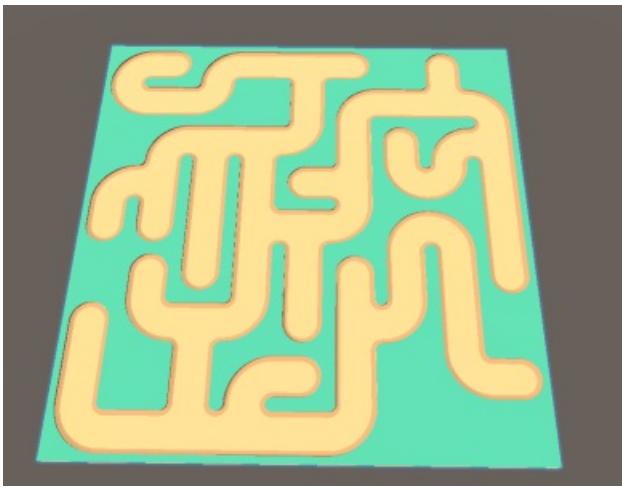
### Loops

Setting loop forces there to be at least two non-overlapping valid paths between any two connected path tiles.



### Acyclic

Setting acyclic to true bans all cycles, forcing a [tree](#) or [forest](#).



### Prioritise

Prioritise is an experimental features. When true, Tessera prefers picking cells near the path for generation over other tiles. This can improve search quality.

### Parity

Experimental setting. Enable this if your path tileset includes no forks or junctions, it can improve the search quality.

# Generating on Mesh Surfaces Tutorial

This tutorial continues from the [Getting Started](#) tutorial. It is recommended you complete that one before starting this.

The final result can be found in the sample scene *GrassPlanet*.

## NOTE

Generating on Mesh Surfaces is only available in Tessera Pro

By default, Tessera generates tiles in a regular grid - every cell is the same and shape, and placed in the same relation to each other.

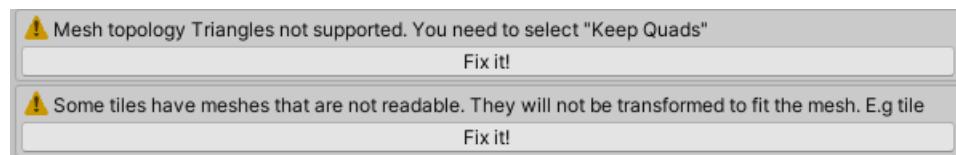
Surface mesh generation allows you to ignore this entirely. Instead, each tile will be placed in a cell corresponding to the face of a given mesh.

That means each cell will have a different shape, and the adjacency connections between cells are determined by the edges of the mesh. The local y-axis used when designing the tile will get rotated to match the normal of the face, and similarly the local x and z axes will point tangent to the face.

Let's make a micro-planet as an example.

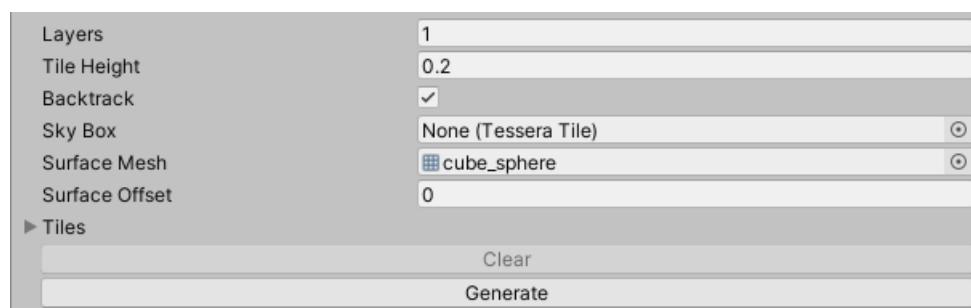
Start with the generator created in the previous tutorials and find the `Surface Mesh` property in the inspector. Click the  icon and select `cube_sphere`, a mesh that comes in the Samples folder of Tessera Pro. You can use any quad mesh (i.e. all the faces have exactly 4 edges).

Depending on which assets you use, you may see the following warnings:

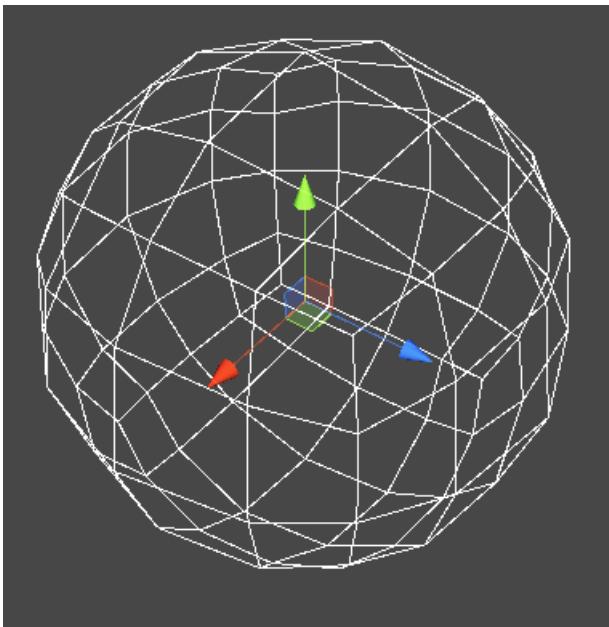


The surface mesh feature requires certain import settings to be configured. Press the fix it buttons will automatically change the assets as needed.

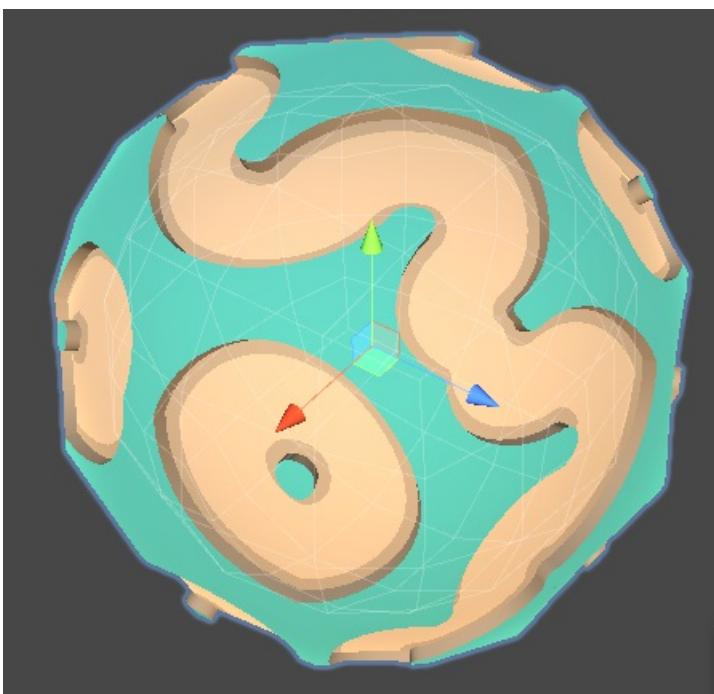
After setting a mesh, the inspector UI will have changed a little:



If you enable gizmos, you should be able to see a wire mesh.

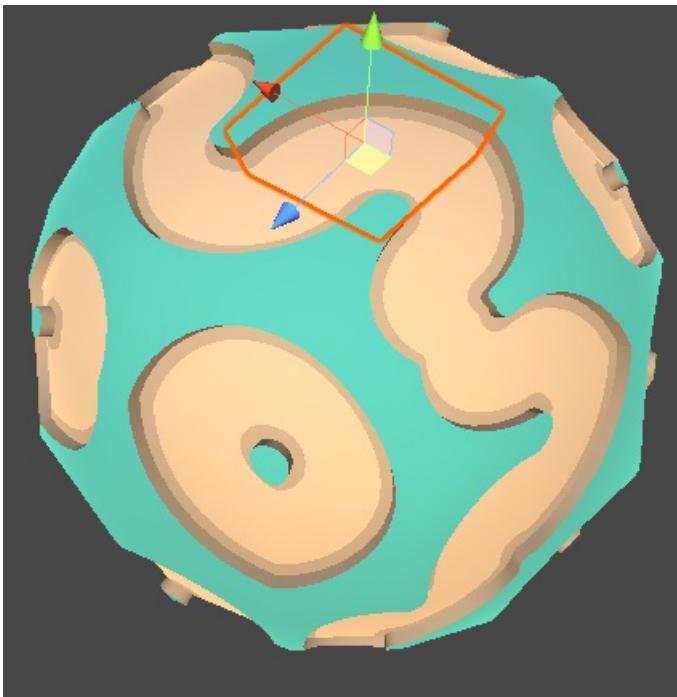


That's pretty much it. Hit generate, and see the results.



As you can see, one tile has been generated for each face of the mesh. Further, Tessera has transformed any meshes associated with the tile to make them fit the face. That gives us a totally seamless result with non square tiles, even though all the designed tiles were square.

Tessera will transform meshes found in MeshFilters, MeshColliders and BoxColliders. It'll also modify the position, rotation and scale, so the created GameObjects interact most compatibly with the other features of the game.



## Caveats

A few features do not work with meshes:

- You cannot use Tilemap Output as Unity Tilemaps expect a regular grid of tiles.
- The mirror constraint is not supported.

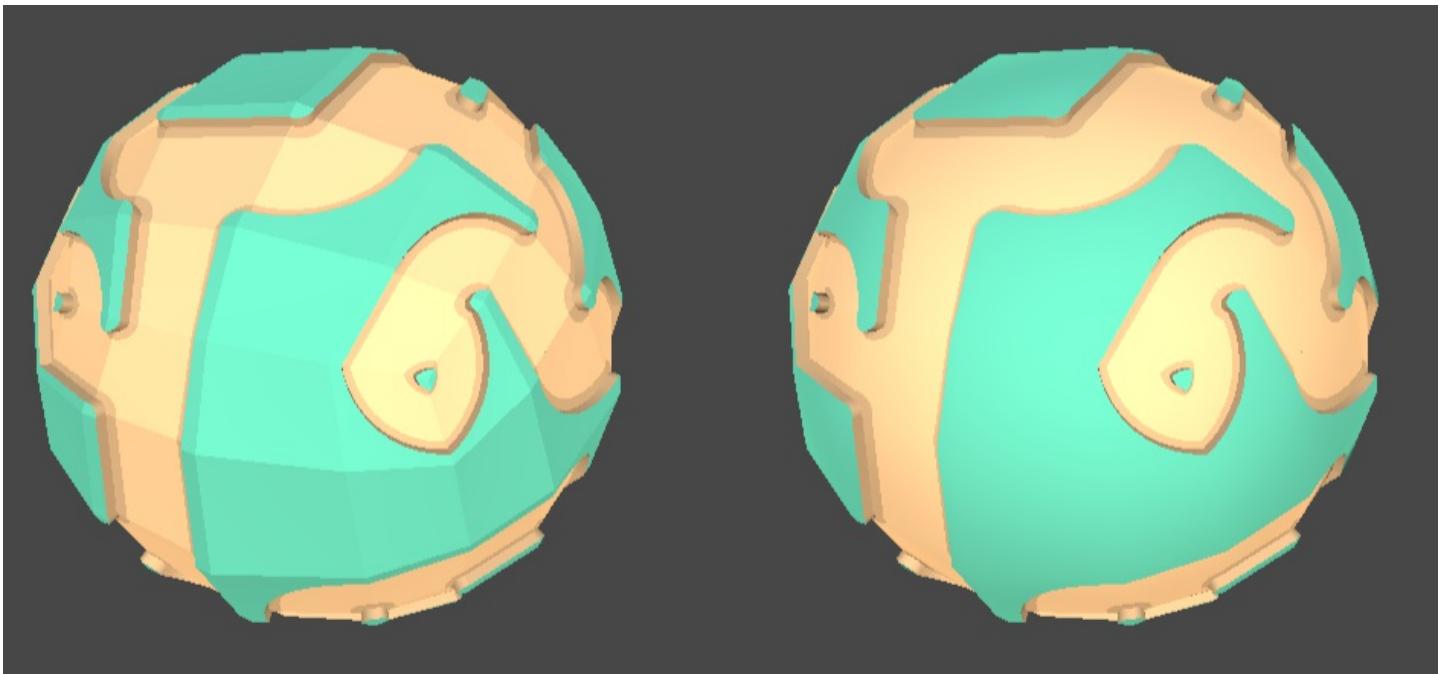
When pressing the Generate button in the Editor, it is not possible to undo the action as Unity can get extremely slow serializing all the transformed meshes.

The mesh distortion pushes around vertices, but never adds any. You may need to add extra vertices to your tile models so that adjacent tiles still line up after distortion.

You are recommended to use tiles with Reflectable and Rotatable enabled. It still works when they are turned off, but it can be hard to control the direction of each cell's local x-axis. It is determined by the order of vertices in the face of the mesh, and most editor programs do not let you easily manipulate this.

## Smooth normals

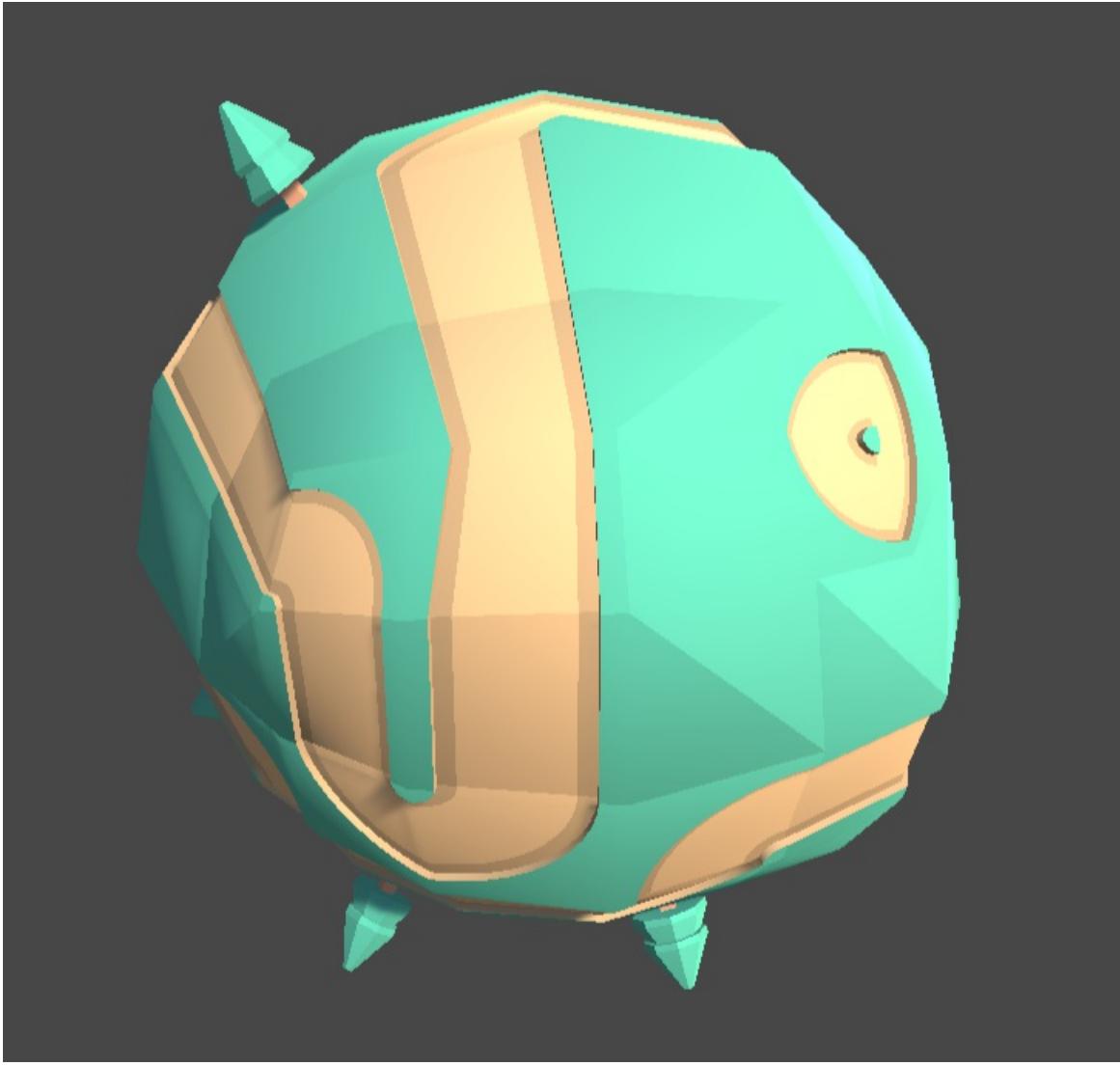
You can configure if normals should be smoothed between adjacent faces of the mesh. For smoothing to work, the surface mesh must have UVs configured.



## Multiple layers

Just as regular generators support a 3d layout of tiles, so does generating on a mesh surface. By setting the `Layers` property (or `size.y` in the API) to a value larger than one, you will stack multiple copies of the mesh surface above one another. Each layer is an expansion of the mesh along the vertex normals by the amount given in `Tile Height` (or `tileSize.y` in the API).

If you've set up your generator with the 3d tiles and skybox described in the [3d tutorial](#), then you can get results like the following on `cube_sphere`.



## Submeshes

If you are Mesh object you've selected for the surface has multiple submeshes, (i.e. multiple material slots), then you can filter which tiles are appropriate for which part of the submesh, similar to volumes, above.

The Generator inspector will automatically detect this case. Simply turn on "Filter By Submesh" and then select which tiles appear where.

Submesh 0 filter
checkboxes
inner_corner <input checked="" type="checkbox"/>
straight <input checked="" type="checkbox"/>
outer_corner <input checked="" type="checkbox"/>
innercorner <input type="checkbox"/>
wall <input type="checkbox"/>
roof <input type="checkbox"/>
outercorner <input type="checkbox"/>

Submesh 1 filter
checkboxes
inner_corner <input type="checkbox"/>
straight <input type="checkbox"/>
outer_corner <input type="checkbox"/>
innercorner <input checked="" type="checkbox"/>
wall <input checked="" type="checkbox"/>
roof <input checked="" type="checkbox"/>
outercorner <input checked="" type="checkbox"/>

Tessera Pro comes with a City example that demonstrates this.

## Triangle grids

The above tutorial demonstrated using cube tiles on a quad mesh. But if you use [triangle tiles](#) you can instead use triangle meshes.

# Multiple passes

This tutorial continues from the [Getting Started](#) tutorial. It is recommended you complete that one before starting this.

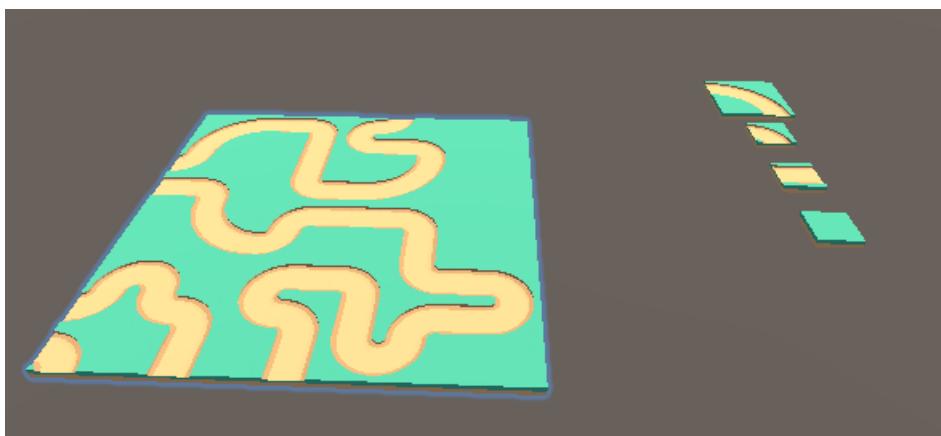
The final result can be found in the sample scene `GrassPathsMultipass`.

So far we've looked at running the generator once, and filling in the entirety of the tiles at once. As an *advanced* technique, it's possible to run multiple linked generators in the same scene, each configured differently. Running separate passes like this can achieve a variety of effects, and usually runs faster and more reliably than a single generator.

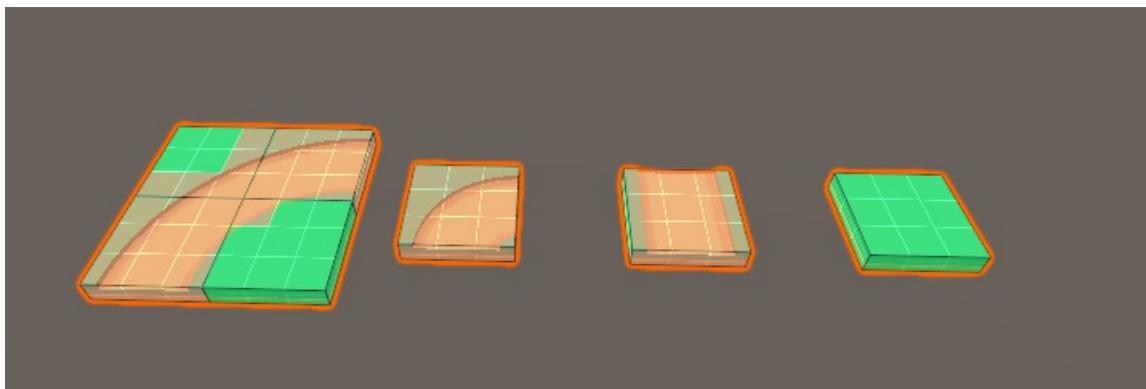
The key idea is that when you run a generator in Tessera, it creates additional game objects to fill the scene. Additionally, before the generator runs, [it scans the scene for any relevant game objects that should constrain the generation](#). So if we run two generators sequentially, overlapping in space, then the output of the first will act as pins constraining the second.

This tutorial will show you how to generate ground tiles in one pass, then add decorations using a second. The decorations will be constrained to only attach to relevant tiles.

In the [previous tutorials](#), we made a generator that generates a series of paths through a grassy plane.



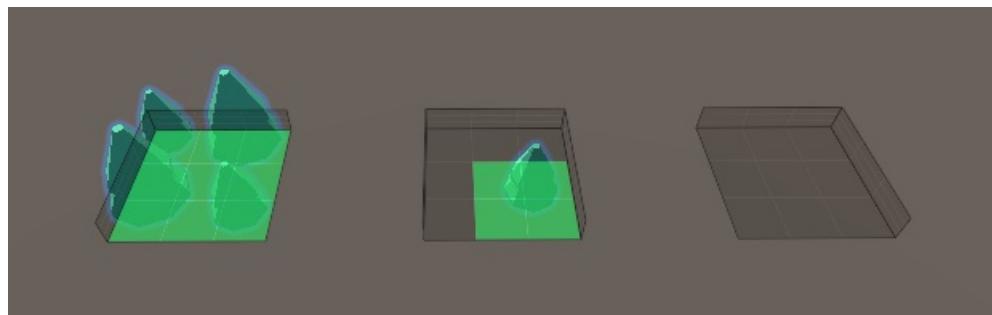
First, we'll add some extra paint on the top surface of the existing tiles to mark where is grassy, and where is path. We paint the entire top green/brown.



Now we need to make a new tileset for the decorations. With the assets supplied, make 3 tiles similar to the given ones.



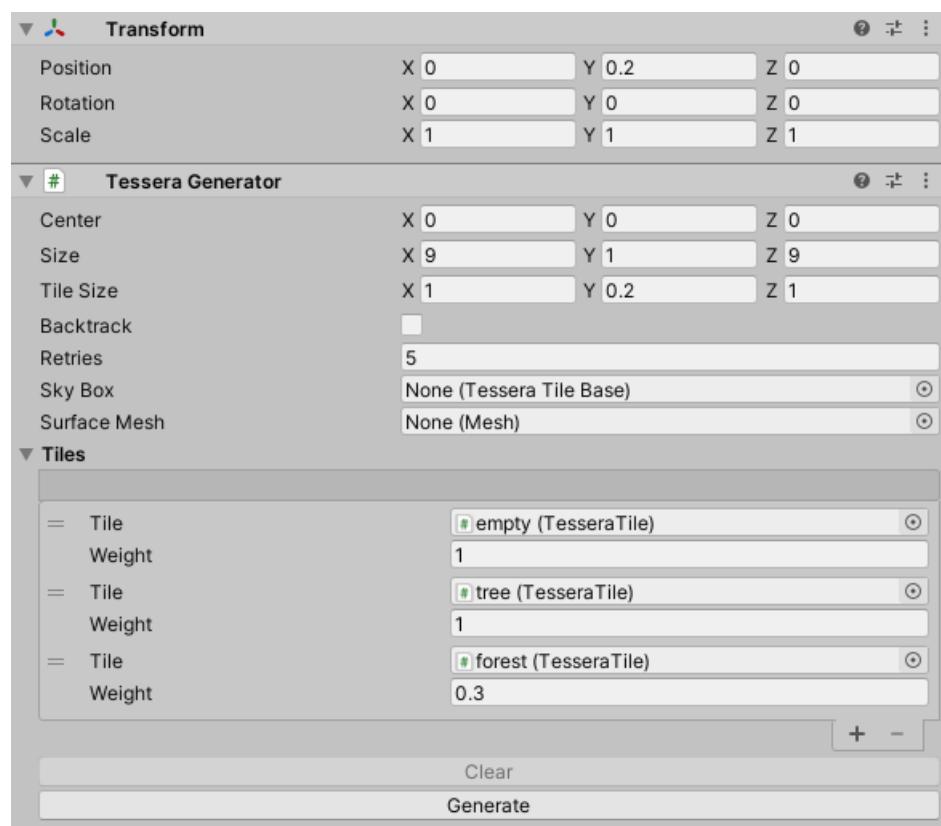
Then, using "Show Backfaces", paint the undersides of the tiles as indicated.



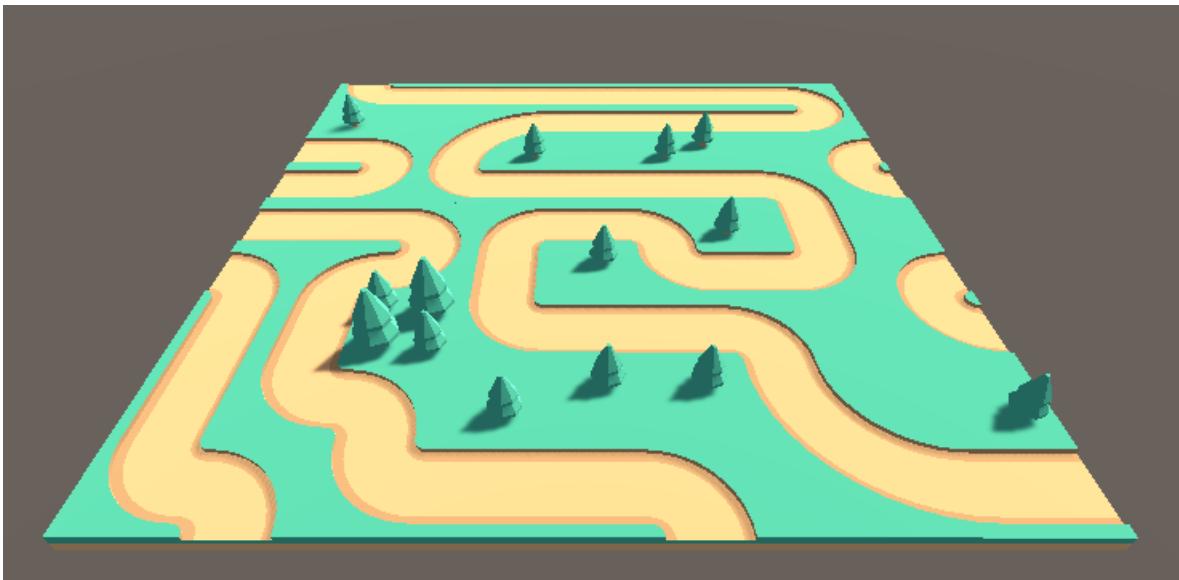
Most of the tile is left unpainted, as we don't care what it connects to. All we want to ensure is that the trees themselves are never placed directly above a path.

Now, create a second generator. This generator should be a new game object which is placed directly above the original one by 0.2 units.

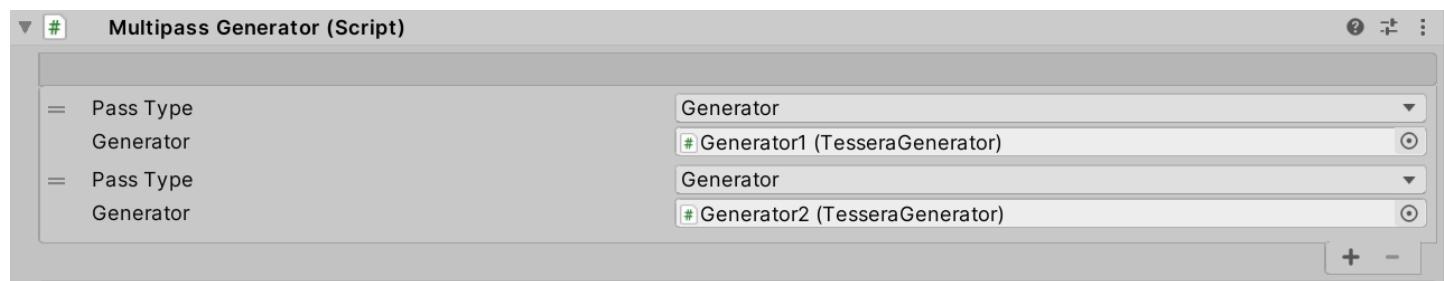
Set it up as follows.



Now run the two generators in order - starting with the path generator, then the tree generator. You should find that the trees are only generated where there is appropriate grass for them.



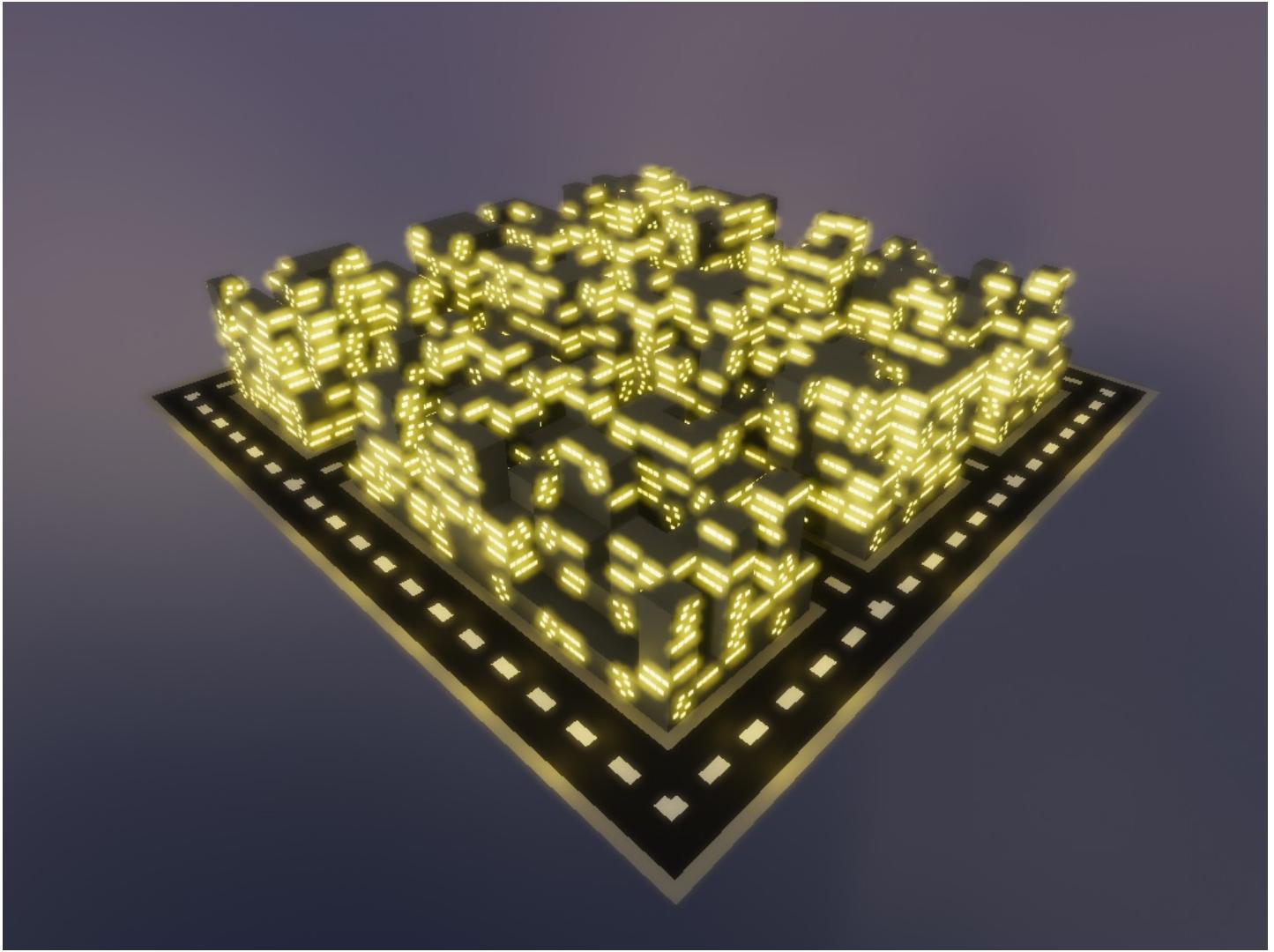
To make working with multiple passes easier, you can create a [Multipass Generator](#) that holds a reference to each of the generators you've just set up. This is a convenience for running both at once.



## Further customization

Multiple passes are even more powerful once you realize that *anything* created by the first pass will be used by the second pass. That means you are not limited to just adding extra tile paint as done above.

See the Skyscraper sample for an advanced usage.



In this sample, the road network is generated by one generator, then the skyscrapers are filled in by another. But the two generators use different grids: the buildings have a tilesize one third of the road tiles. To make this work, we [instruct Tessera](#) to replace the road tiles with big tiles that fit the smaller grid.

▼ # Tessera Instantiate Output

Parent None (Transform)

▼ Tile Mappings

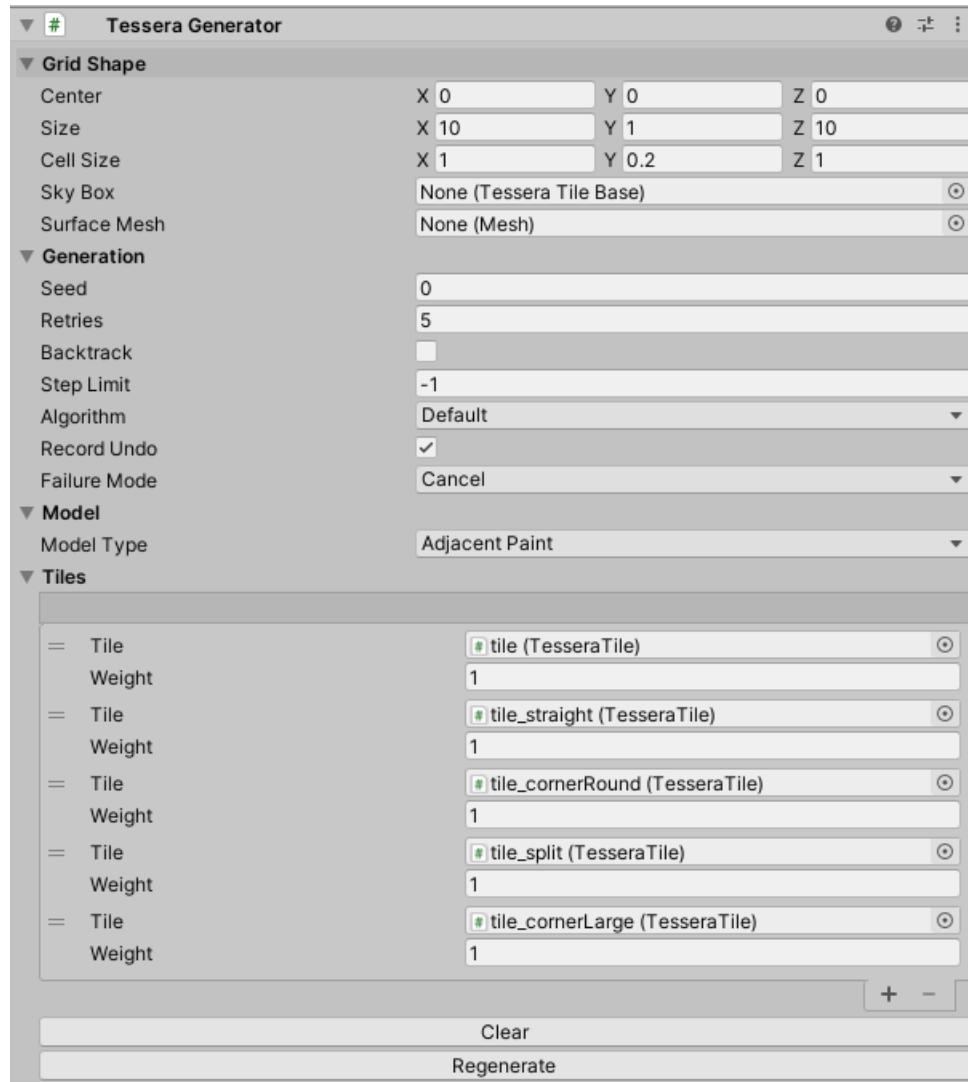
= From	# Straight_big (TesseraTile)
To	↳ Straight_small
Instantiate Children Only	<input type="checkbox"/>
= From	# Threeway_big (TesseraTile)
To	↳ Threeway_small
Instantiate Children Only	<input type="checkbox"/>
= From	# Corner_big (TesseraTile)
To	↳ Corner_small
Instantiate Children Only	<input type="checkbox"/>

+ -

The sample also activates/deactivates [pins](#) so they only apply to particular generators.

# Generator Settings

A TesseraGenerator has a great deal of options on it. Most of them are described more fully on specific pages, but here is a reference.



## Grid Shape

Depending on what sort of tiles you use, Tessera will automatically change the shape of the generator. But each shape has similar settings.

The grid is composed out of a number of cells. During generation, each cell will be filled with a single Tessera tile.

### Center

Where the center of the generator should be, in space local to the generator's GameObject

### Size

The number of cells the grid extends for in each direction.

### Cell Size

The size of each individual cell in the grid. This should match the Cell Size setting on each tile - if not, the tiles will be resized to fit.

The full size of the grid is the size of the grid multiplied by the cell size.

### Skybox

A special constraint configured directly on the generator. It lets you quickly constraint what tiles are allowed on the boundary of the grid. See [Skybox](#) for details.

### Surface Mesh

Set this to use a custom grid shape derived from a Unity Mesh. There are details on the [Surfaces Tutorial](#).

## Generation

### Seed

Fixes the random values used during generation. This gives repeatable behaviour from generation to generation. The default value of 0 is not fixed, and means it uses the built in Unity random source.

### Retries

As generation can fail in some circumstances, this setting allows the generator to retry from scratch. See [Errors and Contradictions](#) for details.

Generally speaking, this setting is not useful if backtracking is set.

### Backtrack

Changes the algorithm so that instead of failing, it undoes the most recent change and resumes from there. This makes the generator much more likely to succeed, but can mean that it gets stuck and hangs indefinitely instead. See the [Quality page](#) for some discussion of this.

### Step Limit

This is a compromise between retrying and backtracking. When set and backtracking is on, the algorithm will try backtrack for a given number of steps, and then fail (and retry). One "step" is roughly speaking, trying to place a single tile.

The default of -1 means no step limit.

### Algorithm

Controls some core details of the algorithm. Feel free to experiment. In theory, some of these are faster, but less reliable than the default algorithm, but I've never seen much difference.

### Record Undo

If you use the [buttons](#) to run the generator in the editor, then it will record the changes to the scene so that they can be undone with Ctrl+Z. This can be slow, you are recommended to disable it if you are having performance problems.

### Failure Mode

By default, if generation fails and all the retries have been exhausted, then the generator stops and logs an error message. This setting forces the generator to dump a partially completed generation for diagnostic purposes.

See [Errors and Contradictions](#) for details.

## Model

The "model" is how Tessera decides what tiles can be safely placed next to each other.

### Model Type

#### Adjacent Paint

This is the default model, and most of the docs assume that you have this set. Each tile is painted as shown in the [getting started tutorial](#). To compute if two rotated tiles can be placed adjacently, the faces of each tile are compared. If the paint patterns match, either exactly, or according the rules set up in the [palette](#), then they can be placed together.

## Overlapping Model

In the [Overlapping Model](#), you need to supply one or more samples. The sample is split up into small patches, and the model ensures that the outputted result is entirely composed out of overlapping copies of those patches.

## Adjacent Model

The [Adjacent Model Type](#) works the same as overlapping model, only the samples are analysed for pairwise adjacencies between tiles instead of small patches.

## Mesh Edges

In the [Mesh Edges Model Type](#), each tile is analysed based on the Mesh attached to the game object. Two tiles can be placed adjacently if their meshes would form a continuous surface.

## Samples

Samples is a list of Unity game objects (they may be prefabs). This setting is only used for Overlapping and Adjacent Model Types. The children of the sample are assumed each correspond to tiles in the generator (matched by name)

## Mesh Edges Tolerance

This is only used with the [Mesh Edges Model Type](#). It controls the degree to which mesh vertices snap to each other.

# Tiles

The list of tiles configures what tiles the generator can place in each cell. The tile objects themselves have a number of configuration options.

You can use several types of tiles: [Cube](#), [Square](#), [Hex](#), [Triangle](#). You cannot mix and match tile types.

Each tile has a Weight variable. This controls how likely the tile is to be selected. A tile with a weight of 10 is twice as likely to be selected as a tile of weight 5.

A weight of 0 means that the tile will never be *randomly* chosen. But it can still be generated, if it's the only legal choice for a given cell.

# Buttons

The generator can be run straight in the editor with these buttons. If you want to run the generator at runtime, use the [API](#), the [Bolt Unit](#) or the included `Generate On Start` script.

## Clear

Deletes all child objects of the generator. If a [generator output](#) is set, then that will be cleared instead.

## Generate / Regenerate

Runs the generator, and instantiates the tiles in the scene. Prefab tiles will be created as prefab links. If a [generator output](#) is set, then will be used to instantiate the tiles instead.

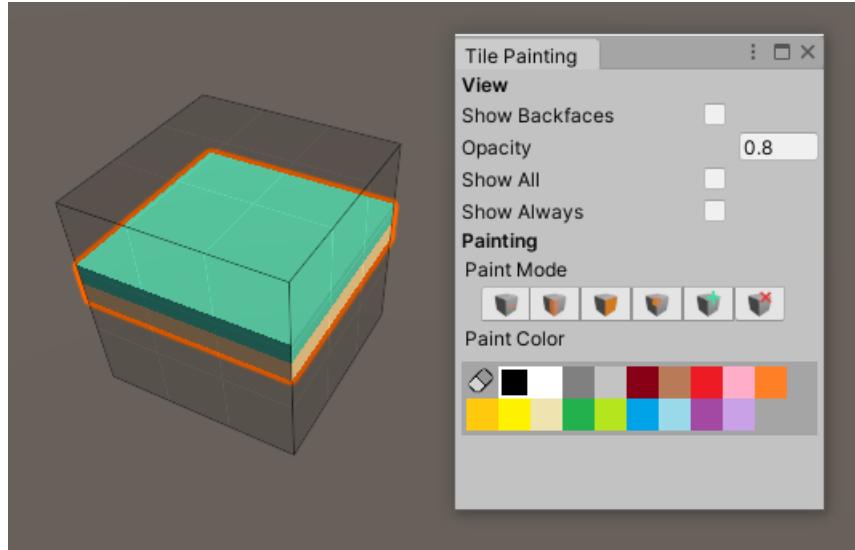
## Clone Sample

Only available for Model Types Overlapping and Adjacent. This instructs the generator to copy the sample exactly, with no generation. This is useful to determine that your sample is being correctly processed.

# Tile Painting Controls

While painting tiles, there are various settings to control the painting process.

For more details on painting, see the [Getting Started Tutorial](#)



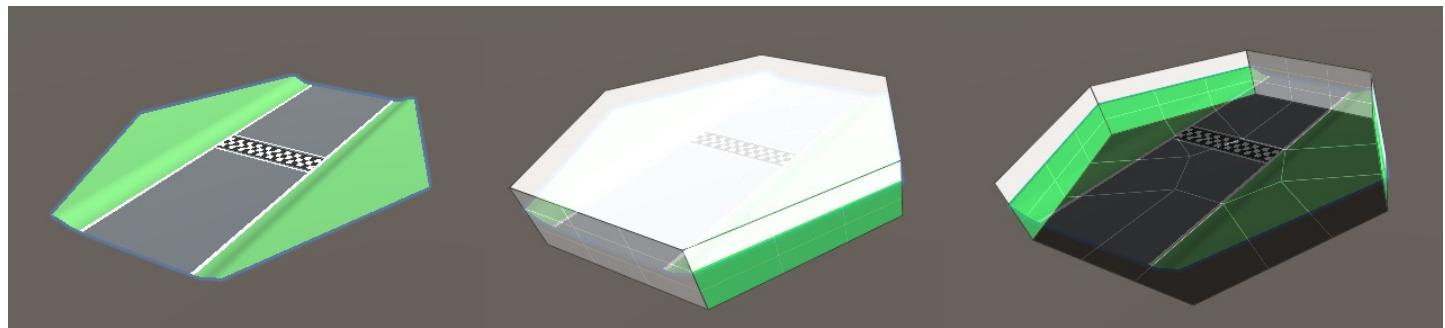
## View Settings

Show Backfaces

*Shortcut:* Shift+Z

If set, then the editor only shows paintable faces that point away from the camera. Toggling this lets you paint every face without having to move the camera.

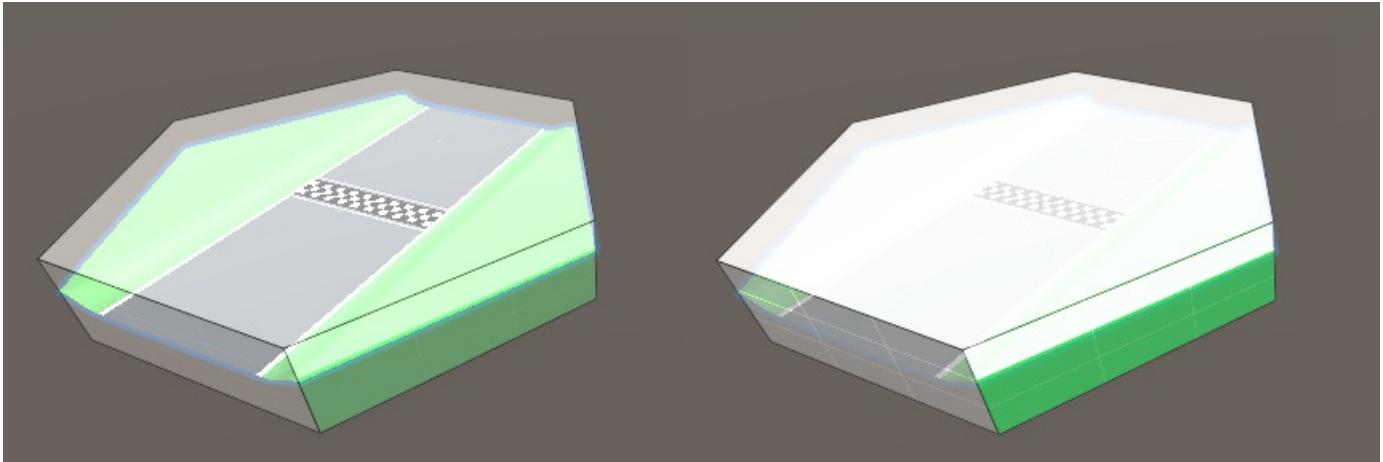
A hex tile, with front and back painting turned on.



Opacity

*Shortcut:* Shift+Q

Controls the opacity the paint overlay.



## Show All

*Shortcut:* Shift+A

When enabled, the paint overlay is shown on every paintable tile in the scene. Otherwise, it's just selected tiles.

## Show Always

*Shortcut:* Ctrl+Shift+A

When enabled, the paint overlay is always shown. Otherwise, it's only shown while you have one of the [Paint Modes](#) active.

# Paint settings

## Paint Mode

When one of these buttons are depressed, it enables painting mode. Clicking on the scene will add some paint or edit the tile. Each mode operates differently.

Pencil Mode 

Paints a single subface.

Edge Mode 

Paints the subfaces along an edge.

Face Mode 

Paints an entire face.

Vertex Mode 

Paints the subfaces touching a corner.

Add Cube 

This alters the shape of the tile. See the [Big Tile](#) for more details.

Remove Cube 

This alters the shape of the tile. See the [Big Tile](#) for more details.

## Paint Color

Sets the color to paint. The different colors and their meanings can be configured with a [Palette](#).

The first entry, an eraser, means removing any paint.

# Keyboard Shortcuts

All shortcuts are used while in tile painting mode. The key combinations can be changed from the defaults using Unity's Shortcut Manager.

Command	Shortcut
Show Backfaces	Shift+Z
Show All	Shift+A
Show Always	Ctrl+Shift+A
Hide All	Shift+Q
Select Previous Color	Shift+X
Activate Pencil Paint Tool	
Activate Face Paint Tool	
Activate Vertex Paint Tool	
Activate Add Cube Tool	
Activate Remove Cube Tool	

Right clicking while using a paint tool will erase the highlighted area, instead of painting it the selected color.

# Constraints

The basic configuration of a generator involves setting up tiles, and painting those tiles to show how they can be placed next to each other.

This page documents further configuration you can do to tightly control the generation process.

There are many constraints available in Tessera:

- [Scene Constraints](#)
  - [Pins](#) fix a particular tile in place
  - [Volume constraints](#) filter a particular area to a subset of tiles
- [Generator Constraints](#)
  - [CountConstraint](#) - ensures the number of tiles in a given set is less than / more than a given number.
  - [MirrorConstraint](#) - ensures the output remains symmetric.
  - [PathConstraint](#) - detects contiguous paths between tiles, and ensures various properties about those paths, such as connectedness.
  - [SeparationConstraint](#) - ensures that the given tiles are spaced at least a certain distance apart
  - [BorderConstraint](#) - ensures that the given tiles are placed at the border of the generation, similar to [skyboxes](#)
- [Skyboxes](#) controls how tiles on the boundary of the generation work
- [Submesh filters](#) applies per material settings when working with [surface meshes](#).

## Scene Constraints

Scene constraints are placed directly in the scene to control specific locations. They are described in more detail [here](#).

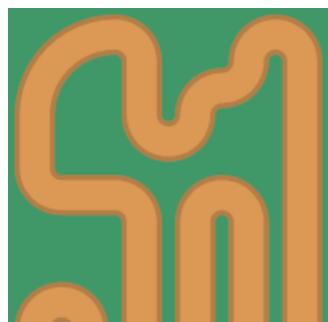
## Generator Constraints

Generator constraints are Unity components added to the same object as the TesseraGenerator component to control global behaviour of the generator. They are described more [here](#).

## Skybox

Setting the skybox property of the generator will automatically constrain all tiles on the boundary of the tile area. The skybox should be a TesseraTile component. Whatever is painted on the top of the skybox, will constrain the top of every tile on the topmost layer of the generator. Similarly for the other sides of the cube.

In this example, the skybox has been used to force the bottom edge to be all paths, and the other edges to have no paths.



### Notes

The tile used for a skybox should not be a big tile, this will cause weird behaviour.

If a pinned tile constraint and the skybox both apply at a particular location, the tile constraint takes precedence.

Note that a given skybox face is repeated as a constraint for *all* tiles which have a corresponding face on the boundary. This can sometimes cause counter-intuitive results. For example, if you have a 3d example like [in this tutorial](#), then you cannot have a skybox that is "grass" on the sides. Doing so allows surface tiles, but would stop air and solid tiles from touching the boundary, which will cause the boundary to fail.

The solution to this is to make a new [palette](#) color just for the skybox. You can then set that color connect with *multiple* other colors, so that it's possible to place all the tiles that can abut the boundary.

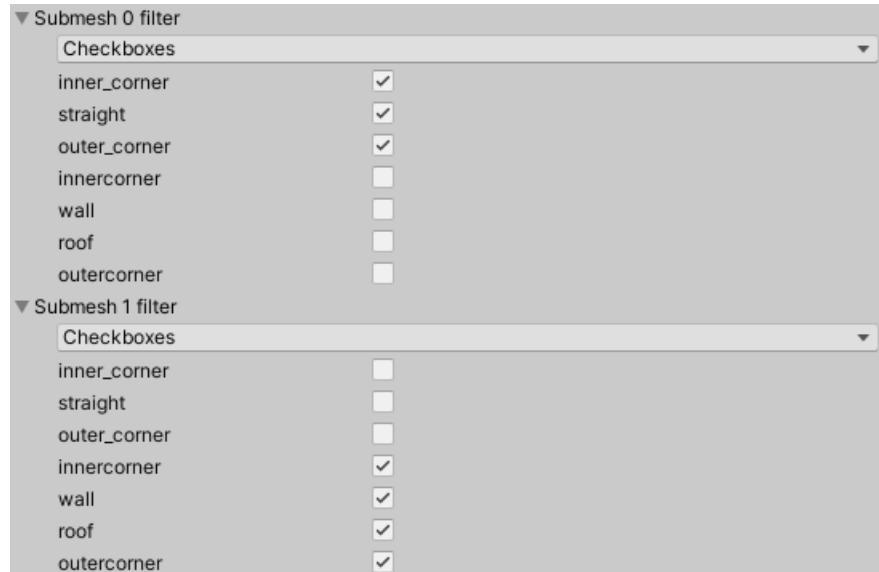
## Submesh filters

### NOTE

Submesh filters are only available in Tessera Pro

If you are [generating on a mesh surface](#), and that mesh has multiple submeshes, (i.e. multiple material slots), then you can filter which tiles are appropriate for which part of the submesh, similar to volumes, above.

The Generator inspector will automatically detect this case. Simply turn on "Filter By Submesh" and then select which tiles appear where.



# Initial Constraints

Initial constraints configue the intial conditions of the generator. They generally set up by adding behaviours into the world.

- [Pins](#) fix a particular tile in place
- [Volume constraints](#) filter a particular area to a subset of tiles

## Pins

Sometimes you just want to place a tile at a particular location without Tessera choosing for you. For example, you might want to place a entrance manually at one side, or write custom code to draw path ways for you. Pinned tiles lets you do this, and then have Tessera generate the rest of the tiles for you.

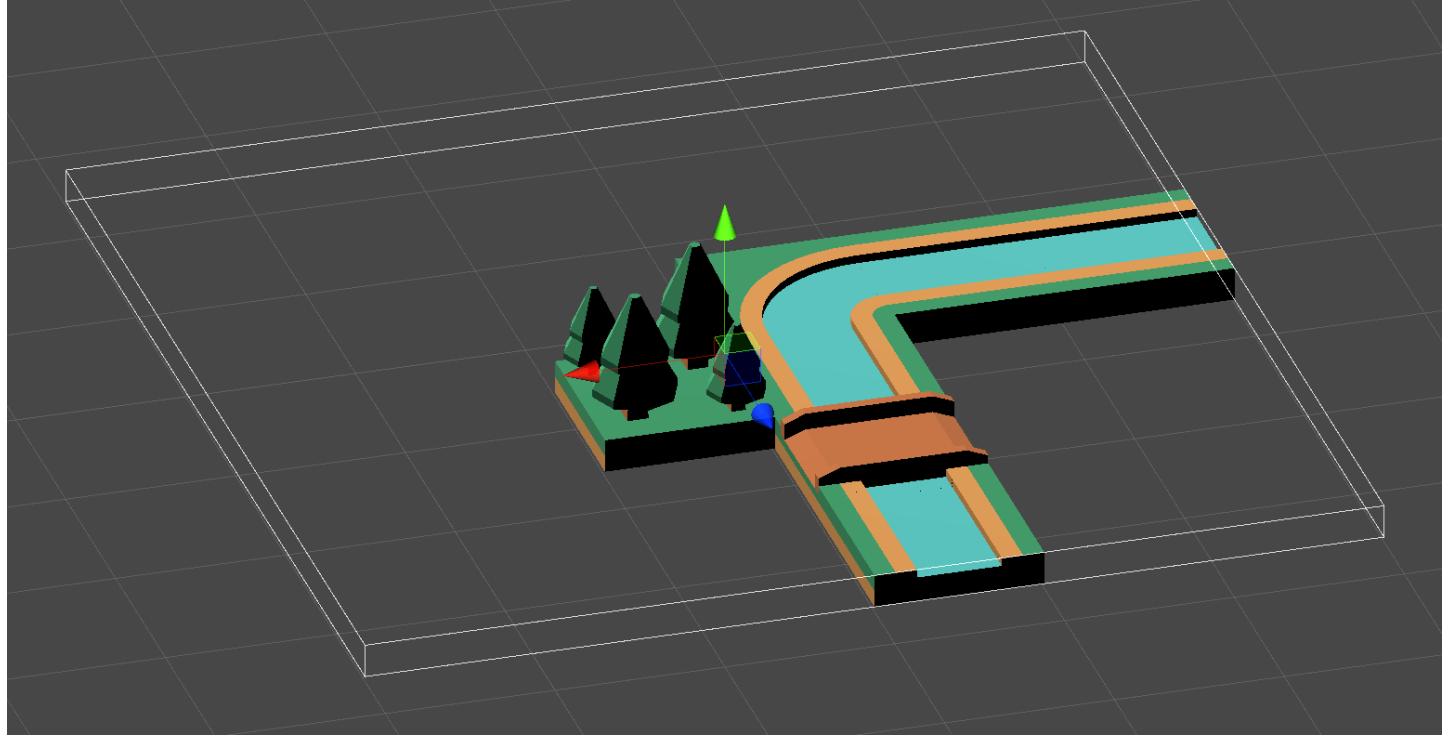
There's multiple ways to specify pins - all of them involve putting a game object inside or adjacent to the generator area, in the position of the cell you want to affect. They will be automatically detected and incorporated into the generation.

The most flexible way is to great a game object with the [TesseraPinned](#) component. You can then specify the [PinType](#) and the tile to pin as properties of that component.

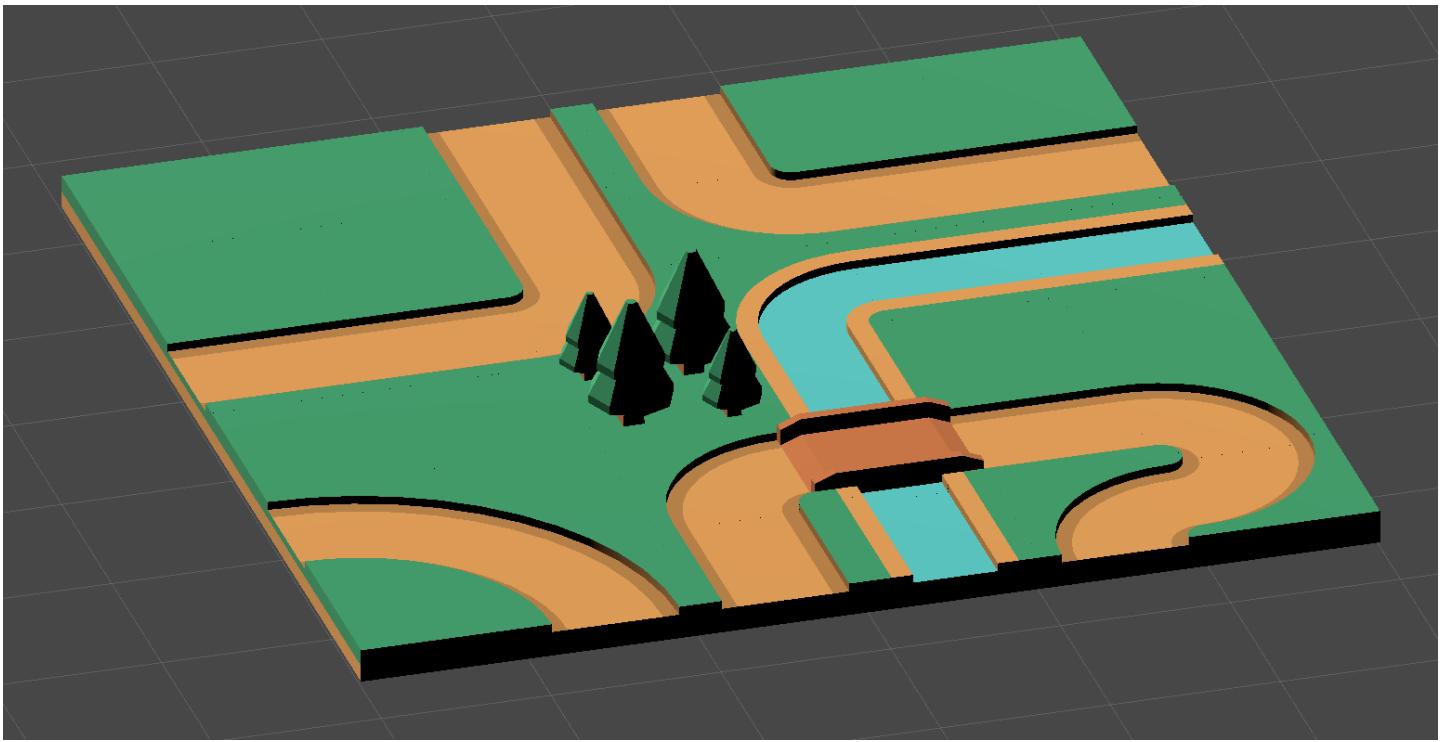
As a convenient short cut, you can also just place a Tessera tile directly. These will be interpreted the same as a TesseraPinned component that references that tile, and has pin type [FacesAndInterior](#).

You can also place an object with both a Tessera tile component and a TesseraPinned component. The TesseraPinned will be assumed to reference the tile component on the same game object.

Here is an example. A generator is set up and a few tiles manually placed.



When the generator is run, the new tiles join up with the placed tiles.



## Configuring pins

TesseraPinned has two key properties:

- **Tile** determines what tile to use for pinning.
- **PinType** controls how exactly the pinned tile effects generation.

These two values are inferred in some cases, as described above.

There are three types of pin:

### **PinType.Pin**

With this pin type, the generator is forced to generate the referenced tile at the location of the pin.

The referenced tile must already be in the generator's tile list (though it can have weight 0).

### **PinType.FacesOnly**

With this pin type, the faces of the referenced tile are used to constrain the cells adjacent to the location of the pin, and the generator is free to pick any tile for that cell.

### **PinType.FacesAndInterior**

With this pin type, the faces of the referenced tile are used to constrain the cells adjacent to the location of the pin. Also the cells covered by the pin are masked out so no tiles will be generated in that location. You may therefore need to manually place a tile or other game object to fill the gap. Unlike [PinType.Pin](#), the referenced tile *does not* need to be in the generator's tile list.

The different pin types can be used to generate a variety of effects. FacesOnly, for example, is good at restricting the generation along a given plane, without forcing any particular tile. FacesAndInterior allows you to put in custom tiles (often called "hero" pieces) that never otherwise appear in generation. And Pin is good for forcing a particular tile to be placed, while still allowing that cell to be part of generation. That can be important if you need the cell to interact with [generator constraints](#) or other Tessera features.

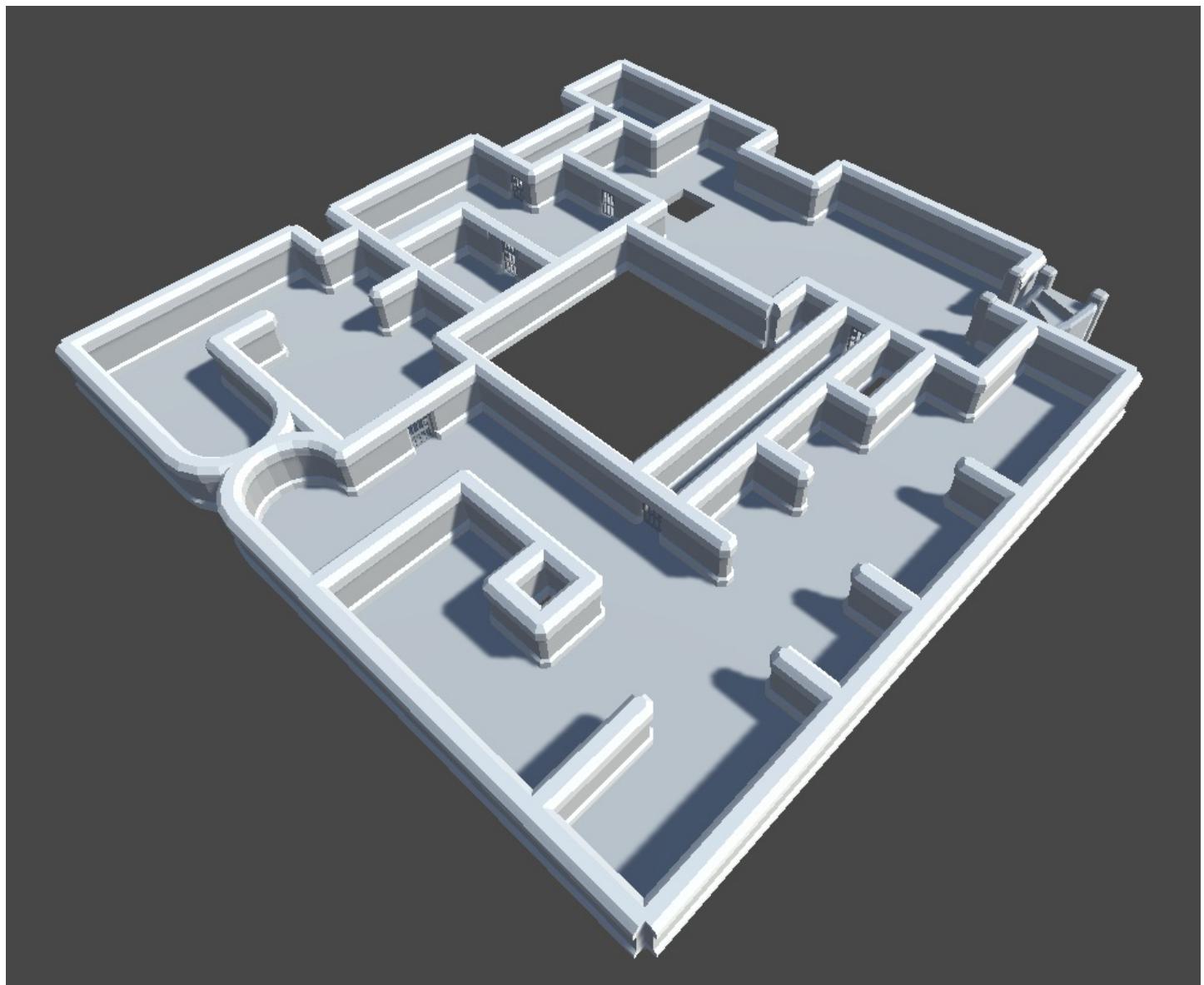
## Volumes

Add a GameObject with the [Tessera Volume](#) component to filter an area of cells to only use a subset of tiles.

To use it, add a game object with the volume component, and set the cells you want to filter to. You can optionally specify a generator for convenience, but it's not necessary.

Then you need to specify the actual volume of space that the collider affects. You do this by attaching more components to the volume's game object. You can either use Unity [Collider](#), or you can use any Tessera Tile component. Tessera will consider a cell of the generator to be inside the volume if the center of that cell is inside at least one collider or tile component.

Here's a dungeon generated with a box collider volume in the center, configured to filter to just an empty tile.



Unlike [pins](#), volumes have no way to configure the rotation and reflection of the generated tile.

Volumes can also be used to remove tiles from the generator entirely, by setting the volume type to [MaskOut](#). This behaves similarly to pinning tiles with [FacesAndInterior](#) - the missing cells interact differently with constraint.

## API

The default behaviour of generators is to search the scene for appropriate pins and volumes. This can be slow for large scenes, and it's not suitable for advanced customization.

You can use the C# API to set pins and volumes directly.

- First, disable the automatic detection by setting `searchInitialConstraints` to false.
- Get a builder by calling `generator.GetInitialConstraintBuilder`.
- Call the methods on the builder to create your own initial constraints. The objects passed do not need to be in the scene, they can be prefabs.

- Construct a `TesseraGenerateOptions` and fill the `initialConstraints` field with an array of your created constraints.
- Pass in the options as an argument when calling `Generate` or `StartGenerate`.

# Generator Constraints

## NOTE

Generator constraints are only available in Tessera Pro

These constraints control the *global* behaviour of generation. They are very powerful, but can use generation to fail more frequently.

To use them, find the game object that has the [generator](#), and add additional components to it from the [Component > Tessera](#) menu.

At present the constraints are:

- [CountConstraint](#) - ensures the number of tiles in a given set is less than / more than a given number.
- [MirrorConstraint](#) - ensures the output remains symmetric.
- [PathConstraint](#) - detects contiguous paths between tiles, and ensures various properties about those paths, such as connectedness.
- [SeparationConstraint](#) - ensures that the given tiles are spaced at least a certain distance apart
- [BorderConstraint](#) - ensures that the given tiles are placed at the border of the generation, similar to [skyboxes](#)

There is a tutorial on [how to use path constraints](#).

# Count Constraint

The [CountConstraint](#) is a [generator constraint](#) that counts the number of tiles in a given set that are generated and ensures that the total in the final output is above/below a given number.

## Eager

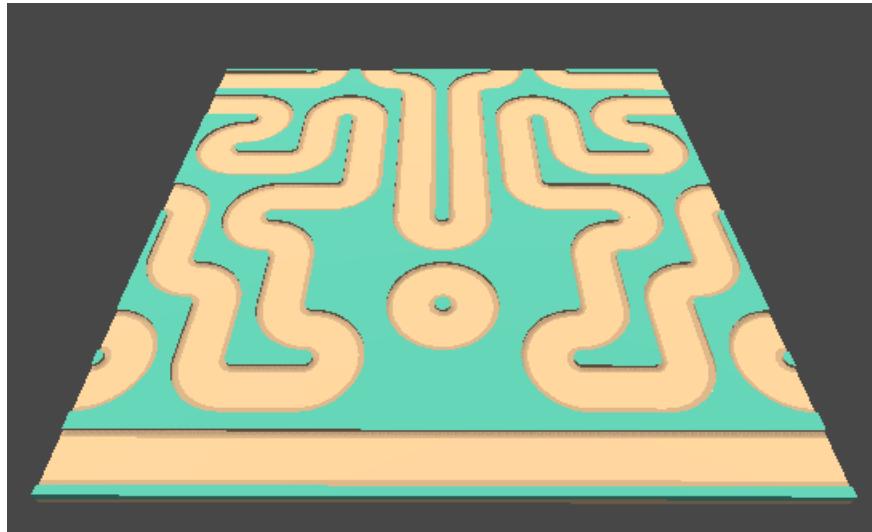
Normally, this constraint is run while generation occurs i.e. it'll count the tiles as generation occurs, and take action once the limit has been reached. This strategy generally works well, however in some cases it'll cause problems.

1. If a tile has a very high weight, but a low limit in the Count Constraint, then the limit will be reached early and the rest of the generation will have none of those tiles.
2. If the tile is very hard to place, and you've required at least a certain number of them, then the Count Constraint can wait too late before forcing placement, so the generation never succeeds.

These issues can be avoided by setting the constraint as "eager". When eager, it'll place the counted tiles first, before attempting to place any others, ensuring an even distribution, and priority positioning.

# Mirror Constraint

The [MirrorConstraint](#) is a [generator constraint](#) that ensures that the generated output is symmetric about a given axis.



You can select one axis, X, Y or Z. If you want symmetry about 2 axes, use two copies of this constraint.

The mirror constraint is unlikely to work very well unless you enable `reflectable` on your tiles.

If you have initial constraint tiles, then the empty cell that mirrors that is ignored by the mirror constraint.

## Odd vs evens

If the tile generator is an odd number of cells wide, then the center line reflects onto itself. All tiles placed on the center line must therefore be symmetric themselves. If the width is even, then there is less restriction as tiles on one side of the center line just have to connect to their own reflection.

## Specifying tile symmetry

By default, the constraint looks at the painted sides of each tile, and infers that the tile is symmetric in a given axis if the painted pattern is symmetric. It does not look at any other details of the tile. So, for example, if you had an asymmetric mesh, but with symmetric connectivity, you would need to override that.

To do so, enable "Override Symmetric Tiles" and check and uncheck which tiles should be considered symmetric. Marking a tile as symmetric means when it is placed in a cell, it'll also be placed in the mirrored cell, *with the same reflection and rotation*. Marking a tile as asymmetric means when it is placed in a cell, a mirrored copy of it will be placed in the mirrored cell. If the tile is marked as asymmetric and it does not have `reflectable` and `rotatable` set to enable that, then it cannot be placed at all.

Note that due to rotations, you may need to specify y and z symmetry even when reflecting on the x axis, and so on.

# Path Constraint

The [PathConstraint](#) is a [generator constraint](#) that forces there to be a connected path between all relevant generated tiles. See [the tutorial](#) for details on its usage.

# Separation Constraint

The [SeparationConstraint](#) is a [generator constraint](#) that forces a given set of tiles not to be placed too near each other. This can be useful to avoid things "clumping" too much, or as an alternative way of controlling the frequency of tiles, distinct from using a Count Constraint or setting the tile weight.

To use it, specify a tile or list of tiles to separate, and a distance. These tiles will never be placed within this many cells of each other. Distance is measured as individual steps from cell to cell ([Manhattan distance](#)), so on a square grid forms a diamond exclusion zone around each tile.

NB: Big tiles are not fully supported, and will always measure distances from one corner.

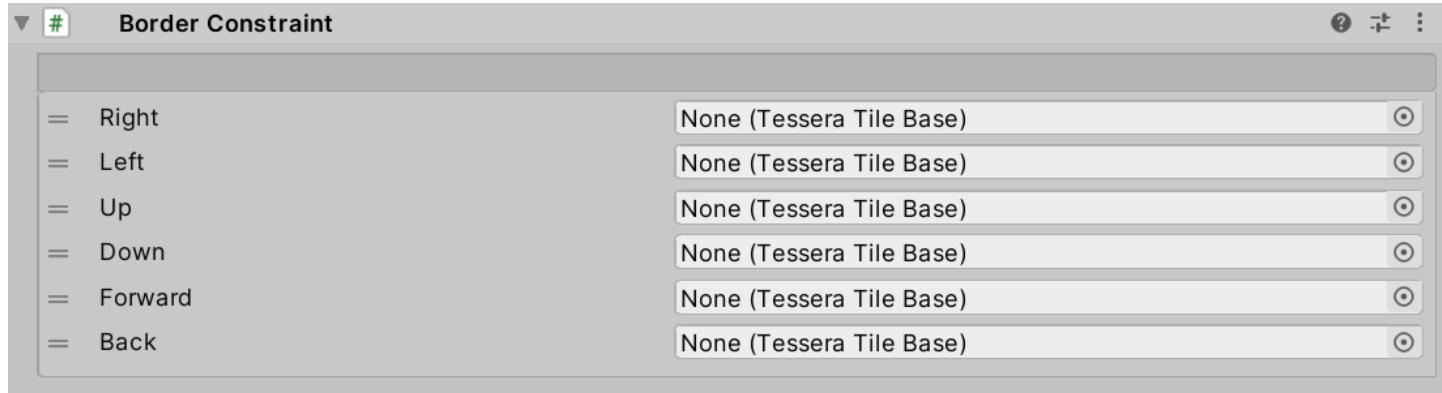
# Border Constraint

The border constraint is a [generator constraint](#) which restricts which tiles can appear on the outermost set of cells of the generator output.

It is similar to placing [volume initial constraints](#), except it automatically works out which cells to affect.

## Usage

Simple add the border constraint to the generator, then select which tiles to use for each border.



You can re-order the sides. Whichever side comes first takes priority for cells that are on multiple sides, i.e. corners.

## Comparison with Skybox

The border constraint has very similar functionality to [skyboxes](#). But they have different advantages.

	SKYBOX	BORDER
Constrains cells	by matching the paint they have along the border	by tile
Controls tile rotation	Yes	No
Works with overlapping models	only if you add paint	yes

# Animation

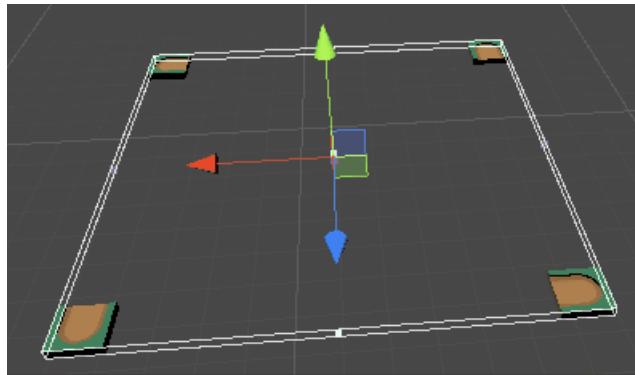
## NOTE

This feature is only available in Tessera Pro

This feature is for fun and debugging.

If you add the [AnimatedGenerator](#) to a generator, you can hit Start to run the normal generation process tile-by-tile instead of all at once. It works in both the Unity Editor and in-game.

This animation is much slower than generating all the tiles at once, but it looks cool, and it can show you where the generator is having difficulty. This can be handy if the generation takes too long, or keeps failing, due to not having the right sort of tiles.



`Seconds Per Step` indicates how long to pause between each step and `Progress Per Step` indicates how many units of work to do in a step.

Each unit of work is one of the following:

- Add a tile, and work out all other tiles that are implied by it.
- Backtrack once, done when the current configuration is impossible (if [backtracking](#) is enabled).

`Uncertainty Tile` should be a game object to use to indicate that Tessera is still thinking about a particular tile. The size of the tile indicates how many possibilities still remain. If the tile used has a [TesseraUncertainty](#) component, it'll be filled with useful info about WFC progress.

# Infinite Generator

## NOTE

This feature is only available in Tessera Pro

Tessera's generator only builds finite levels. [This utility](#) can be used to lazily generate infinite levels by repeatedly applying the generator.

It works by splitting up the space into a grid of chunks. Then it works out which chunks need to be generated, and fills each one with the output of the configured generator.

## Configuration

`generator` - sets the generator that is used to fill the chunks. It also determines the size of each chunk.

`parallelism` - the number of chunks that can be generated concurrently. Note that turning this up can cause slightly worse quality output.

`watchedColliders` - determines the volume in which chunks should be generated. You typically want to use a large trigger collider following the player or camera, to ensure that everything nearby is generated.

`infiniteX` - You can set each of X, Y and Z to repeat infinitely. If this is false, you can specify the min/max chunk to give a limit to the amount of chunks generated.

`strip` - shrinks the size of each chunk by this many tiles on each side, while keeping the original size for the generator. Increasing this can help avoid contradictions generating chunks. [See here](#).

## Chunk Cleanup

By default, chunks are never removed. This can cause memory usage to grow unboundedly as your game runs. You can enable chunk cleanup to resolve this. The generator will periodically scan for chunks that haven't been near a watched collider recently, and clean them up according to `ChunkCleanupType`:

- None - Chunks are never removed.
- Memoize - Remove the GameObjects, but keep tile data so they can be recreated exactly
- Full - Remove everything associated with the chunk.

Memoize is usually desirable if players can return to a chunk later, otherwise Full is more memory efficient.

In either case, you can set `chunkPersistTime` to control how long chunks stick around for when not needed, and `cleanupInterval` to control how frequently to scan for chunks.

## Caveats

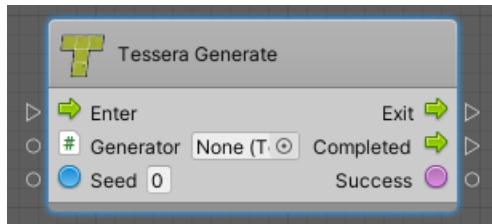
- Only works for square and cube grids
- Only works at runtime
- Generator constraints may not behave intuitively as they only operate on each chunk.
- Scene based constraints are ignored (as scanning for them quickly becomes extremely expensive as the level gets too large).
- Even with a fixed seed, this generation is non-deterministic, as it depends on the order of chunk evaluation.

# Bolt Support

Tessera includes some additional support for Bolt, Unity's visual scripting language.

To enable it, first install Bolt, select `Assets > Import Custom Package...` and select `TesseraBolt.unitypackage` from the Tessera folder. Finally, go to `Tools > Bolt > Build Unit Options` to load the package.

After doing this, a Unit called `Tessera Generate` should be in the fuzzy finder.



This unit will run a particular generator. It works with events run in coroutine mode.

You can also enable Bolt's autogenerated reflection by adding Tessera to the Bolt's assembly list, but this is not recommended as the full API is a bit complicated to use from a visual scripting language.

## Ports

### Generator

The `TesseraGenerator` to run. The default is Self.

### Seed

Fixes the random generation to particular values. The default is 0, which means a new seed each run

### Initial Constraints

This port is only visible if you enable `Set Initial Constraints` in the Unit Inspector. Otherwise, the usual generator behaviour of searching the scene for initial constraints occurs.

A `List<GameObject>` specifying `TesseraTile` and `TesseraVolume` to constraint the generation by. The default is the empty list.

### Exit

Runs immediately after Enter.

### Completed

Runs when the generation is actually completed.

### Success

Set after completion, to indicate if the generation was successful.

# Paint Tile From Mesh

## NOTE

This feature is only available in Tessera Pro

When using [Mesh Edges model type](#), Tessera will automatically infer adjacency information from the mesh of each tile. This can be a real time saver. But it's hard to diagnose visually, and it can only be customized by changing the meshes.

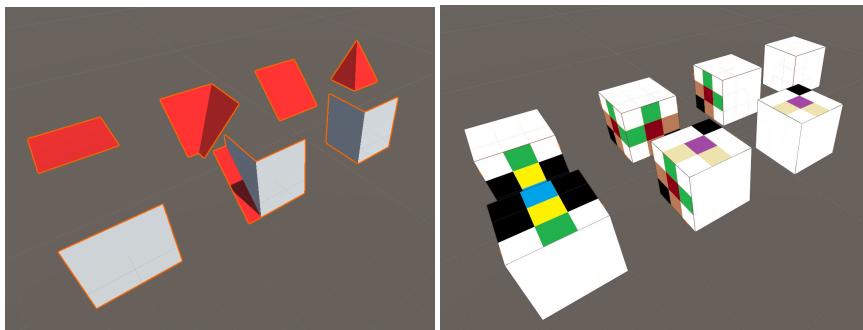
So, as an alternative, you can use the Paint Tile from Mesh Dialog to copy the details of the mesh to tile paint instead. You can then use the [Adjacent Paint model type](#) instead, which is often easier to work with.

This feature is a useful way of visually debugging if your edges are being detected as you expect. Each face is painted with a unique pattern, with the following properties:

- If two faces have the same edges, they'll have the same pattern
- If two faces have mirror-image edges, they'll have mirror-image patterns
- If the edges are symmetric, the pattern will have the same symmetry
- If there are no edges on the face, it'll be a solid color (black/white)
- The corners will always be painted black/white depending on if they are in/out of the mesh (by default)

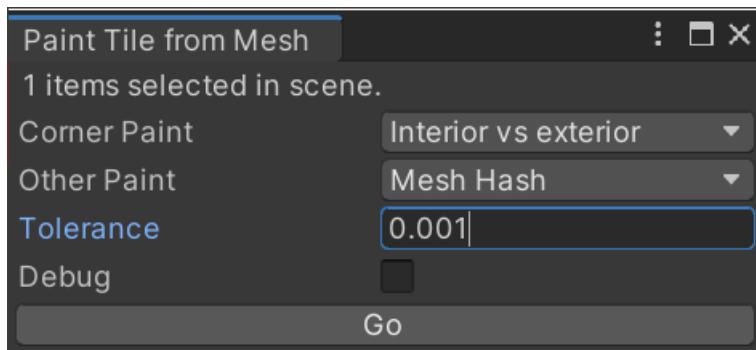
Because the paint is pre-computed, it can be a bit faster than using the Mesh Edges model type, particularly for large meshes.

Here's some tiles painted, from the [Building sample](#). You can see repeated patterns where the tiles form similar shapes.



## Usage

This window is launched from the tools menu, look for `Tools > Tessera > Paint Tile from Mesh...`.



It has the following options:

**X items selected in scene** - Select which tiles you want painted

**Corner Paint** and **Other Paint** - There's several different ways of painting tiles, which can be set separately for corners and other parts of the tile. The different options are [described below](#).

**Tolerance** - Controls how precisely vertices are matched. See [Tolerance](#).

**Debug** - Prints detailed information to the console about exactly which edges were detected.

**Go** - Actually do the painting.

#### Paint Options

- Clear - Removes paint from this area
- Mesh Hash - Paints the face according to what mesh edges lie in that face.
- Interior vs Exterior - Detects the outside and inside of the mesh using vertex normals, and colors either black (interior) or white (exterior). Meshes do not need to be watertight.
- Don't Change - Leaves whatever paint was already present

# Samples

This is a list of all samples that come with Tessera / Tessera Pro, and what they demonstrate.

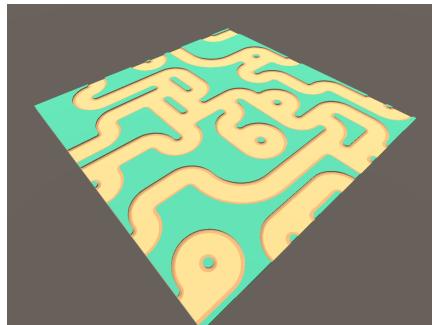
Please note that most samples do not use prefabs as they are more self contained that way. But you are encouraged to make your tiles prefabs in practice.

## GrassPaths

GrassPaths contains a collection of samples that all share a very basic tileset. They demonstrate many Tessera features in as simple a context as possible.

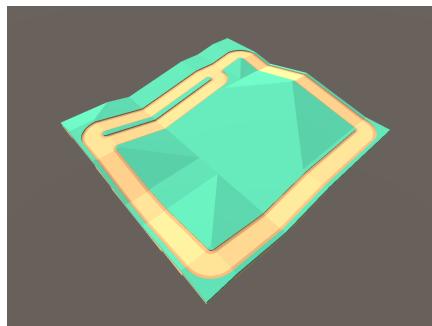
### GrassPaths

Demonstrates configuration of some simple tiles showing a brown path running through a green field of grass. Includes one [big tile](#) that occupies the space of 4 normal tiles. This sample serves the basis for the [getting started](#) and [big tile](#) tutorials.



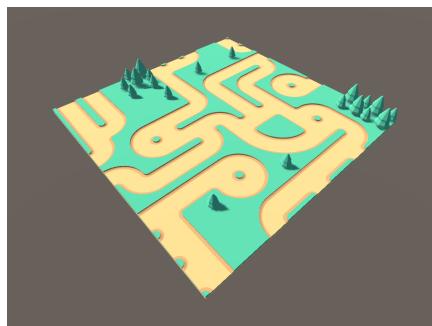
### GrassPaths3d

Adds sloped tiles to GrassPaths.scene, and extends the generator vertically into the 3rd dimension.



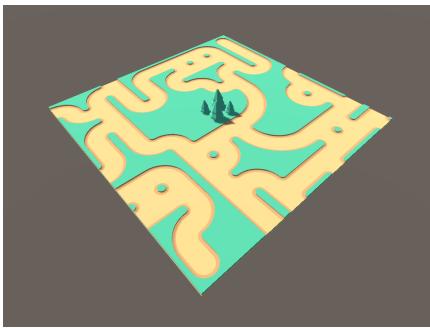
### GrassPathsMultipass

Adds a second generator to GrassPaths.scene, which builds on the first generator by adding vegetation in appropriate places. This is described more in [multipass](#).



### GrassPathsWithPins

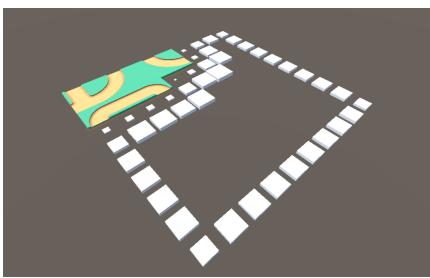
Includes some [pins](#) to GrassPaths.scene, showing how you can fix part of the generation manually.



## GrassPathsAnimated

### Pro only

Adds a [Animated Generator](#) to GrassPaths.scene.

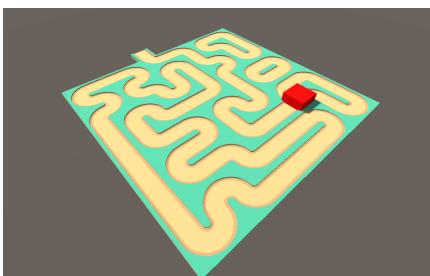


## GrassPathsContradiction

### Pro only

Configures the generator to always fail, so you can see uncertainty tiles in use.

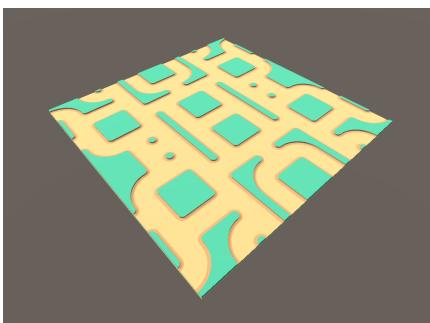
Try fiddling with the `Failure Mode` setting on the generator.



## GrassPathsWithMirror

### Pro only

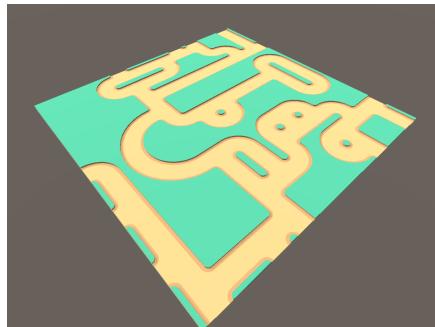
Adds a [mirror constraint](#) to GrassPaths.scene.



## GrassPathsWithPathConstraint

### Pro only

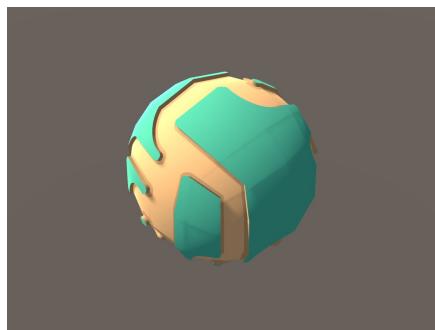
Adds a [path constraint](#) to GrassPaths.scene.



GrassPlanet

**Pro only**

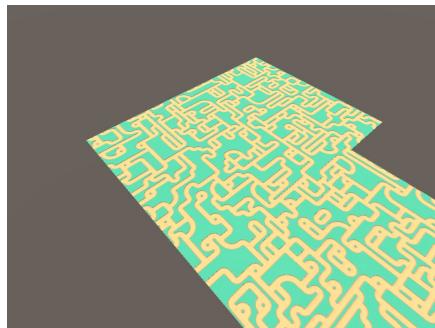
Configures GrassPaths.scene with a [mesh surface](#) to make a spherical planet.



GrassPathsInfinite

**Pro only**

Configures GrassPaths.scene with an [infinite generator](#) that lazy generates tiles near a collider.



GrassPathsOverlapping

**Pro only**

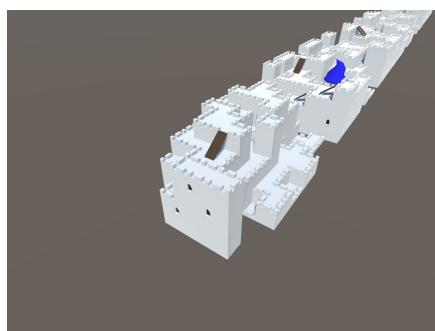
Uses the [overlapping](#) feature to generate an image from a given sample.

Generated

Sample

## Castle

A 3d generation showing multiple useful techniques. The tile paint in this sample is based off of Marching Cubes, a popular way of doing auto-tiling. This sample also comes with a small script showing how Tessera can be used to lazily construct infinite generators. (see also GrassPathsInfinite if you have Tessera Pro).



## Dungeon

A generator that creates Diablo-esque dungeons. Includes a [volume constraint](#) that forces empty space in the center of the generator.



### DungeonWithGeneratorConstraints

#### Pro only

Adds several constraints to Dungeon.scene to encourage the level to be more realistic and playable:

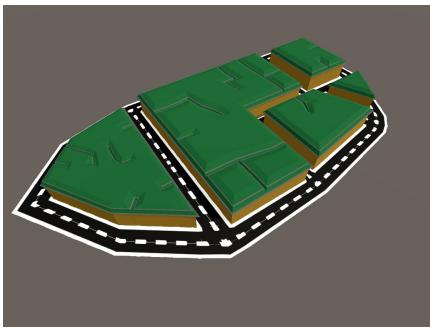
- The level must be fully navigable
- Exactly one staircase up and down.



## City

#### Pro only

Uses a [mesh surface](#) to map out several city blocks, with different mesh materials representing roads and housing.



## Platformer

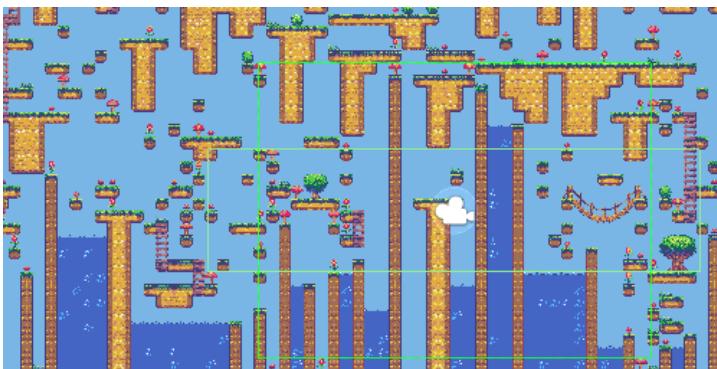
Demonstrates using [square tiles](#) instead of cubes. In Tessera Pro, this sample also demonstrates [writing to a Unity tilemap](#).



## PlatformerInfinite

**Pro only**

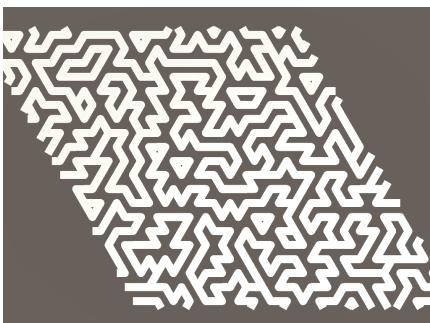
Uses the same tiles as Platformer, but with an [infinite](#) horizontal scroll.



## Hexes

**Pro only**

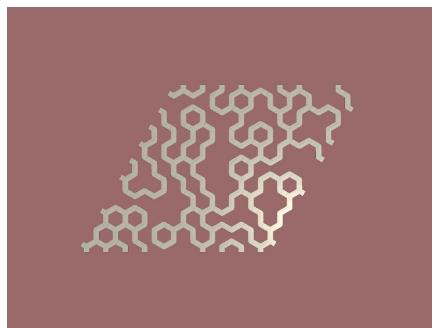
Simples set of tiles demonstrating using [Hex Prism tiles](#).



## Triangles

### Pro only

Simple set of tiles demonstrating using [Triangle Prism tiles](#).



### TriangleSphere

Configures Triangles.scene with a [mesh surface](#) to make a spherical example.

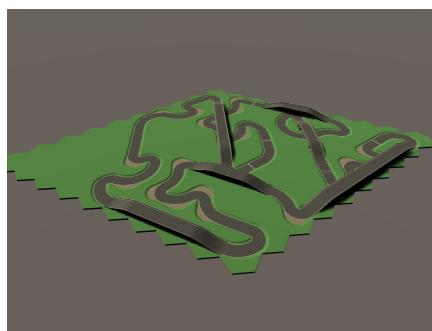


## Hex Raceway

### Pro only

An example with [hex tiles](#) and a [path constraint](#). This is a 3d generator, with height 2, allowing bridges over the track.

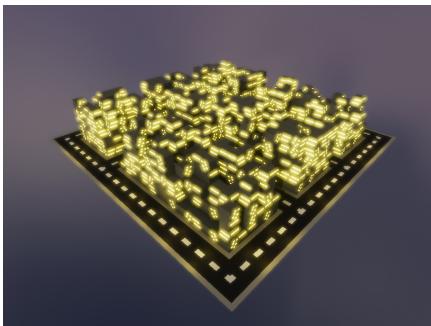
The path constraint has the "Parity" setting on which is specially designed for pathways with no forks, such as this circular track.



## Skyscrapers

### Pro only

An advanced multipass example. The road network is generated in the first pass, and the skyscrapers generated as the second pass.

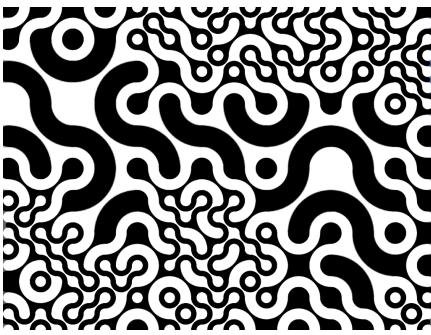


## Truchet

Implements [multi-scale truchet tiles](#), using two different techniques.

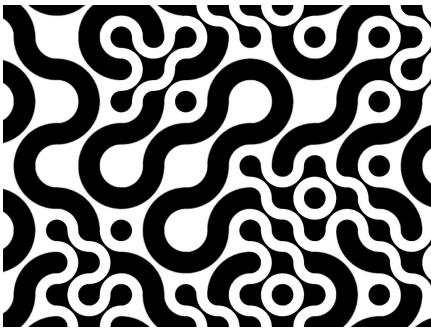
### Truchet Big

Uses [Big Tiles](#) to support multiple scales of tile in one generator.



### Truchet Multipass

Runs two different generators, one for each scale of tile.



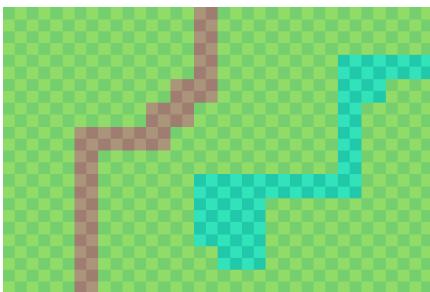
### Plains

#### **Pro only**

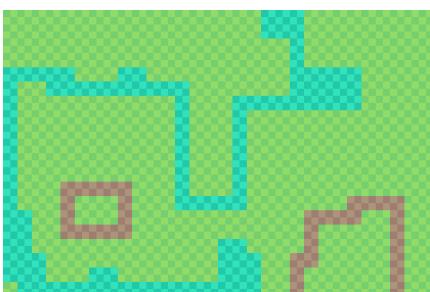
A complicated example using [overlapping model](#) with a Unity Tileset based sample, and multipass.

This shows how Tessera can be used for a simple sort of auto-tiling - the first pass picks the basic terrain type, then the second pass chooses a tile appropriate for the nearby terrain.

### Sample



Pass1



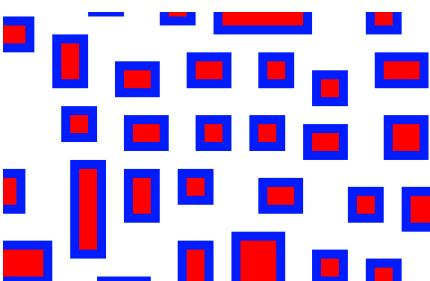
Pass 2



SimpleOverlapping

**Pro only**

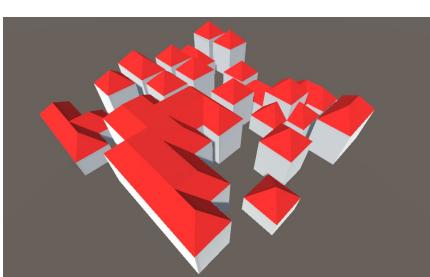
A basic overlapping example, paired with an Infinite Generator. This uses the `strip` option to work reliably.



Building

**Pro only**

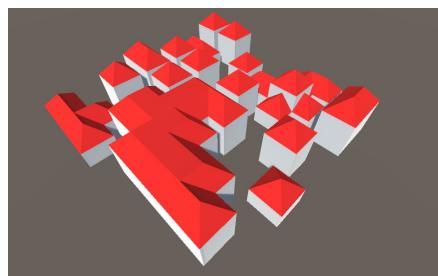
Demonstrates using [Mesh Edges Model Type](#) to connect tiles instead of painting them.



## BuildingPaintFromMesh

### Pro only

Same as the previous sample, but uses the [Paint Tile From Mesh](#) feature instead of Mesh Edges model type.



# Palettes

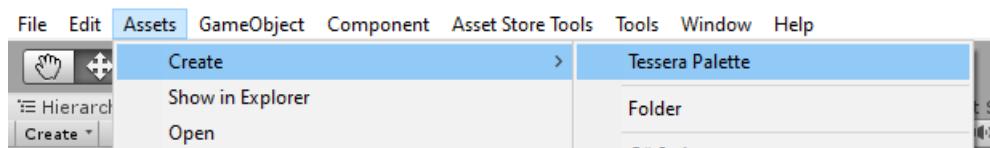
Palettes are an asset type for Tessera that lets you customize which colors you can paint onto tiles, and also controls how colors match.

By default, Tessera comes with a palette of 18 colors (plus transparent). Each color has a short name to help you identify it.

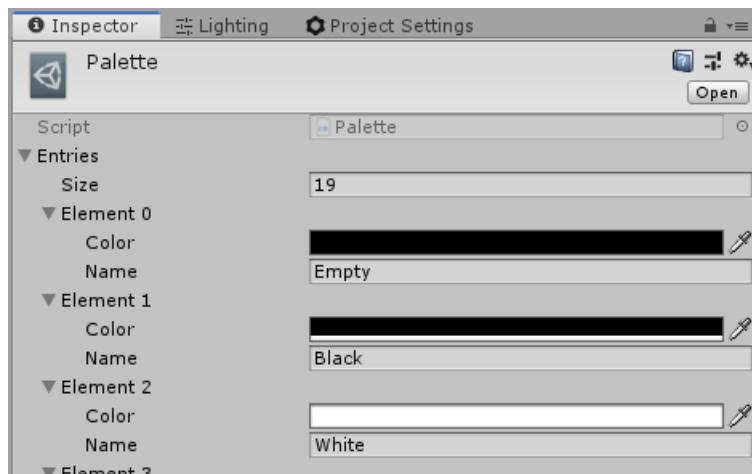
The default palette is configured so that two opposing squares "match" if the colors of those squares are the same, or one is transparent. Recall that two tiles can be placed next to each other if all 9 squares on the face of one tile match with the 9 squares of the opposing face.

## Creating a palette

To customize the palette, create a new Tessera Palette asset from the Assets menus.



This will create a new asset in your project. You can then select the asset and customize the colors in the inspector.



To use the palette, select your tiles and assign the new asset to the palette field.

## Customizing the matching rules

Near the bottom of the inspector, you can see an grid of checkboxes.

Matches															
Black	Black	White	Grey	Brown	Red	Pink	Orange	Yellow	Green	Cyan	Blue	Magenta	Purple	Grey	Black
Black	Black	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
White	Black	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Grey	Black	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Brown	Black	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Red	Black	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Pink	Black	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Orange	Black	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Yellow	Black	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Green	Black	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Cyan	Black	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Blue	Black	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Magenta	Black	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Purple	Black	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Grey	White	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Black	White	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
White	White	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Grey	White	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Brown	White	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Red	White	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Pink	White	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Orange	White	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Yellow	White	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Green	White	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Cyan	White	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Blue	White	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Magenta	White	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Purple	White	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Grey	Grey	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Black	Grey	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
White	Grey	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Grey	Grey	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Brown	Grey	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Red	Grey	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Pink	Grey	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Orange	Grey	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Yellow	Grey	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Green	Grey	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Cyan	Grey	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Blue	Grey	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Magenta	Grey	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Purple	Grey	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Grey	Air	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Black	Air	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
White	Air	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Grey	Water	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Black	Water	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
White	Water	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Grey	Water	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Brown	Water	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Red	Water	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Pink	Water	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Orange	Water	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Yellow	Water	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Green	Water	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Cyan	Water	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Blue	Water	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Magenta	Water	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Purple	Water	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

You can click any cell to toggle if that pair of colors matches.

For example, in the image above, black does not match with itself, but does match with white. That means if we had a tile colored black, and another tiled colored white, the generator cannot place two black tiles adjacent to each other, but can otherwise intermix black and white.

If you open the platformer example, you can see an example of this in practise. E.g. walls (black) cannot directly face another wall, but can face onto air (white) or water (blue). I made a second wall color (grey), that works similarly, but lacks the water matching. That allows us to control exactly where water can be placed.

# Cell Types

## NOTE

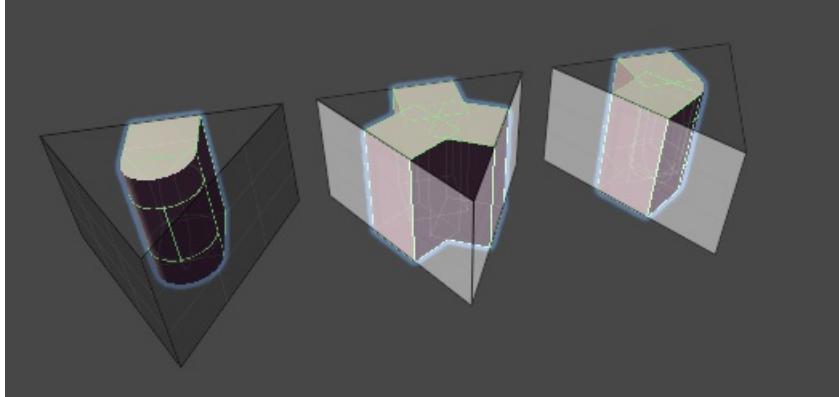
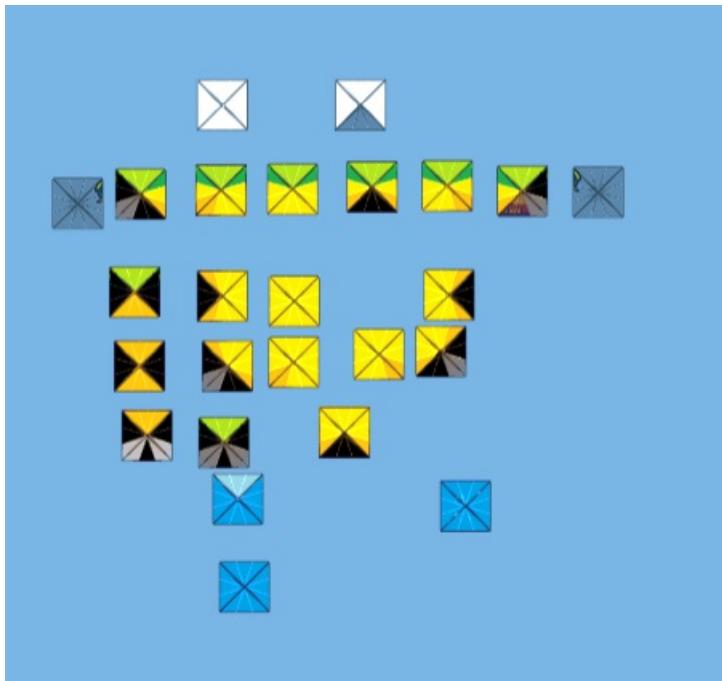
Cell types other than cube and square are only available in Tessera Pro

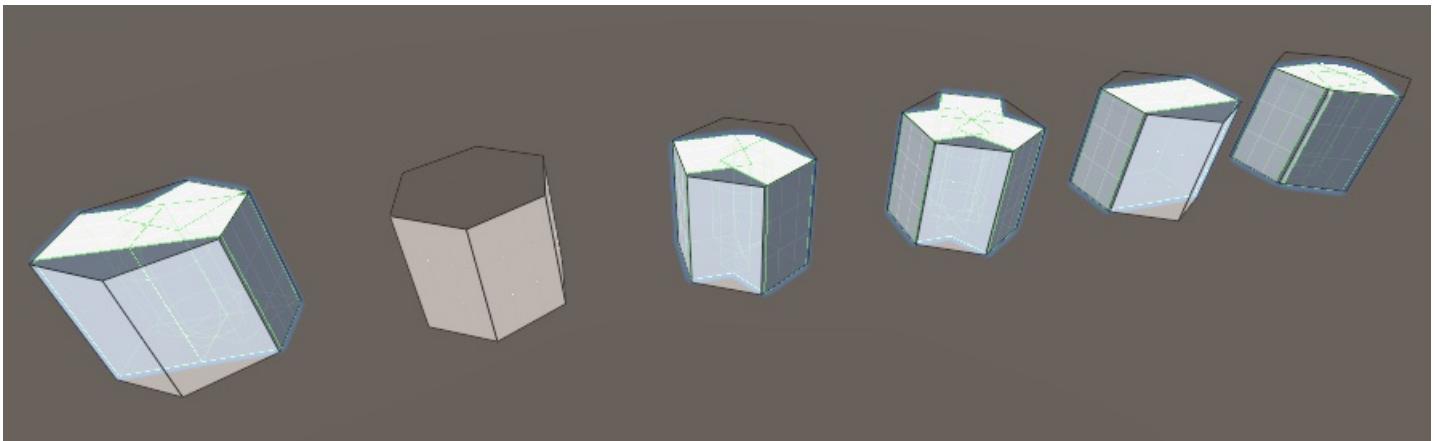
Tessera generally assumes you are working with cube grids, however square, hexagonal and triangular grids are also supported.

To use this, simply use [TesseraSquareTile](#), [TesseraHexTile](#) or [TesseraTrianglePrismTile](#) instead of [TesseraTile](#).

As usual, you can paint the sides of the tiles, and the generator will automatically rotate and place the tiles into cells connecting the sides.

The different tiles in paint mode:





The result of running the generator.



## Caveats

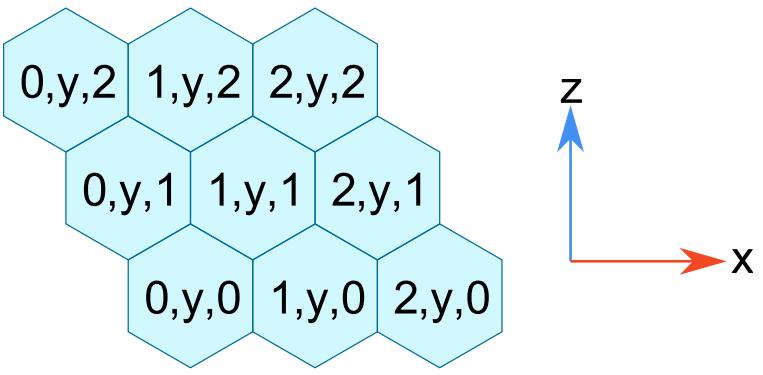
- Only cubes support 3 axes of rotations. Other cell types only rotate in a single plane.
- Triangles generally need rotations enabled, otherwise you must make separate "points up" and "points down" tile variants.
- Different cell types cannot be mixed in a given generator.

## Co-ordinates and measurements

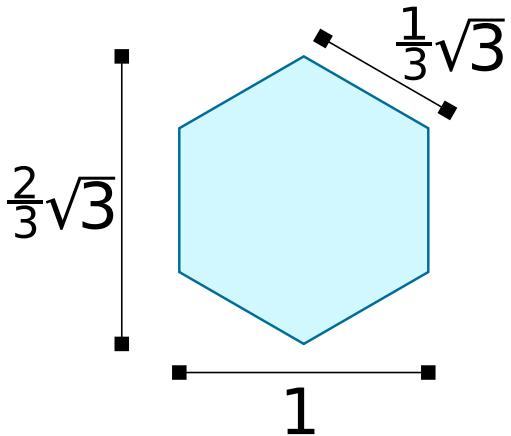
### Hexes

Hexes are laid out in the XZ plane. The Y-axis is used to stack hexes vertically. Hexes are pointy topped, not flat topped. If you need other axes or styles, simply rotate the generator appropriately.

Each hex is referred to using the following co-ordinate scheme:



For a hex of size one, you may find the following measurements helpful.

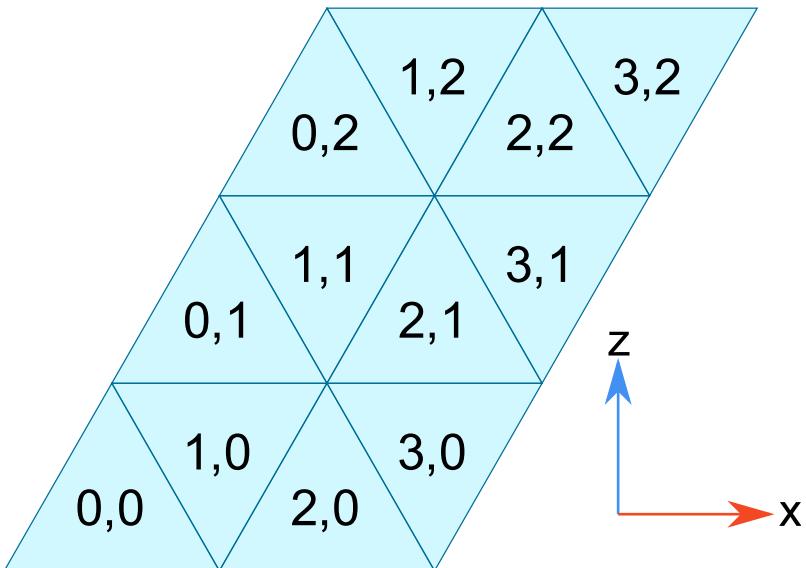


The distance between any two adjacent hex centers is 1.

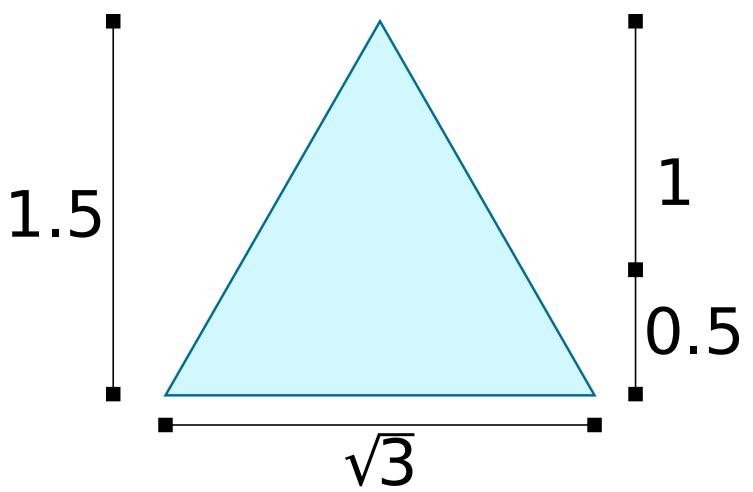
### Triangles

Triangles are laid out in the XZ plane. The Y-axis is used to stack triangles vertically. Triangles point up/down, not left right. If you need other axes or styles, simply rotate the generator appropriately.

Each triangle is referred to using the following co-ordinate scheme:



For a triangle of size one, you may find the following measurements helpful.



The distance between any two adjacent triangle centers is 1.

# Adjacent Paint

This is the default

# Overlapping Model Type

## NOTE

This feature is only available in Tessera Pro

Tessera's main generation algorithm is called WaveFunctionCollapse (WFC), which is responsible for choosing which tiles can be placed in the output.

WFC can be run in two different models: adjacent (aka "simple") and overlapping. By default, Tessera uses the adjacent model. This model forces that pairs of adjacent tiles must connect together, as specified by the tile paint.

In an overlapping model, you supply **samples**, which are pre-made levels that the model will learn from. The generator will then ensure that every rectangle in the output matches a rectangle found in one of the sample levels. This can copy the style of the sample level quite faithfully. More details on how it works can be found at <https://github.com/mxgn/WaveFunctionCollapse> or <https://twitter.com/exppad/status/1267045331004469248>.

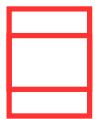
## Example

On the left, a small 6x7 map of tiles has been specified as the sample, and a larger map has been made from it. Every 2x2 rectangle in the output comes from somewhere in the input, with the red box showing one particular patch.

## Generated



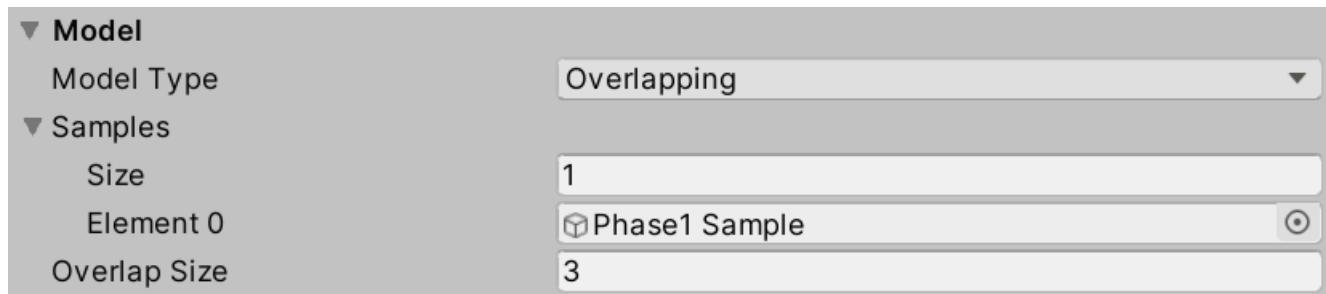
## Sample



## Usage

In the generator settings, set the **Model Type** to **Overlapping**.

You can then supply a list of samples, and the overlap size.



## Creating samples

To create a sample, create an empty game object and add it to the [samples](#) array of the generator.

Then add objects as children of the sample. The children objects should be arranged in a grid with spacing matching the tile size. The children don't need a TesseraTile component, they need to be "linked" to the original tiles instead. Tile are linked by either:

- The name of the child matching the name of a configured tile (suffixes are ignored)
- The child having a [TesseraPinned](#) component, with the tile property set to the tile in question.

Using grid alignment tools such as [ProBuilder](#) can help set up your sample.

For 2d grids, you can also use a Unity TileMap as the sample input. The tiles should have names that correspond with configured Tessera Tile objects. See the "Plains" sample for details.

### Clone Sample

Getting Tessera to detect your sample can be a bit finickity. Use the `Clone sample` button that appears in the inspector to get Tessera to dump out its internal view of the sample. If you've got everything right, it should look like the original. If not, see what's different.

### Rotation

If the tiles have rotation / reflection on, then reflected / rotated copies of the sample will also be used. It's highly recommended you enable the [symmetric](#) option if you have rotatable tiles, as overlapping model copes much less well than the adjacent model with redundant tile rotations. This feature requires tile paint.

## Caveats

- Overlapping model is much slower than adjacent models. Using overlapSize greater than 3 is not recommended.
- Overlapping model is a strict constraint - *every* output rectangle must come from a sample. You will often need to make samples quite large and detailed to get sufficient variety for a good quality output.
- It supports cube and square grids only
- You must manually create sample levels.
- Overlapping interacts strangely with irregularly shaped outputs. You are recommended to avoid pins other than PinType.Pin, and to avoid the MaskOut feature of Tessera volumes.
- Tile paint is not used for the model, but it will still be read by many other Tessera features such as rotational symmetry, pinned tiles, the path constraint and skyboxes. Thus you will likely need to paint the tiles in addition to making samples.

# Adjacent Model Type

## NOTE

This feature is only available in Tessera Pro

The Adjacent Model works the same as [overlapping model](#), only the samples are analysed for pairwise adjacencies between tiles instead of small patches.

# Mesh Edges Model Type

## NOTE

This feature is only available in Tessera Pro

The usual way of working with Tessera is that you manually paint the connections between tiles. This gives you ultimate control over behaviour.

However, Tessera also supports auto-detecting tile connections based on the mesh of each tile. Two tiles connect if

- the open edges (aka borders) of their meshes exactly match when placed next to each other.
- the corners interior information agrees

Sample [Building](#) shows a an example of this Model Type in action.

## Setting up the generator

The generator can be set up normally. Set the Model Type to `Mesh Edges`.

## Setting up the tiles

Tiles can be set up normally. There's no need to paint the tiles as it's not used in this model type (though some other Tessera features like the path constraint do use it). This feature works best with Cube tiles, but Hex tiles also work.

Each tile should have a `MeshFilter` component on it or a child object. You don't need a `MeshRenderer` component.

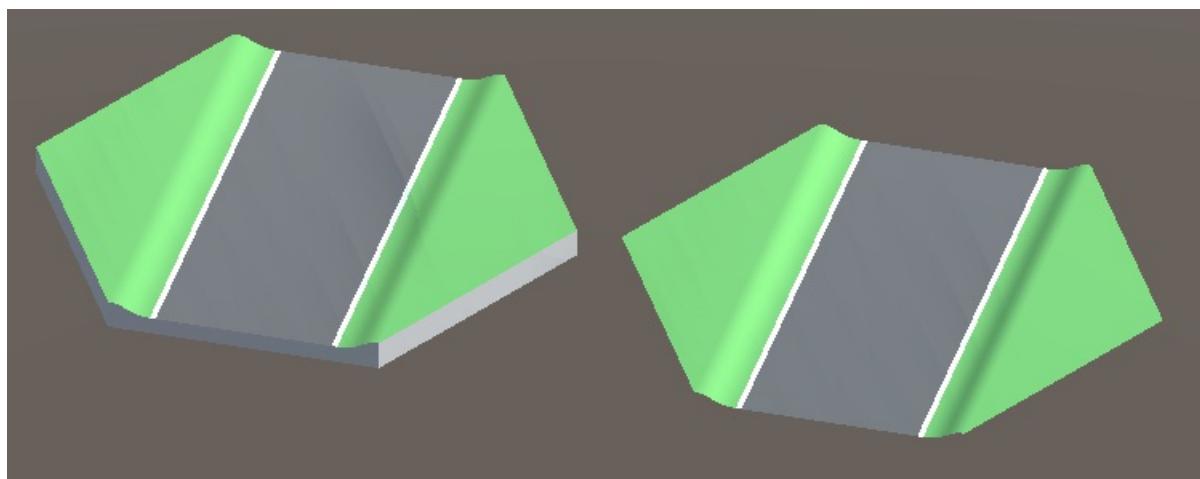
The meshes need to be designed for this in mind. Generally, the meshes should not be a fully closed solid. Instead, each should be a surface width edges terminating at the tile boundary. This will enable Tessera to detect which edges are on each side of the tile.

Also, Tessera will not match a single edge in one mesh with two edges on a second mesh, even if they cover the same distance. So it's necessary to make sure all tiles have the same vertices and subdivision.

You can see some suitable examples in the [Building](#) and [HexRaceway](#) samples.

Typically, asset packs give tiles that are fully closed. It's necessary to delete edges and size the mesh carefully before it will work in Tessera.

Left shows the [original mesh](#), right [the mesh with extra faces](#) deleted so that the mesh edges will line up with those from another tile..



If you are still in doubt, [WildTile](#) has a good tutorial about how the meshes should work

## Empty tiles

When a Tessera Generator runs, it always fills every cell of the grid with a tile. Thus it's often necessary to explicitly supply an empty or "air" tile for Tessera to use when you don't want anything else added.

You also may find the need to create a "solid" tile which is used to fill in the interiors or builds, or underground, etc.

Edge detectors can often get confused between these two tiles - they fulfill different roles and have different connections, but both of them don't need any mesh as there is nothing to display.

In order to resolve this, Tessera will treat a tile with an **empty mesh** (no vertices) as all interior (solid) and a tile with **no mesh** (null, or without a `MeshFilter` component at all) as all exterior (air). Tessera comes with an `EmptyMesh.fbx` asset for use in the first case.

## Debugging

Mesh edge matching is not a 100% reliable process. Often, some edges won't match, because you've not desinged your tiles correctly, or something has gone wrong.

To debug issues, you are recommended to use the [Paint Tile From Mesh](#) feature. It gives a clear visual indication of how the mesh has been interpreted. It also comes with a Debug setting, which dumps out exactly what Tessera found when analysing the mesh. For even more debug info, set `EdgeDetection.ExtendedDebugInfo` to true in the source code, which gives internal details about the snapping and hashing process.

## Tolerance

Tessera classifies edge by snapping vertices to a grid, then forming a "hash" that matches pairs of edges.

The tolerance setting controls the level of snapping. A small value means vertices have to be incredibly close to each other to match, which can often cause similar looking meshes to not match. A large value means vertices do not need to be as close, which can cause different meshes to accidentally match.

If you have having issues with the matching process, try tweaking the tolerance. Sometimes picking non-round numbers helps too, as this can stop similar vertices being snapped in different directions. Try 1/252 for example.

## Samples

The [Building](#) and [HexRaceway](#) samples both come with tilesets suitable for edge detection. Feel free to experiment.

# Controlling Output

## NOTE

Output control is only available in Tessera Pro

By default, after doing generation, `TesseraGenerator` will instantiate copies of all the tiles as child objects.

This is usually what you need, but it is possible to customize it further. To do this, you need to attach one of these behaviours to the generator:

- [TesseraInstantiateOutput](#) - Finer control over what gameobjects/prefabs are instantiated
- [TesseraMeshOutput](#) - Merge all the tiles into a single mesh. Often much faster than creating each tile separately.
- [TesseraTilemapOutput](#) - Write the output into a Unity Tilemap.

## Handling the output in code

If none of the built in behaviours above are appropriate, you can handle output yourself.

When invoking the `Generate` method, you can set `onComplete` or `onComplete` to completely replace the the default behaviour with your own code. There's an example in the [API docs](#).

# Creating instances

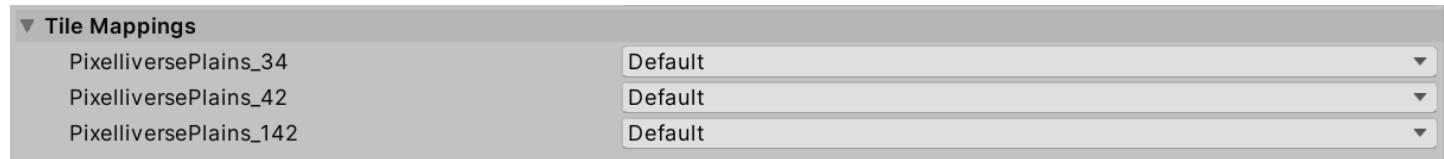
By default Tessera will instantiate each tile as a child of the generator. If you want to customize this, you can use a [TesseraInstantiateOutput](#).

When using TesseraInstantiateOutput, instead of creating a copy of the tile itself, you can specify a different prefab to be instantiated instead.

See the [Plains Sample](#) for an example of this out in action.

## Tile Mappings

Each tile in the generator can be separately configured in the editor. There are several different options.



### Default

The default setting simply instantiates a copy of the Tile object.

### Nothing

No output will be made for this tile.

### Instantiate As

With this setting, you can specify a different prefab to instantiate. By using "Instantiate Children Only" you can create multiple objects for one tile, like the [equivalent setting on the Tile components](#).

### Constraint

Specify a [scene constraint](#) like a Pin or Volume. A copy of that constraint will be instantiated in the correct position. For volume constraints, the newly created object will be configured to exactly cover the space of the original tile.

### Function

This fires a Unity event when this tile needs instantiating. The event function should have three arguments:

```
void MyEvent(TesseraCompletion completion, TesseraTileInstance instance, Action<GameObject> results)
```

The first two arguments are documented in the [API](#). The third argument can be optionally used to record any game objects that you created during the event (used for undo and animation).

# MeshOutput

## NOTE

`TesseraMeshOutput` is only available in Tessera Pro

You can use `TesseraMeshOutput` to write directly to a mesh. Your tiles objects must have a MeshFilter and MeshRenderer, unless you check Instantiate Children Only, in which case the children must have a MeshFilter and Mesh Renderer.

## Materials

Because Tessera is merging multiple tiles, the resulting mesh will often need multiple materials. There are different options for how to handle this.

- **Single** - Ignore the materials set on the tiles, and just use the specified material instead
- **Unique** - Use every material found on any tile.
- **UniqueByName** - Use every material found on any tile, deduplicating materials if they share the same name.

## Colliders

Mesh output has 3 settings for handling colliders:

- **None** - No colliders are output
- **ReuseRenderMesh** - A mesh collider is set up, which uses the same mesh as the MeshFilter.
- **Merge** - The meshes of all the mesh colliders of the tiles are merged together.

## Other components

Because mesh output does not create unity game objects for each tile, all other behaviours are lost. You can use the [C# API](#) to add them back if needed.

## Chunking

By default, mesh output writes to a single MeshFilter on the generator itself. This can get awkward for large generations. By enabling chunking, the tiles will be split into small groups, and a separate MeshFilter/MeshRenderer created for each.

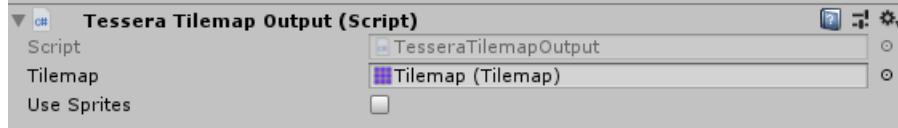
# Tilemap Output

## NOTE

`TesseraTilemapOutput` is only available in Tessera Pro

Unity comes with a `Tilemap` component that lets you store sprites and components in a regular grid.

To enable this, select the game object with the `TesseraGenerator` component, and add the `TesseraTilemapOutput` component. Then set the `Tilemap` property to the tilemap component. Then instead of instantiating objects, it will find the appropriate for cell fo the tilemap, and fill that in instead.



## NOTE

You must ensure that the grid spacing of the Tilemap and of the generator are aligned.

Tessera comes with a sample called `Platformer` that demonstrates writing to Tilemaps.

If you check the `Use Sprites` property, then Tessera will attempt to detect game objects that contain a sprite, and write the sprite directly to the tilemap. This is considerably more efficient than inserting the entire game object into the tilemap, but you lose any other components.

# Quality, Performance and Debugging

Tessera uses [Wave Function Collapse](#) as its core algorithm, though with many custom enhancements. This is a constraint solving based approach to generation, and it is very powerful, but it comes with some limitations. The algorithm searches through the space of possible tile choices to find one that fits all the constraints and adjacency rules required.

If there's no possible solution, Tessera will fail, reporting a contradiction. And if there's only a few possible solutions there are, the more Tessera will struggle to find one. [Tile adjacency](#) is Turing complete, so there are inputs that will just take forever to generate.

This page gives some advice on how to get the most out of Tessera.

## Backtracking

Backtracking is a critical feature of most constraint solvers, though normally WFC doesn't feature it. With backtracking off, the generator will give up the moment it encounters any difficulty. With it on, it will instead undo the last tile generated, and try again with a different option.

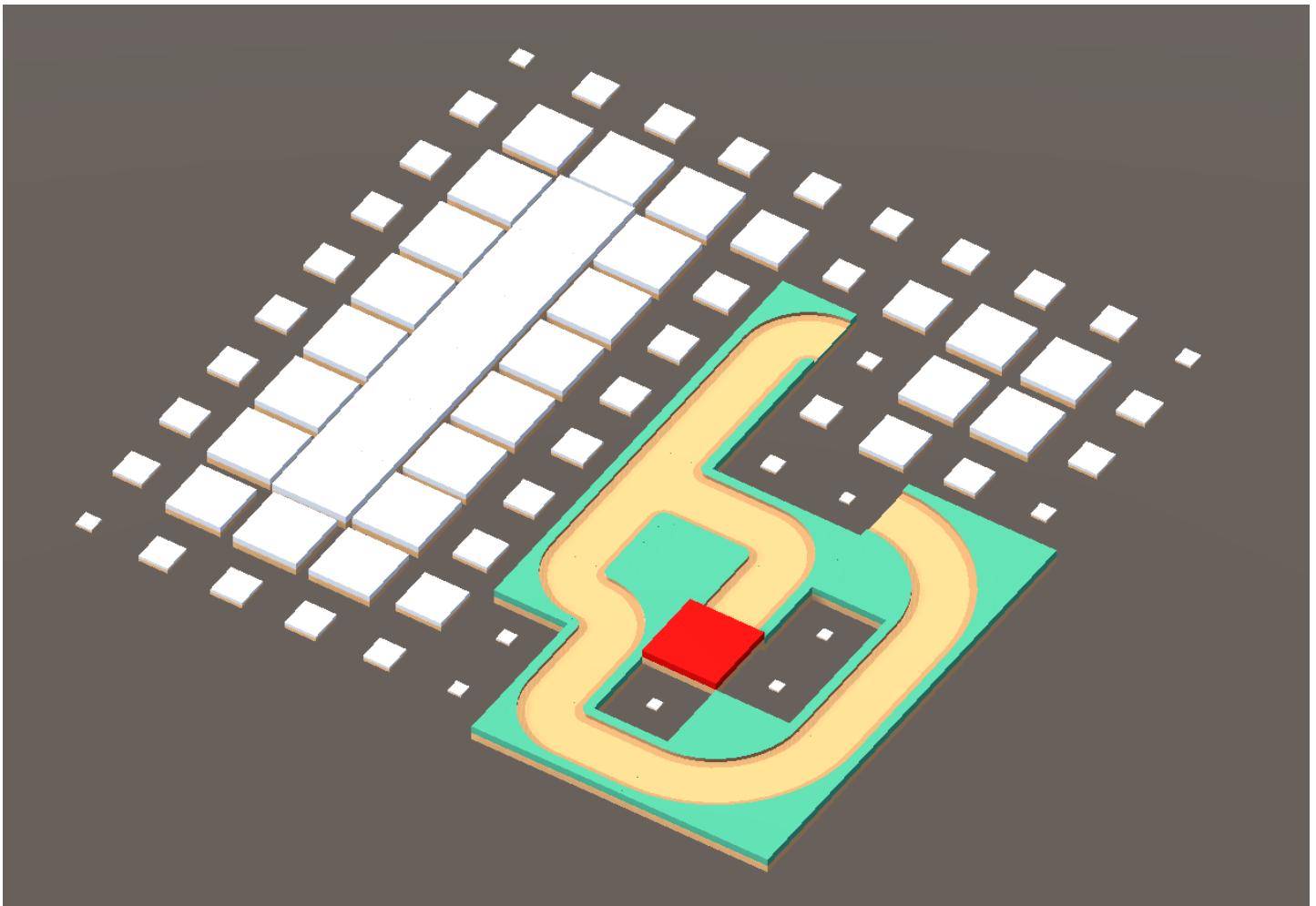
Enabling backtracking can therefore make the solver work much harder at finding a solution. The downside is that it can run *forever* looking for a solution, and can sometimes get stuck exploring possibilities that will never work.

To get the best of both worlds, you can introduce a [Step Limit](#). This will allow some backtracking to occur, but after a fixed amount of computation it will automatically retry with a fresh generation.

## Errors and contradictions

Not all setups for the generator have a solution at all. When the generator fails to find a solution, it'll log a warning detailing what has gone wrong.

To enable more diagnostics, set [Failure Mode](#) to Last, and add prefabs to Uncertainty Tile / Contradiction Tile. This will cause the generator to output a clearer picture of the problem.



The above picture shows a failed generation, with white tiles indicating what has yet to be generated, and the red tile indicating where the contradiction occurred. You will have to infer what the specific problem is. In the above case, the generator wasn't set up with a dead end tile, so there's no way for the path to complete.

Here are some other tips to help determine what is going wrong:

- You can use a fixed [seed](#) to make any issues reproducible.
- You can select uncertainty tiles in the Inspector, they have some additional information attached.
- Try disabling the constraints (both initial constraints and generator constraints) until things start working.
- If you have no constraints, try removing tiles that are difficult to fit in, such as Big Tiles.
- Conversely, sometimes adding tiles can help. You can try adding a tile without any faces painted and with very low weight, which means it can be placed anywhere in the generator as a last resort.
- (Pro only) Use [AnimatedGenerator](#) to display the generation in realtime. This can be useful when backtracking is on, highlighting areas that cannot be successfully completed.

## Performance Tips

If you have very large levels, or a particularly hard to generate setup, then you may find Tessera's performance starts to become a problem. Here are some suggests to combat that.

If all else fails, ask on Discord. I'm making performance enhancements all the time.

### Disable "Record Undo"

This option is on by default, and affects TesseraGenerator when you run it by clicking the "Generate" button in the Inspector panel. Recording undo information is painfully slow in Unity, and will cause the Editor to freeze temporarily while creating all the generator tiles. This setting has no effect at runtime.

### Change generation algorithm.

Tessera's default [constraint resolution algorithm](#) is called "AC4", which resolves all constraints every step. You can instead try "OneStep", which only resolves nearby constraints. It can be faster in some circumstances, but also has less lookahead so is more likely to fail or backtrack.

### Reduce the number of tiles

Doubling the number of tiles makes Tessera approximately 4x as slow, so it's worthwhile minimizing tiles. The easiest way of doing so is enabling the [symmetric](#) setting and avoiding similar/duplicate tiles.

### Path constraints

The [Path constraint](#) is very expensive compared with the rest of Tessera. Avoid if possible, for example, by pre-generating your path and using [pins](#) to make Tessera use it.

The path constraint does come with two experimental settings, [prioritize](#) and [parity](#) that improve search quality, so you may want to try enabling those.

# FAQ

Can I generate with Tessera?

Tessera uses the WaveFunctionCollapse algorithm to generate things. This is a very flexible system, but it cannot do everything.

- Tessera is tile based. This means it is worse at naturally shaped or curving things.
- Tiles are generated based mostly of local constraints. That makes generations very uniform when you have a larger output size. So Tessera is not so good at large scale features, like you'd see in a map generator.

If in doubt, take a look at [the list of samples](#) for some ideas what Tessera can do.

Why is my generation failing?

The nature of the WaveFunctionCollapse algorithm makes this very hard to answer in general.

The [Errors and Contradiction](#) section has some tips.

Why is Tessera slow when run in the Editor?

Turn off "Record Undo". More [performance tips here](#).

I'm getting weird compilation errors

Try deleting the Tessera folder, and re-installing from the asset store.

Does Tessera support editing maps, like Townscaper?

Not at present, sorry! Consider buying [Wildtile](#) instead (no affiliation).

How do I have make a hexagonal grid of hexagons?

The generator always makes a rhombus. make a rhombus larger than your desired size, then use [volumes](#) to cut the grid down to your ideal shape.

# Using the API

The other tutorials have shown you how to set up a tiles and a generator, without any coding. But now you need to hook up the generation to the rest of your level.

## Starting the generator

The easiest way to start working with Tessera is to press the Generate button on the inspector.

However, when it comes to running it in a game, you'll need to actually launch it programmatically. There's two methods for this, [Generate](#) and [StartGenerate](#).

[Generate](#) will start the generation, wait for it to complete, and return some details about the result. This is easy to use, but can cause Unity to freeze briefly while the generation runs.

```
using UnityEngine;
using Tessera;
using System.Collections;

public class MyBehaviour : MonoBehaviour
{
    private TesseraGenerator generator;

    void Start()
    {
        generator = GetComponent<TesseraGenerator>();
        var completion = generator.Generate();
        Debug.Log(completion.success);
    }
}
```

[StartGenerate](#) instead runs the generation asynchronously (if the platform supports it). It is intended for use in coroutines.

```
using UnityEngine;
using Tessera;
using System.Collections;

public class MyBehaviour : MonoBehaviour
{
    private TesseraGenerator generator;

    void Start()
    {
        StartCoroutine(MyCoro());
    }

    public IEnumerator MyCoro()
    {
        // Start the generation
        var run = generator.StartGenerate();
        // Waits for it to run
        yield return run;
        // Get the details of the run.
        var completion = run.Result;
        Debug.Log(completion.success);
    }
}
```

To do so, you need to call or [StartGenerate](#). [Generate](#) will run synchronously, and return details about the generation. It's easier to use, but can cause noticeable stutter if you are doing a big generation. [StartGenerate](#) behaves exactly the same, but can be used from a Unity coroutine.

Because co-routines cannot return information, you can instead supply various callbacks using [TesseraGenerateOptions](#). Most commonly, you'll want to set [onCreate](#) to replace the behaviour for instantiating new tiles. The default behaviour instantiates them all at once, which can cause stutter.

```
using UnityEngine;
using Tessera;
using System.Collections;

public class MyBehaviour : MonoBehaviour
{
    private TesseraGenerator generator;

    void Start()
    {
        generator = GetComponent<TesseraGenerator>();
        StartCoroutine(MyCoro());
    }

    IEnumerator MyCoro()
    {
        var options = new TesseraGenerateOptions { onCreate = MyCreate };
        yield return generator.StartGenerate(options);
        // Any following code will be run after the generation
    }

    void MyCreate(TesseraTileInstance instance)
    {
        Debug.Log("Creating " + instance.Tile.gameObject.name);
        // Do the default behaviour
        TesseraGenerator.Instantiate(instance, generator.transform);
    }
}
```

## Generation options

[TesseraGenerateOptions](#) lets you customize the generation in much greater detail than the options in the inspector.

### onCreate

A callback called for each newly generated tile. By default, is used.

This can be used to customize the output of Tessera (in addition to the [output settings](#)).

### onComplete

A callback called when the generation is complete. By default, checks for success then invokes [onCreate](#) on each instance.

This can be used to complete replace the behaviour of Tessera after a run has completed.

### progress

A callback called with a string describing the current phase of the calculations, and the progress from 0 to 1. Progress can move backwards for retries or backtracing.

#### NOTE

[progress](#) can be called from threads other than the main thread.

### cancellationToken

Allows interruption of the calculations

## seed

Fixes the seed for random number generator. By default, random numbers from Unity.Random.

## multithreaded

If set, then generation is offloaded to another thread stopping Unity from freezing. Requires you to use StartGenerate in a coroutine. Multithreaded is ignored in the WebGL player, as it doesn't support threads.

## initialConstraints

If set, overrides TesseraGenerator.initialConstraints and TesseraGenerator.searchInitialConstraints.

This setting is discussed [further below](#).

## onComplete example

Here's an example of overriding `onComplete`, to so we can create the tiles one at a time rather than all at once:

```

using UnityEngine;
using Tessera;
using System.Collections;
using System.Collections.Generic;

public class MyBehaviour : MonoBehaviour
{
    private TesseraGenerator generator;

    void Start()
    {
        generator = GetComponent<TesseraGenerator>();
        StartCoroutine(MyCoro());
    }

    IEnumerator MyCoro()
    {
        IList<TesseraTileInstance> instances = null;
        var options = new TesseraGenerateOptions
        {
            onComplete = completion =>
            {
                if(completion.success)
                {
                    // Store the tile instances for instantiation later.
                    instances = completion.tileInstances;
                }
            }
        };
        yield return generator.StartGenerate(options);

        if(instances != null)
        {
            // Loop over every object that we want to create.
            foreach(var instance in instances)
            {
                TesseraGenerator.Instantiate(instance, generator.transform);
                // Wait for next frame.
                yield return null;
            }
        }
    }
}

```

## Initial constraints

By default, Tessera scans the scene for pins, placed tiles, and volumes, which serve as constraints on generation. Collectively, these are known as "initial constraints". They are described more [here](#).

Scanning the scene for objects, can be quite slow and inconvenient, so Tessera has an API specifically for setting up initial constraints in the generator without instantiating anything.

To use it, create an builder using [TesseraGenerator.GetInitialConstraintBuilder](#).

The builder then has many methods on it for creating [ITesseraInitialConstraint](#) objects. Store as many of these as you like in a list, then pass that list into the generator options (described above) when running the generation.

Example:

```
var builder = generator.GetInitialConstraintBuilder();
var initialConstraints = new List<ITesseraInitialConstraint>();
// Pins a tile at a given position.
// The tile doesn't need to be in the scene, it can be a prefab.
initialConstraints.Add(builder.GetInitialConstraint(tile, localToWorldMatrix));

// Set up and run the generation.
var generateOptions = new TesseraGenerateOptions();
generateOptions.initialConstraints = initialConstraints;
generator.Generate(generateOptions);
```

## TesseraCompletion

After a run is finished, a [TesseraCompletion](#) object is returned.

This gives a complete description of what has been generated. It's also used in the [onComplete](#) callback.

TesseraCompletion has the following properties.

[success](#)

True if all tiles were successfully found.

[tileInstances](#)

The list of tiles to create.

Big tiles will be listed a single time, and each [TileInstance](#) has the world position set.

[tileData](#)

The raw tile data, describing which tile is located in each cell.

[retries](#)

The number of times the generation process was restarted.

[backtrackCount](#)

The number of times the generation process backtracked.

[contradictionLocation](#)

If success is false, indicates where the generation failed.

[isIncremental](#)

Indicates these instances should be added to the previous set of instances.

[grid](#)

Describes the geometry and layout of the cells. See separate [Sylves documentations](#) for more details.

[gridTransform](#)

The position of the grid in world space. (Sylves grids always operate in local space).

## GetGrid

The grid that Tessera uses is available via [GetGrid](#). This gives a [Sylves.IGrid](#) object lets you easily work out the position of each cell, convert positions to cells, and many other operations. The grid is also available in the [TesseraCompletion](#) object.

Tessera grids are handled by [Sylves](#), an open source library. You can refer to [its own docs](#) for how to use it.

**NOTE**

In many places, Tessera uses Vector3Int, which needs to be explicitly cast to a Sylves.Cell when working with Sylves' API.

**NOTE**

If you are using a Mesh Surface based grid, then GetGrid() can be quite slow, and you are recommended to cache the result of it.

# Developing Tessera

Tessera's source is available in Pro versions for you to make changes to, however it's a large and complex project. Feel free to make private details public if you need them for your game, but you are not recommended to attempt major modifications.

If you cannot achieve what you want in the editor, check if it can be done with the [C# API](#) or ask on Discord for support.

That said, this page gives a rough overview of how the project internals works, if you want to understand in further details.

## Sylves and DeBroglie

Much of the core logic of Tessera is open source!

[DeBroglie](#) is the library that handles the core WaveFunctionCollapse algorithm and constraint solving.

[Sylves](#) handles the grid logic, and is how Tessera generalizes to several sorts of grids.

These libraries are separately documented and you are recommended to familiarize yourself with them before dealing with Tessera code.

Tessera supplies these libraries as pre-built dlls, but you can find full source on [github](#).

## Translating to DeBroglie

Most of Tessera is translating the Unity configuration into terms that DeBroglie can understand, running DeBroglie, then translating back.

This is mostly the responsibility of [TesseraGeneratorHelper](#). It creates a DeBroglie TilePropagator, and sets it up with all the details it needs:

- A tile model describing the tessera tiles and how they connect.
- A topology describing the grid to output to.
- Tessera constraints are replaced with DeBroglie constraints
- Various settings get passed through to DeBroglie

### Concepts

#### ModelTiles

DeBroglie requires that tiles are one-to-one with the cells that they are in, and that there is a unique tile for every rotation. Tessera tiles can cover multiple cells, and support multiple rotations. Therefore, every Tessera tile is cut up into multiple smaller tiles for DeBroglie, called a "model tile". Model tiles have 3 pieces of information:

- Tile - the tessera tile that this model tile subdivides
- Rotation - the specific [rotation](#) of the tile file.
- Offset - which subdivision of the tile this model tile refers to (see below)

After DeBroglie is run, ModelTiles are merged together and converted back into a form more convenient for users, [TesseraTileInstance](#).

#### Offsets

Each tile basically has a mini grid associated with it, and `offset` gives the co-ordinate of the cell in that grid corresponding to the subdivision.

You can access this grid with [SylvesCellGrid](#), but for performance reasons, this grid is *not* scaled to match the settings of the tile. Instead, you need to do that translation and scaling every time you access the grid. There are convenience functions for this in `SylvesExtensions.GetCellCenter` and `SylvesExtensions.GetCellSizeTransform`

## Spaces

In Unity, there are the concepts of "world" space and "local" space. These are different co-ordinate systems. It's useful to track which space a vector is associated with, as you can convert vectors between them with the appropriate matrix.

Tessera internally uses a similar terminology, for handling things.

- **World space** - as in unity
- **Generator space** - local space, for the TesseraGenerator object.

The grid of the generator is always specified in local space.

You can transform to worldspace via `generator.localToWorldMatrix`

- **Tile space** - local space for a `TesseraTileBase` object.

You can transform to generator space via the properties of TesseraCellInstance. Note that this is a non-linear transform (a `Sylves.Deformation`) in some cases.

- **Cell space** - sometimes called canonical space, this is the space of a single cell in a tile. It's a translated/scaled version of tile space.

You can convert from to tile space with `SylvesExtensions.GetCellCenter` and `SylvesExtensions.GetCellSizeTransform`

## Execution Overview

What actually happens when you press Generate? Here's a description:

- Copy all relevant Unity data to threadsafe structures, and reorganize (`CreateTesseraGeneratorHelper`)
  - Initialize random seed
  - Validate generator (`Validate`)
  - Analyse the tiles (`TileModelInfoBuilder.Create`)
    - Also analyse samples, for Overlapping models.
    - Also analyse meshes, for MeshEdges models.
  - Get all constraints
    - Submesh filter (`GetSubmeshTilesInitialConstraints`)
    - From scene (`SearchInitialConstraints`)
    - Generator constraints (`GetGeneratorInitialConstraints`, `GetTileConstraints`)
    - Skybox
- Repeat several times until success, possibly run in another thread (`FullRun`)
  - Create DeBroglie Model (`Init`)
  - Create DeBroglie Propagator (`CreatePropagator`)
  - Setup
    - Copy initial constraints into DeBroglie (`ApplyInitialConstraintsAndSkybox`)
    - Additional constraints needed for big tiles (`BanBigTiles`)
  - Run DeBroglie, with some extra tracking (`Run`)
  - Handle failure (`PostProcess`)
- Copy the results of DeBroglie into a TesseraCompletion
- If `onComplete` is defined, call it and exit.
- Log error message
- Pass the completion to the appropriate `ITesseraTileOutput`

## Paint Storage

When you paint a tile in the editor, the data is stored in `sylvesFaceDetails`. This format public data you can tweak and generate, but is a bit difficult to work with.

Essentially, each cell is split into faces. A cube shaped cell has 6 faces, a hex cell has 8 (6 around the sides and top and bottom). Big tiles can have even more faces. A face is uniquely identified by an offset ([see above](#)) and a direction (Sylves.CellDir).

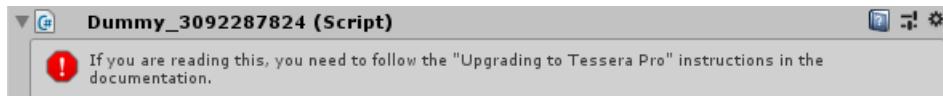
Do not add or remove faces, instead use methods AddOffset/RemoveOffset, which will handle it for you.

Each faces have several `int` fields each of which corresponds to a "subface". Subfaces are the elements that you actually paint in the editor. A square face has 9 subfaces, in a 3x3 grid.

# Upgrading from Tessera to Tessera Pro

Unfortunately, it's not possible to preserve compatibility in Unity when replacing a .dll with a set of scripts.

If you previously developed a game with Tessera, and you have now downloaded Tessera Pro instead, you'll see the following error.



Don't panic. All your data is still there.

To fix it, please backup your project, then run `Tools > Tessera > Upgrade Scene to Tessera Pro`.

If you like, you can delete all the "dummy" files in the Tessera folder - these files only exist to simplify this transition.

## Advanced

If you don't like upgrading scene by scene, you can edit Unity's yaml directly. The following instructions replace the main script references.

Find: `fileID: -65694588, guid: 5ec9deea42ffdf94eae3261973878f98`

Replace: `fileID: 11500000, guid: e3ad2bf01b7a6b7409eb683402aa8669`

Find: `fileID: 2003858105, guid: 5ec9deea42ffdf94eae3261973878f98`

Replace: `fileID: 11500000, guid: 8a3f7e4cbfb5a184b8e397a0175d7112`

Find: `fileID: -96226770, guid: 5ec9deea42ffdf94eae3261973878f98`

Replace: `fileID: 11500000, guid: 333e56fb2e5d1ff4bb53c10611586ded`

Find: `fileID: 1044799892, guid: 5ec9deea42ffdf94eae3261973878f98`

Replace: `fileID: 11500000, guid: d1efb6dc65363b7479c2f8be4b856e61`

Find: `fileID: 1700055444, guid: 5ec9deea42ffdf94eae3261973878f98`

Replace: `fileID: 11500000, guid: eae741d319c3bff43839bad0f95dceca`

Find: `fileID: -758439564, guid: 5ec9deea42ffdf94eae3261973878f98`

Replace: `fileID: 11500000, guid: b6f7252aa33bd554b9de1f6d885c2d7d`

Save your changes, then reload the scene in Unity. If done correctly, the scripts should now work.

## Downgrading

Downgrades can be done by following the Advanced instructions, reversing Find and Replace.

# Release notes

## 6.3.1

- Fixed palette serialization in Unity 2022 and up

## 6.3.0

- Can now specify TileSelectionType.Ordered
- Mesh grids now support pins in layers above/below the generation. (Pro only)
- TesseraPinned now accepts non-cube tiles
- Fixed bug with big tiles 3x3 and larger
- Fix bug with stacking triangle prism tiles
- Fix bug with infinite generator and square tiles
- Fix Upgrade Scene to Tessera Pro script

## 6.2.1

- Fix broken interaction of skybox/pins on big tiles

## 6.2.0

- HTML Docs are now searchable
- Revamped [TesseraInstantiateOutput](#) to be more useful
- TesseraVolume component now accepts volumes in the form of tiles as well as colliders.
- Fix allowing mouse to be dragged while painting in later Unity versions
- Tessera no longer hangs if there's a single tile in the generator
- Fixed big tiles on mesh grids
- Fixed TesseraVolumes
- MultipassGenerator now has StartGenerate method.

## 6.1.0

- Added [ModelType.MeshEdges](#) that infers the tile connections from meshes. (Pro only)
- Added [Paint Tile From Mesh](#) utility. (Pro only)
- Added [VolumeType.TilesetExclude](#)
- Added "Show Always" setting for working with scenes containing tessera tiles
- Infinite generator now works with Overlapping models. (Pro only)
- Infinite generator now supports [strip](#) option (Pro only)
- Added a sample [SimpleOverlapping](#) demonstrating the above. (Pro only)
- Added a sample [Building](#) (Pro only)
- Icons are now slightly sharper
- Errors in AnimatedGenerator handled better
- Fixed issue with deleted tiles
- Fix issue with masks on non-square grids
- Fix issue with triangle reflections
- Update to using Sylves 1.0

## 6.0.5

- Better error messages for some contradiction failures.
- Fix count constraint on hex/triangle grids.

- Add method CreateVolume.

## 6.0.4

- Fix issues with volumes on hex grids
- Fix issue with triangle mesh sample

## 6.0.3

- Fix issues with pinned tiles
- Fix several issues with triangle prism grids
- Fix issues with big tiles on mesh surfaces

## 6.0.2

- Fix prefab links to no longer attempt to link objects that are not the root of the prefab
- Fix handling of rotated tile constraints in the scene
- Fix tile constraints for triangle prism grids
- Fix height of triangle prism grids

## 6.0.1

- Fix interaction between MaskOut volumes and other volumes
- Fix submesh filter / City sample

## 6.0.0

- **Moved files, you'll need to delete the Tessera directory and reload from scratch**
- Min supported version now Unity 2019.4.
- Extensive internal changes, Tessera now uses [Sylves](#) internally.
- Tessera no longer has to be placed in the root `Assets/` folder.
- Generating in-editor will connect instances to prefabs.
- Provided an `.asmdef` file for faster compilation and avoiding namespace clashes (Pro only).
- Fix for Smooth Normals (Pro only).
- Added [Clone Sample](#) button for diagnosing issues with overlapping model (Pro only).
- Overlapping samples are a bit less fussy about exact placement.
- Improve error message when using native models.
- Users are now warned if tiles are marked as Static.
- [Shortcuts](#) now use Unity's Shortcut Manager.
- Tiles are now automatically resized to match the generator.
- Added samples:
  - [GrassPathsAnimated](#) (Pro only).
  - [GrassPathsContradiction](#) (Pro only).
- Add [TesseraUncertainty](#) to better debug issues (Pro only).
- Fixed InfiniteGenerator to allow the root object to be rotated (Pro only)
- Using AnimatedGenerator no longer slows the editor over time (Pro only)
- [TesseraMeshOutput](#) has been revamped (Pro only):
  - Better support for multiple materials
  - Supports breaking the mesh into multiple chunks
  - Handles colliders
- Small tweaks for Dark Mode

## 5.2.0

- Fix errors with nesting Foldout Headers in later Unity versions.
- Added a separate build for Unity 2021.2+. This fixes various failures and warnings Unity generates (non-Pro only)
- Add [Border Constraint](#) for setting the tiles on the border of the generation.

## 5.1.1

- Fix Tiles list not displaying in recent Unity versions
- Fix for AnimatedGenerator not backtracking properly
- Improved Inspector GUI for AnimatedGenerator
- Fix issue with Path constraint in DeBroglie

## 5.1.0

- Added [Overlapping model support](#) (Pro only)
- Added samples:
  - [GrassPathsOverlapping](#) (Pro only)
  - [Plains](#) (Pro only)
- Generator output now available in new [tileData](#) format.
- Generator seed can now be set in the UI.
- AnimatedGenerator now respects stepCount.
- Fix issues with square tiles/grids
- Fix issue with pins and big tiles
- Fix hex/triangle vertical adjacencies
- Performance improvement for long tile lists in the Inspector

## 5.0.0

- **Moved files, you'll need to delete the Tessera directory and reload from scratch**
- Added [symmetry](#) option improving performance in some cases.
- Added [multipass generator](#)
- Added [infinite generator](#) (Pro only)
- Added [instantiate output](#) (Pro only)
- Reworked MirrorConstraint somewhat (Pro only)
- Warn if palette is set inconsistently.
- Added [sample docs](#) describing what demos are available.
- Added more samples:
  - [GrassPathsInfinite](#)
  - [GrassPathsWithPathConstraint](#)
  - [PlatformerInfinite](#)
  - [Skyscrapers](#)
  - [Truchet](#)
- Added some keyboard shortcuts, Shift-Click, A, Shift+A and X.
- Fix pin constraints for square cells.

## 4.3.0

- Fixed warnings about obsolete APIs (Pro only)
- Fixes for hexagon and cube rotations, causing misaligned tiles.
- Fix Tessera Bolt integration.

- Fix MirrorConstraint editor exceptions (Pro only)

## 4.2.0

- Added [TesseraSquareTile](#) and updated the Platformer sample to use it
- Added [failureMode](#) and other options for diagnosing issues.
- Fixed using AnimatedGenerator and TileOutput at the same time. (Pro only)
- Fixed Regenerate function in play mode
- Fixed exceptions when using masking.

## 4.1.1

- Fix stack overflow error when using Path constraint on large maps

## 4.1.0

- Add multipass sample and [tutorial](#)
- Add HexRaceway sample (Pro only)
- Added many PathConstraint options (Pro only):
  - [loop](#)
  - [acyclic](#)
  - [parity](#)
- Added many extra generation options:
  - [stepLimit](#)
  - [algorithm](#)
  - [recordUndo](#)
- Added notes on [quality and performance](#)
- Fixed issue with normals when generating on mesh surfaces (Pro only)

## 4.0.0

- [Hexagonal and triangular tiles are now supported.](#) (Pro only)
- [Triangle mesh surfaces now supported](#) (instead of just quad meshes) (Pro only)
- AnimatedGenerator has some support for multithreading (Pro only)
- Major internal refactoring
- Some performance improvements
- Worker thread now registers with Unity's profiler
- Improvements to provided samples
- Better support for Unity 2020
- Now easier to access the completion object after calling StartGenerate.
- Fixes some issues with Big Tiles on mesh surfaces (Pro only)
- Fix surface meshes reflecting every tile by default (Pro only)
- Removed TesseraGenerator.initialConstraints (use TesseraGeneratorOptions.initialConstraints)
- Fixed using `InstantiateChildrenOnly` with `TesseraMeshOutput`

## 3.4.1

- Instantiated tiles now have names that include their cell location.
- Fix exception in PathConstraint when Prioritize is on.
- Fix Volumes
- Fix rare issue with big tiles on mesh surfaces.

- Warn if meshes are not readable and using Mesh Output.

## 3.4.0

- Performance improvements with pins and volumes
- Volumes can now be set to [MaskOut](#)
- Refactored GetInitialConstraint methods and TesseraVolumeFilter (breaking)
- Fixed exception with out of bounds palette indices
- Fixed interaction between big tiles and path constraint (Pro only)

## 3.3.0

- Pinned Tile Constraints now work on Surface Meshes (Pro only).
- Refactored some code, changed API of ITesseraTileOutput (Pro only) (breaking)
- Added "Fix It" for inconsistent tile sizes.
- Added [TesseraPinned](#) for more initial constraint options.
- Added extra samples
- Added "Upgrade to Tessera Pro" utility (Pro only)

## 3.2.0

- Tile rotations no longer confined to XZ plane
- Added tooltips to inspector
- [Mirror constraint](#) now supports on all 3 axes (Pro only)
- [Add support for Bolt](#)

## 3.1.2

- Fixed Instantiate Only Children to work with mesh deformations
- Fixed "BoxColliders do not support negative scale or size" warning.
- Fixed generation on mesh surfaces with multiple layers
- Volumes work with generation on mesh surfaces
- Improved Smooth Normals setting

## 3.1.1

- Asset can now be loaded in Unity 2019.1 and 2019.2
- Fixed the missing script in the Castle sample

## 3.1.0

- Add [Tessera Volume](#)
- Dungeon example now demos Volumes
- WebGL builds now ignore the multithreaded option
- Can now choose whether [normals should be smooth](#) or not when generating on a surface (Pro only)
- Can now [filter tiles per-submesh](#) when generating on a surface (Pro only)
- Add [Prioritize](#) option to PathConstraint (Pro only)
- Add [Separation constraint](#) (Pro only)
- Added City example to Samples (Pro only)
- Fixed bug in AnimatedGenerator
- Fixed bug when normal smoothing

## 3.0.0

- Various api changes, larger internal refactorings (breaking)
- Support [generation on surface of a mesh](#) (Pro only)
- Paint tools no longer switches off when you click children of tiles
- Add [Clear](#) and Regenerate methods
- Improved validation messages
- Fixed exception for Mesh Output when materials are missing

## 2.3.0

- Tessera Palette now serializes correctly
- Fix some Inspector display glitches in Unity 2019.3
- Mesh output can now be animated (Pro only)

## 2.2.0

- [Generation can now be animated](#) (Pro only)
- [Tilemap output](#) (Pro only)
- [Mesh output](#) (Pro only)
- Added "Show all" view option when painting.

## 2.1.0

- Added a palette asset that lets you:
  - Customize the paint colors Tessera uses
  - Name the colors (shows in tooltips)
  - Control what colors match each other
- Added a new sample, Platformer
- Fixed a bug that prevented the use of big tiles as fixed tile constraints
- A warning is now emitted if inconsistent tileSize are used
- Multithreading can now be disabled, for platforms that don't support it.

## v2.0.0

- Some performance improvements
- Visible source code (Pro only)
- Added [CountConstraint](#) (Pro only)
- Added [MirrorConstraint](#) (Pro only)
- Added [PathConstraint](#) (Pro only)
- Seeds have changed (breaking)
- Removed [defaultParent](#) (breaking)
- Added a new sample, Dungeon.

## v1.1.1

- Fixed issue with reflected tiles using incorrect rotation for bottom face

## v1.1.0

- Fixed "BeginLayoutGroup must be called first" errors.

- Fixed issue with rotated initial tile constraints.
- Fixed a display glitch in orthographic views
- Added keyboard shortcuts:
  - Delete to remove tiles from the generators list.
  - Z to toggle backfaces.
- Added another scene to samples.
- Improved [documentation on constraints](#).
- Added [contradictionLocation](#)

## v1.0.1

- Removed a Debug.Log line
- [Random seed](#) can now be set. Default from Unity.Random.
- "Clear Children" button on Generator component.
- Fix spurious exceptions when calling Generate.

## v1.0.0

- Initial release

# Namespace Tessera

## Classes

### [AnimatedGenerator](#)

Attach this to a TesseraGenerator to run the generator stepwise over several updates, displaying the changes so far.

#### NOTE

This class is available only in Tessera Pro

### [BiMap<U, V>](#)

Represents a 1:1 mapping between two types

### [BorderConstraint](#)

Forces cells near the edge to be a particular tile. Compare with [skyBox](#).

#### NOTE

This class is available only in Tessera Pro

### [BorderItem](#)

### [CountConstraint](#)

Keeps track of the number of tiles in a given set, and ensure it is less than / more than a given number.

#### NOTE

This class is available only in Tessera Pro

### [CubeDirExtensions](#)

### [CubeFaceDirExtensions](#)

### [EdgeDetection](#)

This utility analyses a mesh, finding a hash that indicates what it can connect two. There's two steps to the analysis.

- For each corner of the cell, determine if it is inside or outside the mesh. This handles non watertight meshes.
- For each face of the cell, find the mesh edges coincident to that face.

The hashes are unique to a specific face and specific rotation of the tile, and are structured in a way that you can get the same has for a left pointing and right pointing face if they share the same mesh edges and corners.

### [EdgeDetection.Face](#)

### [EnumeratorWithResult<T>](#)

An IEnumerator that also records a given result when it is finished. It is intended for use with Unity coroutines.

### [FaceDetails](#)

Records the painted colors for a single face of one cube in a [TesseraTile](#)

### [HexGeometryUtils](#)

**NOTE**

This class is available only in Tessera Pro

## HexPrismDirExtensions

**NOTE**

This class is available only in Tessera Pro

## HexPrismFaceDirExtensions

**NOTE**

This class is available only in Tessera Pro

## InfiniteGenerator

### InstantiateOutput

#### MeshUtils

Utility for working with meshes.

**NOTE**

This class is available only in Tessera Pro

## MirrorConstraint

Ensures that the generation is symmetric when x-axis mirrored. If there are any tile constraints, they will not be mirrored.

**NOTE**

This class is available only in Tessera Pro

## PaletteEntry

### PathConstraint

Forces a network of tiles to connect with each other, so there is always a complete path between them. Two tiles connect along the path if:

- Both tiles are in `pathTiles` (if `hasPathTiles` set); and
- The central color of the sides of the tiles leading to each other are in `pathColors` (if `pathColors` set)

**NOTE**

This class is available only in Tessera Pro

## PrefixLookup<T>

Stores key-value pairs, with efficient searching for the longest key that is a prefix of a given string.

### SeparationConstraint

#### SquareFaceDirExtensions

## [SquareFaceExtensions](#)

### [SylvesConversions](#)

Utilities for converting legacy Tessera enumerations to Sylves equivalents

### [SylvesExtensions](#)

#### [TRS](#)

Represents a position / rotation and scale. Much like a Transform, but without the association with a unity object.

### [TesseraBaseSceneConstraintComponent](#)

Marks any component in the scene that can be interpreted by Tessera as a constraint. These eventually become [ITesseralInitialConstraint](#) objects

### [TesseraCompletion](#)

Returned by [TesseraGenerator](#) after generation finishes

### [TesseraConstraint](#)

Abstract class for all generator constraint components.

#### **NOTE**

This class is available only in Tessera Pro

### [TesseraGenerateOptions](#)

Additional settings to customize the generation at runtime.

### [TesseraGenerator](#)

GameObjects with this behaviour contain utilities to generate tile based levels using Wave Function Collapse (WFC). Call [Generate\(TesseraGenerateOptions\)](#) or [StartGenerate\(TesseraGenerateOptions\)](#) to run. The generation takes the following steps:

- Inspect the tiles in [tiles](#) and work out how they rotate and connect to each other.
- Setup any initial constraints that fix parts of the generation ([initialConstraints](#)).
- Fix the boundary of the generation if [skyBox](#) is set.
- Generate a set of tile instances that fits the above tiles and constraints.
- Optionally [retries](#) or [backtrack](#).
- Instantiates the tile instances.

### [TesseraHexTile](#)

GameObjects with this behaviour record adjacency information for use with a [TesseraGenerator](#).

### [TesseralInitialConstraint](#)

Initial constraint objects fix parts of the generation process in places. Use the utility methods on [TesseraGenerator](#) to create these objects.

### [TesseralInitialConstraintBuilder](#)

Utility for creating [ITesseralInitialConstraint](#) objects, which describe a variety of different constraints.

### [TesseraInstantiateEvent](#)

### [TesseraInstantiateOutput](#)

Attach this to a [TesseraGenerator](#) to control how tiles are instantiated.

#### **NOTE**

This class is available only in Tessera Pro

### [TesseraInstantiateOutputTileMapping](#)

#### [TesseraMeshOutput](#)

Attach this to a TesseraGenerator to output the tiles to a single mesh instead of instantiating them.

#### **NOTE**

This class is available only in Tessera Pro

### [TesseraMultipassGenerator](#)

#### [TesseraMultipassPass](#)

#### [TesseraPalette](#)

#### [TesseraPinConstraint](#)

#### [TesseraPinned](#)

#### [TesseraSquareTile](#)

GameObjects with this behaviour record adjacency information for use with a [TesseraGenerator](#).

#### [TesseraTile](#)

GameObjects with this behaviour record adjacency information for use with a [TesseraGenerator](#).

#### [TesseraTileBase](#)

#### [TesseraTileInstance](#)

Represents a request to instantiate a TesseraTile, post generation.

#### [TesseraTilemapOutput](#)

Attach this to a TesseraGenerator to output the tiles to a Unity Tilemap component instead of directly instantiating them.

#### **NOTE**

This class is available only in Tessera Pro

### [TesseraTransformedTile](#)

Defines a Unity tile that has a specific transform applied to it. Used by [TesseraTilemapOutput](#)

#### [TesseraTrianglePrismTile](#)

GameObjects with this behaviour record adjacency information for use with a [TesseraGenerator](#).

#### [TesseraUncertainty](#)

If this is set on the "uncertainty tile" used by TesseraGenerator/AnimatedGenerator, it will be populated with data about which tiles are actually possible.

#### [TesseraVolume](#)

#### [TesseraVolumeFilter](#)

## [TileEntry](#)

Specifies a tile to be used by [TesseraGenerator](#)

## [TileList](#)

### [TrianglePrismDirExtensions](#)

#### **NOTE**

This class is available only in Tessera Pro

### [TrianglePrismFaceDirExtensions](#)

#### **NOTE**

This class is available only in Tessera Pro

### [TrianglePrismGeometryUtils](#)

#### **NOTE**

This class is available only in Tessera Pro

## [UnityEngineInterface](#)

### Structs

#### [CubeRotation](#)

#### [HexRotation](#)

Represents rotations / reflections of a hexagon

#### **NOTE**

This class is available only in Tessera Pro

## [ModelTile](#)

Actual tiles used internally. There's a many-to-one relationship between ModelTile and TesseraTile due to rotations and "big" tile support.

#### [OrientedFace](#)

Legacy class used when serializing. You should use SylvesOrientedFace instead. Records the painted colors and location of single face of one cube in a [TesseraTile](#)

#### [SquareRotation](#)

Represents rotations / reflections of a square

#### [SubFaceTuple](#)

Each face of a tessera tile is split up into several sub faces that are independently paintable. There's 9 for a square face - one center, 4 corners and 4 "dirs". The corner / dir values come from Sylves.SquareCellType, and similarly for faces with another number of sides. SubFaceTuple is used for address a specific subface.

It's an improvement on the earlier enum SubFace, which covers the same thing and has yet to be deprecated.

#### [SylvesOrientedFace](#)

Records the painted colors and location of single face of one cube in a [TesseraTile](#)

## [TriangleRotation](#)

Represents rotations / reflections of a hexagon

### **NOTE**

This class is available only in Tessera Pro

## Interfaces

[IEngineInterface](#)

[ISubFaceInfo](#)

[ITesseraInitialConstraint](#)

[ITesseraTileOutput](#)

## Enums

[AnimatedGenerator.AnimatedGeneratorState](#)

[CellFaceDir](#)

Legacy class used to represents a particular face of a generic cell when serializing. You should use Sylves.CellDir instead.

[CellRotation](#)

Legacy class used to represents a rotation of a generic cell when serializing. You use use Sylves.CellRotation instead.

[ChunkCleanupType](#)

[CubeFaceDir](#)

Enum of the 6 faces on a cube.

[FaceType](#)

[FailureMode](#)

[HexPrismFaceDir](#)

Enum of the 8 faces on a hex prism.

### **NOTE**

This class is available only in Tessera Pro

[MirrorConstraint.Axis](#)

[ModelType](#)

Different models Tessera supports. The model dictates how nearby tiles relate to each other.

[PinType](#)

[RotationGroupType](#)

[SquareFaceDir](#)

Enum of the 4 sides of a square.

[SubFaceType](#)

[SylvesHexPrismCorner](#)

[SylvesHexPrismDir](#)

[SylvesTrianglePrismCorner](#)

[SylvesTrianglePrismDir](#)

[TesseraInstantiateBehaviour](#)

[TesseraMeshOutputCollider](#)

[TesseraMeshOutputMaterialGrouping](#)

[TesseraMultipassPassType](#)

[TesseraWfcAlgorithm](#)

[TileSelectionType](#)

[TrianglePrismFaceDir](#)

**NOTE**

This class is available only in Tessera Pro

[VolumeType](#)

# Class AnimatedGenerator

Attach this to a TesseraGenerator to run the generator stepwise over several updates, displaying the changes so far.

## NOTE

This class is available only in Tessera Pro

Inheritance

[object](#)

[Object](#)

[Component](#)

[Behaviour](#)

[MonoBehaviour](#)

[AnimatedGenerator](#)

Inherited Members

[MonoBehaviour.IsInvoking\(\)](#)

[MonoBehaviour.CancelInvoke\(\)](#)

[MonoBehaviour.Invoke\(string, float\)](#)

[MonoBehaviour.InvokeRepeating\(string, float, float\)](#)

[MonoBehaviour.CancelInvoke\(string\)](#)

[MonoBehaviour.IsInvoking\(string\)](#)

[MonoBehaviour.StartCoroutine\(string\)](#)

[MonoBehaviour.StartCoroutine\(string, object\)](#)

[MonoBehaviour.StartCoroutine\(IEnumerator\)](#)

[MonoBehaviour.StartCoroutine\\_Auto\(IEnumerator\)](#)

[MonoBehaviour.StopCoroutine\(IEnumerator\)](#)

[MonoBehaviour.StopCoroutine\(Coroutine\)](#)

[MonoBehaviour.StopCoroutine\(string\)](#)

[MonoBehaviour.StopAllCoroutines\(\)](#)

[MonoBehaviour.print\(object\)](#)

[MonoBehaviour.useGUILayout](#)

[MonoBehaviour.RunWithEditMode](#)

[Behaviour.enabled](#)

[Behaviour.isActiveAndEnabled](#)

[Component.GetComponent\(Type\)](#)

[Component.GetComponent<T>\(\)](#)

[Component.TryGetComponent\(Type, out Component\)](#)

[Component.TryGetComponent<T>\(out T\)](#)

[Component.GetComponent\(string\)](#)

[Component.GetComponentInChildren\(Type, bool\)](#)

[Component.GetComponentInChildren\(Type\)](#)

[Component.GetComponentInChildren<T>\(bool\)](#)

[Component.GetComponentInChildren<T>\(\)](#)

[Component.GetComponentsInChildren\(Type, bool\)](#)

[Component.GetComponentsInChildren\(Type\)](#)

[Component.GetComponentsInChildren<T>\(bool\)](#)

[Component.GetComponentsInChildren<T>\(bool, List<T>\)](#)

[Component.GetComponentsInChildren<T>\(\)](#)

[Component.GetComponentsInChildren<T>\(List<T>\)](#)

[Component.GetComponentInParent\(Type\)](#)

```
Component.GetComponentInParent<T>()
Component.GetComponentsInParent(Type, bool)
Component.GetComponentsInParent(Type)
Component.GetComponentsInParent<T>(bool)
Component.GetComponentsInParent<T>(bool, List<T>)
Component.GetComponentsInParent<T>()
Component.GetComponents(Type)
Component.GetComponents(Type, List<Component>)
Component.GetComponents<T>(List<T>)
Component.GetComponents<T>()
Component.CompareTag(string)
Component.SendMessageUpwards(string, object, SendMessageOptions)
Component.SendMessageUpwards(string, object)
Component.SendMessageUpwards(string)
Component.SendMessageUpwards(string, SendMessageOptions)
Component.SendMessage(string, object)
Component.SendMessage(string)
Component.SendMessage(string, object, SendMessageOptions)
Component.SendMessage(string, SendMessageOptions)
Component.BroadcastMessage(string, object, SendMessageOptions)
Component.BroadcastMessage(string, object)
Component.BroadcastMessage(string)
Component.BroadcastMessage(string, SendMessageOptions)
Component.transform
Component.gameObject
Component.tag
Object.GetInstanceID()
Object.GetHashCode()
Object.Equals(object)
Object.Instantiate(Object, Vector3, Quaternion)
Object.Instantiate(Object, Vector3, Quaternion, Transform)
Object.Instantiate(Object)
Object.Instantiate(Object, Transform)
Object.Instantiate(Object, Transform, bool)
Object.Instantiate<T>(T)
Object.Instantiate<T>(T, Vector3, Quaternion)
Object.Instantiate<T>(T, Vector3, Quaternion, Transform)
Object.Instantiate<T>(T, Transform)
Object.Instantiate<T>(T, Transform, bool)
Object.Destroy(Object, float)
Object.Destroy(Object)
Object.DestroyImmediate(Object, bool)
Object.DestroyImmediate(Object)
Object.FindObjectsOfType(Type)
Object.DontDestroyOnLoad(Object)
Object.DestroyObject(Object, float)
Object.DestroyObject(Object)
Object.FindSceneObjectsOfType(Type)
Object.FindObjectsOfTypeIncludingAssets(Type)
Object.FindObjectsOfType<T>()
ObjectFindObjectOfType<T>()
Object.FindObjectsOfTypeAll(Type)
```

`ObjectFindObjectOfType(Type)`

`Object.ToString()`

`Object.name`

`Object.hideFlags`

Namespace: [Tessera](#)

Assembly: Tessera.dll

Syntax

```
[RequireComponent(typeof(TesseraGenerator))]  
[AddComponentMenu("Tessera/Animated Generator", 41)]  
public class AnimatedGenerator : MonoBehaviour
```

Fields

`multithread`

If true, use threading to avoid stalling Unity. (ignored on WebGL builds)

Declaration

```
public bool multithread
```

Field Value

TYPE	DESCRIPTION
<a href="#">bool</a>	

`progressPerStep`

Declaration

```
public float progressPerStep
```

Field Value

TYPE	DESCRIPTION
<a href="#">float</a>	

`scaleUncertaintyTile`

If true, the uncertainty tiles shrink as the solver gets more certain.

Declaration

```
public bool scaleUncertaintyTile
```

Field Value

TYPE	DESCRIPTION
<a href="#">bool</a>	

`secondsPerStep`

Declaration

```
public float secondsPerStep
```

Field Value

<b>TYPE</b>	<b>DESCRIPTION</b>
float	

## uncertaintyTile

Game object to show in cells that have yet to be fully solved.

Declaration

```
public GameObject uncertaintyTile
```

## Field Value

<b>TYPE</b>	<b>DESCRIPTION</b>
GameObject	

## Properties

### IsStarted

Declaration

```
public bool IsStarted { get; }
```

## Property Value

<b>TYPE</b>	<b>DESCRIPTION</b>
bool	

## State

Declaration

```
public AnimatedGenerator.AnimatedGeneratorState State { get; }
```

## Property Value

<b>TYPE</b>	<b>DESCRIPTION</b>
AnimatedGenerator.AnimatedGeneratorState	

## Methods

### PauseGeneration()

Declaration

```
public void PauseGeneration()
```

### ResumeGeneration()

Declaration

```
public void ResumeGeneration()
```

### StartGeneration()

Declaration

```
public void StartGeneration()
```

## Step()

Declaration

```
public void Step()
```

## StopGeneration()

Declaration

```
public void StopGeneration()
```

# Enum AnimatedGenerator.AnimatedGeneratorState

Namespace: [Tessera](#)

Assembly: Tessera.dll

## Syntax

```
public enum AnimatedGenerator.AnimatedGeneratorState
```

## Fields

NAME	DESCRIPTION
Initializing	
Paused	
Running	
Stopped	

# Class BiMap<U, V>

Represents a 1:1 mapping between two types

Inheritance

[object](#)

[BiMap<U, V>](#)

Implements

[IEnumerable<\(U, V\)>](#)

[IEnumerable](#)

Namespace: [Tessera](#)

Assembly: Tessera.dll

Syntax

```
public class BiMap<U, V> : IEnumerable<(U, V)>, IEnumerable
```

Type Parameters

NAME	DESCRIPTION
U	
V	

Constructors

[BiMap\(IEnumerable<\(U, V\)>\)](#)

Declaration

```
public BiMap(IEnumerable<(U, V)> data)
```

Parameters

TYPE	NAME	DESCRIPTION
<a href="#">IEnumerable&lt;(U, V)&gt;</a>	data	

Properties

[Count](#)

Declaration

```
public int Count { get; }
```

Property Value

TYPE	DESCRIPTION
<a href="#">int</a>	

[this\[U\]](#)

Declaration

```
public V this[U u] { get; }
```

Parameters

TYPE	NAME	DESCRIPTION
U	u	

Property Value

TYPE	DESCRIPTION
V	

this[V]

Declaration

```
public U this[V v] { get; }
```

Parameters

TYPE	NAME	DESCRIPTION
V	v	

Property Value

TYPE	DESCRIPTION
U	

Methods

GetEnumerator()

Declaration

```
public IEnumerator<(U, V)> GetEnumerator()
```

Returns

TYPE	DESCRIPTION
IEnumerator<(U, V)>	

Implements

IEnumerable<T>

IEnumerable

# Class BorderConstraint

Forces cells near the edge to be a particular tile. Compare with [skyBox](#).

## NOTE

This class is available only in Tessera Pro

Inheritance

[object](#)

[Object](#)

[Component](#)

[Behaviour](#)

[MonoBehaviour](#)

[TesseraConstraint](#)

[BorderConstraint](#)

Inherited Members

[MonoBehaviour.IsInvoking\(\)](#)

[MonoBehaviour.CancelInvoke\(\)](#)

[\*\*MonoBehaviour.Invoke\(string, float\)\*\*](#)

[MonoBehaviour.InvokeRepeating\(string, float, float\)](#)

[MonoBehaviour.CancelInvoke\(string\)](#)

[MonoBehaviour.IsInvoking\(string\)](#)

[MonoBehaviour.StartCoroutine\(string\)](#)

[MonoBehaviour.StartCoroutine\(string, object\)](#)

[MonoBehaviour.StartCoroutine\(IEnumerator\)](#)

[MonoBehaviour.StartCoroutine\\_Auto\(IEnumerator\)](#)

[MonoBehaviour.StopCoroutine\(IEnumerator\)](#)

[MonoBehaviour.StopCoroutine\(Coroutine\)](#)

[\*\*MonoBehaviour.StopCoroutine\(string\)\*\*](#)

[MonoBehaviour.StopAllCoroutines\(\)](#)

[MonoBehaviour.print\(object\)](#)

[MonoBehaviour.useGUILayout](#)

[MonoBehaviour.runInEditMode](#)

[Behaviour.enabled](#)

[Behaviour.isActiveAndEnabled](#)

[Component.GetComponent\(Type\)](#)

[Component.GetComponent<T>\(\)](#)

[Component.TryGetComponent\(Type, out Component\)](#)

[Component.TryGetComponent<T>\(out T\)](#)

[Component.GetComponent\(string\)](#)

[Component.GetComponentInChildren\(Type, bool\)](#)

[Component.GetComponentInChildren\(Type\)](#)

[Component.GetComponentInChildren<T>\(bool\)](#)

[Component.GetComponentInChildren<T>\(\)](#)

[Component.GetComponentsInChildren\(Type, bool\)](#)

[Component.GetComponentsInChildren\(Type\)](#)

[Component.GetComponentsInChildren<T>\(bool\)](#)

[Component.GetComponentsInChildren<T>\(bool, List<T>\)](#)

[Component.GetComponentsInChildren<T>\(\)](#)

[Component.GetComponentsInChildren<T>\(List<T>\)](#)

Component.GetComponentInParent(Type)  
Component.GetComponentInParent<T>()  
Component.GetComponentsInParent(Type, bool)  
Component.GetComponentsInParent(Type)  
Component.GetComponentsInParent<T>(bool)  
Component.GetComponentsInParent<T>(bool, List<T>)  
Component.GetComponentsInParent<T>()  
Component.GetComponents(Type)  
Component.GetComponents(Type, List<Component>)  
Component.GetComponents<T>(List<T>)  
Component.GetComponents<T>()  
Component.CompareTag(string)  
Component.SendMessageUpwards(string, object, SendMessageOptions)  
Component.SendMessageUpwards(string, object)  
Component.SendMessageUpwards(string)  
Component.SendMessageUpwards(string, SendMessageOptions)  
Component.SendMessage(string, object)  
Component.SendMessage(string)  
Component.SendMessage(string, object, SendMessageOptions)  
Component.SendMessage(string, SendMessageOptions)  
Component.BroadcastMessage(string, object, SendMessageOptions)  
Component.BroadcastMessage(string, object)  
Component.BroadcastMessage(string)  
Component.BroadcastMessage(string, SendMessageOptions)  
Component.transform  
Component.gameObject  
Component.tag  
Object.GetInstanceID()  
Object.GetHashCode()  
**Object.Equals(object)**  
Object.Instantiate(Object, Vector3, Quaternion)  
Object.Instantiate(Object, Vector3, Quaternion, Transform)  
Object.Instantiate(Object)  
Object.Instantiate(Object, Transform)  
**Object.Instantiate(Object, Transform, bool)**  
Object.Instantiate<T>(T)  
Object.Instantiate<T>(T, Vector3, Quaternion)  
Object.Instantiate<T>(T, Vector3, Quaternion, Transform)  
Object.Instantiate<T>(T, Transform)  
**Object.Instantiate<T>(T, Transform, bool)**  
Object.Destroy(Object, float)  
Object.Destroy(Object)  
Object.DestroyImmediate(Object, bool)  
Object.DestroyImmediate(Object)  
**Object.FindObjectsOfType(Type)**  
Object.DontDestroyOnLoad(Object)  
**Object.DestroyObject(Object, float)**  
Object.DestroyObject(Object)  
**Object.FindSceneObjectsOfType(Type)**  
**Object.FindObjectsOfTypeIncludingAssets(Type)**  
Object.FindObjectsOfType<T>()  
ObjectFindObjectOfType<T>()

[Object.FindObjectsOfTypeAll\(Type\)](#)

[Object.FindObjectOfType\(Type\)](#)

[Object.ToString\(\)](#)

[Object.name](#)

[Object.hideFlags](#)

Namespace: [Tessera](#)

Assembly: Tessera.dll

## Syntax

```
[AddComponentMenu("Tessera/Border Constraint", 21)]  
[RequireComponent(typeof(TesseraGenerator))]  
public class BorderConstraint : TesseraConstraint
```

## Fields

**borders**

Declaration

```
public BorderItem[] borders
```

### Field Value

TYPE	DESCRIPTION
<a href="#">BorderItem</a> []	

# Class BorderItem

Inheritance

[object](#)

BorderItem

Namespace: [Tessera](#)

Assembly: Tessera.dll

Syntax

```
[Serializable]
public class BorderItem
```

Fields

**cellDir**

Declaration

```
public CellDir cellDir
```

Field Value

TYPE	DESCRIPTION
CellDir	

**tile**

Declaration

```
public TesseraTileBase tile
```

Field Value

TYPE	DESCRIPTION
TesseraTileBase	

# Enum CellFaceDir

Legacy class used to represents a particular face of a generic cell when serializing. You should use Sylves.CellDir instead.

Namespace: [Tessera](#)

Assembly: Tessera.dll

Syntax

```
public enum CellFaceDir
```

# Enum CellRotation

Legacy class used to represents a rotation of a generic cell when serializing. You use use Sylves.CellRotation instead.

Namespace: [Tessera](#)

Assembly: Tessera.dll

Syntax

```
public enum CellRotation
```

# Enum ChunkCleanupType

Namespace: [Tessera](#)

Assembly: Tessera.dll

## Syntax

```
public enum ChunkCleanupType
```

## Fields

NAME	DESCRIPTION
Full	Remove everything associated with the chunk.
Memoize	Remove the GameObjects, but keep tile data so they can be recreated exactly
None	Never cleanup chunks

# Class CountConstraint

Keeps track of the number of tiles in a given set, and ensure it is less than / more than a given number.

## NOTE

This class is available only in Tessera Pro

Inheritance

[object](#)

[Object](#)

[Component](#)

[Behaviour](#)

[MonoBehaviour](#)

[TesseraConstraint](#)

[CountConstraint](#)

Inherited Members

[MonoBehaviour.IsInvoking\(\)](#)

[MonoBehaviour.CancelInvoke\(\)](#)

[MonoBehaviour.Invoke\(string, float\)](#)

[MonoBehaviour.InvokeRepeating\(string, float, float\)](#)

[MonoBehaviour.CancelInvoke\(string\)](#)

[MonoBehaviour.IsInvoking\(string\)](#)

[MonoBehaviour.StartCoroutine\(string\)](#)

[MonoBehaviour.StartCoroutine\(string, object\)](#)

[MonoBehaviour.StartCoroutine\(IEnumerator\)](#)

[MonoBehaviour.StartCoroutine\\_Auto\(IEnumerator\)](#)

[MonoBehaviour.StopCoroutine\(IEnumerator\)](#)

[MonoBehaviour.StopCoroutine\(Coroutine\)](#)

[MonoBehaviour.StopCoroutine\(string\)](#)

[MonoBehaviour.StopAllCoroutines\(\)](#)

[MonoBehaviour.print\(object\)](#)

[MonoBehaviour.useGUILayout](#)

[MonoBehaviour.runInEditMode](#)

[Behaviour.enabled](#)

[Behaviour.isActiveAndEnabled](#)

[Component.GetComponent\(Type\)](#)

[Component.GetComponent<T>\(\)](#)

[Component.TryGetComponent\(Type, out Component\)](#)

[Component.TryGetComponent<T>\(out T\)](#)

[Component.GetComponent\(string\)](#)

[Component.GetComponentInChildren\(Type, bool\)](#)

[Component.GetComponentInChildren\(Type\)](#)

[Component.GetComponentInChildren<T>\(bool\)](#)

[Component.GetComponentInChildren<T>\(\)](#)

[Component.GetComponentsInChildren\(Type, bool\)](#)

[Component.GetComponentsInChildren\(Type\)](#)

[Component.GetComponentsInChildren<T>\(bool\)](#)

[Component.GetComponentsInChildren<T>\(bool, List<T>\)](#)

[Component.GetComponentsInChildren<T>\(\)](#)

[Component.GetComponentsInChildren<T>\(List<T>\)](#)

Component.GetComponentInParent(Type)  
Component.GetComponentInParent<T>()  
Component.GetComponentsInParent(Type, bool)  
Component.GetComponentsInParent(Type)  
Component.GetComponentsInParent<T>(bool)  
Component.GetComponentsInParent<T>(bool, List<T>)  
Component.GetComponentsInParent<T>()  
Component.GetComponents(Type)  
Component.GetComponents(Type, List<Component>)  
Component.GetComponents<T>(List<T>)  
Component.GetComponents<T>()  
Component.CompareTag(string)  
Component.SendMessageUpwards(string, object, SendMessageOptions)  
Component.SendMessageUpwards(string, object)  
Component.SendMessageUpwards(string)  
Component.SendMessageUpwards(string, SendMessageOptions)  
Component.SendMessage(string, object)  
Component.SendMessage(string)  
Component.SendMessage(string, object, SendMessageOptions)  
Component.SendMessage(string, SendMessageOptions)  
Component.BroadcastMessage(string, object, SendMessageOptions)  
Component.BroadcastMessage(string, object)  
Component.BroadcastMessage(string)  
Component.BroadcastMessage(string, SendMessageOptions)  
Component.transform  
Component.gameObject  
Component.tag  
Object.GetInstanceID()  
Object.GetHashCode()  
**Object.Equals(object)**  
Object.Instantiate(Object, Vector3, Quaternion)  
Object.Instantiate(Object, Vector3, Quaternion, Transform)  
Object.Instantiate(Object)  
Object.Instantiate(Object, Transform)  
**Object.Instantiate(Object, Transform, bool)**  
Object.Instantiate<T>(T)  
Object.Instantiate<T>(T, Vector3, Quaternion)  
Object.Instantiate<T>(T, Vector3, Quaternion, Transform)  
Object.Instantiate<T>(T, Transform)  
**Object.Instantiate<T>(T, Transform, bool)**  
Object.Destroy(Object, float)  
Object.Destroy(Object)  
Object.DestroyImmediate(Object, bool)  
Object.DestroyImmediate(Object)  
**Object.FindObjectsOfType(Type)**  
Object.DontDestroyOnLoad(Object)  
**Object.DestroyObject(Object, float)**  
Object.DestroyObject(Object)  
**Object.FindSceneObjectsOfType(Type)**  
**Object.FindObjectsOfTypeIncludingAssets(Type)**  
Object.FindObjectsOfType<T>()  
ObjectFindObjectOfType<T>()

[Object.FindObjectsOfTypeAll\(Type\)](#)

[Object.FindObjectOfType\(Type\)](#)

[Object.ToString\(\)](#)

[Object.name](#)

[Object.hideFlags](#)

Namespace: [Tessera](#)

Assembly: Tessera.dll

Syntax

```
[AddComponentMenu("Tessera/Count Constraint", 21)]  
[RequireComponent(typeof(TesseraGenerator))]  
public class CountConstraint : TesseraConstraint
```

Fields

**comparison**

How to compare the count of [tiles](#) to [count](#).

Declaration

```
public CountComparison comparison
```

Field Value

TYPE	DESCRIPTION
CountComparison	

**count**

The count to be compared against.

Declaration

```
public int count
```

Field Value

TYPE	DESCRIPTION
int	

**eager**

If set, this constraint will attempt to pick tiles as early as possible. This can give a better random distribution, but higher chance of contradictions.

Declaration

```
public bool eager
```

Field Value

TYPE	DESCRIPTION
bool	

**tiles**

The set of tiles to count

Declaration

```
public List<TesseraTileBase> tiles
```

Field Value

TYPE	DESCRIPTION
List<TesseraTileBase>	

# Class CubeDirExtensions

Inheritance

[object](#)

CubeDirExtensions

Namespace: [Tessera](#)

Assembly: Tessera.dll

Syntax

```
public static class CubeDirExtensions
```

## Methods

[Forward\(CubeDir\)](#)

Declaration

```
public static Vector3Int Forward(this CubeDir dir)
```

Parameters

TYPE	NAME	DESCRIPTION
CubeDir	dir	

Returns

TYPE	DESCRIPTION
Vector3Int	The normal vector for a given face.

[Inverted\(CubeDir\)](#)

Declaration

```
public static CubeDir Inverted(this CubeDir dir)
```

Parameters

TYPE	NAME	DESCRIPTION
CubeDir	dir	

Returns

TYPE	DESCRIPTION
CubeDir	Returns the face dir with the opposite normal vector.

[Up\(CubeDir\)](#)

Declaration

```
public static Vector3Int Up(this CubeDir dir)
```

Parameters

<b>TYPE</b>	<b>NAME</b>	<b>DESCRIPTION</b>
CubeDir	dir	

Returns

<b>TYPE</b>	<b>DESCRIPTION</b>
Vector3Int	Returns (0, 1, 0) vector for most faces, and returns (0, 0, 1) for the top/bottom faces.

# Enum CubeFaceDir

Enum of the 6 faces on a cube.

Namespace: [Tessera](#)

Assembly: Tessera.dll

Syntax

```
public enum CubeFaceDir
```

Fields

NAME	DESCRIPTION
Back	
Down	
Forward	
Left	
Right	
Up	

Extension Methods

[CubeFaceDirExtensions.Forward\(CubeFaceDir\)](#)

[CubeFaceDirExtensions.Inverted\(CubeFaceDir\)](#)

[CubeFaceDirExtensions.Up\(CubeFaceDir\)](#)

# Class CubeFaceDirExtensions

Inheritance

[object](#)

CubeFaceDirExtensions

Namespace: [Tessera](#)

Assembly: Tessera.dll

Syntax

```
public static class CubeFaceDirExtensions
```

## Methods

[Forward\(CubeFaceDir\)](#)

Declaration

```
public static Vector3Int Forward(this CubeFaceDir faceDir)
```

Parameters

TYPE	NAME	DESCRIPTION
<a href="#">CubeFaceDir</a>	faceDir	

Returns

TYPE	DESCRIPTION
<a href="#">Vector3Int</a>	The normal vector for a given face.

[Inverted\(CubeFaceDir\)](#)

Declaration

```
public static CubeFaceDir Inverted(this CubeFaceDir faceDir)
```

Parameters

TYPE	NAME	DESCRIPTION
<a href="#">CubeFaceDir</a>	faceDir	

Returns

TYPE	DESCRIPTION
<a href="#">CubeFaceDir</a>	Returns the face dir with the opposite normal vector.

[Up\(CubeFaceDir\)](#)

Declaration

```
public static Vector3Int Up(this CubeFaceDir faceDir)
```

Parameters

TYPE	NAME	DESCRIPTION
CubeFaceDir	faceDir	

Returns

TYPE	DESCRIPTION
Vector3Int	Returns (0, 1, 0) vector for most faces, and returns (0, 0, 1) for the top/bottom faces.

# Struct CubeRotation

Inherited Members

[ValueType.ToString\(\)](#)

Namespace: [Tessera](#)

Assembly: Tessera.dll

Syntax

```
public struct CubeRotation
```

Properties

All

Declaration

```
public static IEnumerable<CubeRotation> All { get; }
```

Property Value

TYPE	DESCRIPTION
<a href="#">IEnumerable&lt;CubeRotation&gt;</a>	

Identity

Declaration

```
public static CubeRotation Identity { get; }
```

Property Value

TYPE	DESCRIPTION
<a href="#">CubeRotation</a>	

IsReflection

Declaration

```
public bool IsReflection { get; }
```

Property Value

TYPE	DESCRIPTION
<a href="#">bool</a>	

ReflectX

Declaration

```
public static CubeRotation ReflectX { get; }
```

Property Value

TYPE	DESCRIPTION
<a href="#">CubeRotation</a>	

## ReflectY

### Declaration

```
public static CubeRotation ReflectY { get; }
```

### Property Value

TYPE	DESCRIPTION
CubeRotation	

## ReflectZ

### Declaration

```
public static CubeRotation ReflectZ { get; }
```

### Property Value

TYPE	DESCRIPTION
CubeRotation	

## RotateXY

### Declaration

```
public static CubeRotation RotateXY { get; }
```

### Property Value

TYPE	DESCRIPTION
CubeRotation	

## RotateXZ

### Declaration

```
public static CubeRotation RotateXZ { get; }
```

### Property Value

TYPE	DESCRIPTION
CubeRotation	

## RotateYZ

### Declaration

```
public static CubeRotation RotateYZ { get; }
```

### Property Value

TYPE	DESCRIPTION
CubeRotation	

## Methods

## **Equals(object)**

Declaration

```
public override bool Equals(object obj)
```

Parameters

TYPE	NAME	DESCRIPTION
object	obj	

Returns

TYPE	DESCRIPTION
bool	

Overrides

[ValueType.Equals\(object\)](#)

## **GetHashCode()**

Declaration

```
public override int GetHashCode()
```

Returns

TYPE	DESCRIPTION
int	

Overrides

[ValueType.GetHashCode\(\)](#)

## **Invert()**

Declaration

```
public CubeRotation Invert()
```

Returns

TYPE	DESCRIPTION
CubeRotation	

## **Operators**

### **operator ==(CubeRotation, CubeRotation)**

Declaration

```
public static bool operator ==(CubeRotation a, CubeRotation b)
```

Parameters

TYPE	NAME	DESCRIPTION
CubeRotation	a	

TYPE	NAME	DESCRIPTION
CubeRotation	b	

Returns

TYPE	DESCRIPTION
bool	

### implicit operator CubeRotation(CellRotation)

Declaration

```
public static implicit operator CubeRotation(CellRotation r)
```

Parameters

TYPE	NAME	DESCRIPTION
CellRotation	r	

Returns

TYPE	DESCRIPTION
CubeRotation	

### implicit operator CellRotation(CubeRotation)

Declaration

```
public static implicit operator CellRotation(CubeRotation r)
```

Parameters

TYPE	NAME	DESCRIPTION
CubeRotation	r	

Returns

TYPE	DESCRIPTION
CellRotation	

### operator !=(CubeRotation, CubeRotation)

Declaration

```
public static bool operator !=(CubeRotation a, CubeRotation b)
```

Parameters

TYPE	NAME	DESCRIPTION
CubeRotation	a	
CubeRotation	b	

Returns

TYPE	DESCRIPTION
bool	

operator \*(CubeRotation, CubeFaceDir)

Declaration

```
public static CubeFaceDir operator *(CubeRotation r, CubeFaceDir dir)
```

Parameters

TYPE	NAME	DESCRIPTION
CubeRotation	r	
CubeFaceDir	dir	

Returns

TYPE	DESCRIPTION
CubeFaceDir	

operator \*(CubeRotation, CubeRotation)

Declaration

```
public static CubeRotation operator *(CubeRotation a, CubeRotation b)
```

Parameters

TYPE	NAME	DESCRIPTION
CubeRotation	a	
CubeRotation	b	

Returns

TYPE	DESCRIPTION
CubeRotation	

operator \*(CubeRotation, BoundsInt)

Declaration

```
public static BoundsInt operator *(CubeRotation r, BoundsInt bounds)
```

Parameters

TYPE	NAME	DESCRIPTION
CubeRotation	r	
BoundsInt	bounds	

Returns

TYPE	DESCRIPTION
BoundsInt	

**operator \*(CubeRotation, Vector3)**

Declaration

```
public static Vector3 operator *(CubeRotation r, Vector3 v)
```

Parameters

TYPE	NAME	DESCRIPTION
CubeRotation	r	
Vector3	v	

Returns

TYPE	DESCRIPTION
Vector3	

**operator \*(CubeRotation, Vector3Int)**

Declaration

```
public static Vector3Int operator *(CubeRotation r, Vector3Int v)
```

Parameters

TYPE	NAME	DESCRIPTION
CubeRotation	r	
Vector3Int	v	

Returns

TYPE	DESCRIPTION
Vector3Int	

# Class EdgeDetection

This utility analyses a mesh, finding a hash that indicates what it can connect two. There's two steps to the analysis.

- For each corner of the cell, determine if it is inside or outside the mesh. This handles non watertight meshes.
- For each face of the cell, find the mesh edges coincident to that face.

The hashes are unique to a specific face and specific rotation of the tile, and are structured in a way that you can get the same has for a left pointing and right pointing face if they share the same mesh edges and corners.

Inheritance

[object](#)

**EdgeDetection**

Namespace: [Tessera](#)

Assembly: Tessera.dll

Syntax

```
public static class EdgeDetection
```

Fields

**DefaultTolerance**

Declaration

```
public const float DefaultTolerance = 1E-06
```

Field Value

TYPE	DESCRIPTION
<a href="#">float</a>	

Methods

**GetFaces(TesseraTileBase, IEnumerable<CellRotation>, MeshData, HashSet<(Vector3Int, CellCorner)>, float, bool)**

Declaration

```
public static IList<EdgeDetection.Face> GetFaces(TesseraTileBase tile, IEnumerable<CellRotation> rotations,
MeshData meshData, HashSet<(Vector3Int, CellCorner)> interiorCorners, float tolerance = 1E-06, bool debug =
false)
```

Parameters

TYPE	NAME	DESCRIPTION
<a href="#">TesseraTileBase</a>	tile	
<a href="#">IEnumerable&lt;CellRotation&gt;</a>	rotations	
<a href="#">MeshData</a>	meshData	
<a href="#">HashSet&lt;(Vector3Int, CellCorner)&gt;</a>	interiorCorners	
<a href="#">float</a>	tolerance	
<a href="#">bool</a>	debug	

Returns

TYPE	DESCRIPTION
IList<EdgeDetection.Face>	

GetFaces(TesseraTileBase, IEnumerable<CellRotation>, Mesh, float, bool)

Extracts a summary of all the tile faces.

Declaration

```
public static IList<EdgeDetection.Face> GetFaces(TesseraTileBase tile, IEnumerable<CellRotation> rotations, Mesh mesh, float tolerance = 1E-06, bool debug = false)
```

Parameters

TYPE	NAME	DESCRIPTION
TesseraTileBase	tile	
IEnumerable<CellRotation>	rotations	
Mesh	mesh	
float	tolerance	
bool	debug	

Returns

TYPE	DESCRIPTION
IList<EdgeDetection.Face>	

GetInteriorCorners(TesseraTileBase, MeshData, bool)

null mesh => all exterior empty mesh => all interior

Declaration

```
public static HashSet<(Vector3Int, CellCorner)> GetInteriorCorners(TesseraTileBase tile, MeshData meshData, bool debug = false)
```

Parameters

TYPE	NAME	DESCRIPTION
TesseraTileBase	tile	
MeshData	meshData	
bool	debug	

Returns

TYPE	DESCRIPTION

TYPE	DESCRIPTION
<a href="#">HashSet</a> <(Vector3Int, CellCorner)>	

# Class EdgeDetection.Face

Inheritance

[object](#)

[EdgeDetection.Face](#)

Namespace: [Tessera](#)

Assembly: Tessera.dll

Syntax

```
public class EdgeDetection.Face
```

Fields

**dir**

Declaration

```
public CellDir dir
```

Field Value

TYPE	DESCRIPTION
CellDir	

**hash1**

Declaration

```
public int hash1
```

Field Value

TYPE	DESCRIPTION
int	

**hash2**

Declaration

```
public int hash2
```

Field Value

TYPE	DESCRIPTION
int	

**offset**

Declaration

```
public Vector3Int offset
```

Field Value

TYPE	DESCRIPTION
Vector3Int	

## rotation

### Declaration

```
public CellRotation rotation
```

### Field Value

TYPE	DESCRIPTION
CellRotation	

## tile

### Declaration

```
public TesseraTileBase tile
```

### Field Value

TYPE	DESCRIPTION
TesseraTileBase	

# Class EnumeratorWithResult<T>

An IEnumerator that also records a given result when it is finished. It is intended for use with Unity coroutines.

Inheritance

[object](#)

[IEnumeratorWithResult<T>](#)

Implements

[IEnumerator](#)

Namespace: [Tessera](#)

Assembly: Tessera.dll

Syntax

```
public class EnumeratorWithResult<T> : IEnumerator
```

Type Parameters

NAME	DESCRIPTION
T	

Constructors

[EnumeratorWithResult\(IEnumerator\)](#)

Declaration

```
public EnumeratorWithResult(IEnumerator e)
```

Parameters

TYPE	NAME	DESCRIPTION
<a href="#">IEnumerator</a>	e	

Properties

[Current](#)

Declaration

```
public object Current { get; }
```

Property Value

TYPE	DESCRIPTION
<a href="#">object</a>	

[Result](#)

The value returned by this enumerator. This will throw if you attempt to access it before fully iterating through the enumerator.

Declaration

```
public T Result { get; }
```

Property Value

TYPE	DESCRIPTION
T	

## Methods

### MoveNext()

Declaration

```
public bool MoveNext()
```

Returns

TYPE	DESCRIPTION
bool	

### Reset()

Declaration

```
public void Reset()
```

### TryGetResult(out T)

The value returned by this enumerator. This will return false if you attempt to access it before fully iterating through the enumerator.

Declaration

```
public bool TryGetResult(out T result)
```

Parameters

TYPE	NAME	DESCRIPTION
T	result	

Returns

TYPE	DESCRIPTION
bool	

## Implements

IEnumerator

# Class FaceDetails

Records the painted colors for a single face of one cube in a [TesseraTile](#)

Inheritance

[object](#)

[FaceDetails](#)

Namespace: [Tessera](#)

Assembly: Tessera.dll

Syntax

```
[Serializable]
public class FaceDetails
```

Fields

**bottom**

Declaration

```
public int bottom
```

Field Value

TYPE	DESCRIPTION
<a href="#">int</a>	

**bottomLeft**

Declaration

```
public int bottomLeft
```

Field Value

TYPE	DESCRIPTION
<a href="#">int</a>	

**bottomRight**

Declaration

```
public int bottomRight
```

Field Value

TYPE	DESCRIPTION
<a href="#">int</a>	

**center**

Declaration

```
public int center
```

Field Value

TYPE	DESCRIPTION
int	

## faceType

Declaration

```
public FaceType faceType
```

Field Value

TYPE	DESCRIPTION
FaceType	

## hexBottomRightAndRight

Declaration

```
public int hexBottomRightAndRight
```

Field Value

TYPE	DESCRIPTION
int	

## hexLeftAndBottomLeft

Declaration

```
public int hexLeftAndBottomLeft
```

Field Value

TYPE	DESCRIPTION
int	

## hexRightAndTopRight

Declaration

```
public int hexRightAndTopRight
```

Field Value

TYPE	DESCRIPTION
int	

## hexTopLeftAndLeft

Declaration

```
public int hexTopLeftAndLeft
```

Field Value

TYPE	DESCRIPTION
int	

## left

Declaration

```
public int left
```

Field Value

TYPE	DESCRIPTION
int	

## meshEdgeHash1

Declaration

```
[NonSerialized]  
public int meshEdgeHash1
```

Field Value

TYPE	DESCRIPTION
int	

## meshEdgeHash2

Declaration

```
[NonSerialized]  
public int meshEdgeHash2
```

Field Value

TYPE	DESCRIPTION
int	

## right

Declaration

```
public int right
```

Field Value

TYPE	DESCRIPTION
int	

## top

Declaration

```
public int top
```

Field Value

<b>TYPE</b>	<b>DESCRIPTION</b>
int	

## topLeft

Declaration

```
public int topLeft
```

Field Value

<b>TYPE</b>	<b>DESCRIPTION</b>
int	

## topRight

Declaration

```
public int topRight
```

Field Value

<b>TYPE</b>	<b>DESCRIPTION</b>
int	

## Properties

### this[SubFaceTuple]

Declaration

```
public int this[SubFaceTuple subFace] { get; set; }
```

Parameters

<b>TYPE</b>	<b>NAME</b>	<b>DESCRIPTION</b>
SubFaceTuple	subFace	

Property Value

<b>TYPE</b>	<b>DESCRIPTION</b>
int	

## hexBottomLeft

Declaration

```
public int hexBottomLeft { get; set; }
```

Property Value

<b>TYPE</b>	<b>DESCRIPTION</b>
int	

## hexBottomLeftAndBottomRight

Declaration

```
public int hexBottomLeftAndBottomRight { get; set; }
```

Property Value

TYPE	DESCRIPTION
int	

## hexBottomRight

Declaration

```
public int hexBottomRight { get; set; }
```

Property Value

TYPE	DESCRIPTION
int	

## hexCenter

Declaration

```
public int hexCenter { get; set; }
```

Property Value

TYPE	DESCRIPTION
int	

## hexLeft

Declaration

```
public int hexLeft { get; set; }
```

Property Value

TYPE	DESCRIPTION
int	

## hexRight

Declaration

```
public int hexRight { get; set; }
```

Property Value

TYPE	DESCRIPTION
int	

## hexTopLeft

## Declaration

```
public int hexTopLeft { get; set; }
```

## Property Value

TYPE	DESCRIPTION
int	

## hexTopRight

### Declaration

```
public int hexTopRight { get; set; }
```

## Property Value

TYPE	DESCRIPTION
int	

## hexTopRightAndTopLeft

### Declaration

```
public int hexTopRightAndTopLeft { get; set; }
```

## Property Value

TYPE	DESCRIPTION
int	

## Methods

### IsEquivalent(FaceDetails)

Checks if two FaceDetails have the same values. This is an exact match, with no reflection built in. See TesseraPalette.Match for a fuzzier match.

### Declaration

```
public bool IsEquivalent(FaceDetails other)
```

## Parameters

TYPE	NAME	DESCRIPTION
FaceDetails	other	

## Returns

TYPE	DESCRIPTION
bool	

## ToString()

### Declaration

```
public override string ToString()
```

Returns

TYPE	DESCRIPTION
string	

Overrides

[object.ToString\(\)](#)

# Enum FaceType

Namespace: [Tessera](#)

Assembly: Tessera.dll

## Syntax

```
public enum FaceType
```

## Fields

NAME	DESCRIPTION
Edge	
Hex	
Point	
Square	
Triangle	

# Enum FailureMode

Namespace: [Tessera](#)

Assembly: Tessera.dll

## Syntax

```
public enum FailureMode
```

## Fields

NAME	DESCRIPTION
Cancel	If a failure occurs, don't output anything
Last	If a failure occurs, output the progress so far
LastGood	If a failure occurs, backtrack to the last safe point.
Minimal	Examines the progress so far for the minimal set of tiles that cause an issue

# Class HexGeometryUtils

## NOTE

This class is available only in Tessera Pro

## Inheritance

[object](#)

**HexGeometryUtils**

Namespace: [Tessera](#)

Assembly: Tessera.dll

## Syntax

```
public static class HexGeometryUtils
```

## Methods

**CubeRotate(HexRotation, Vector3Int)**

### Declaration

```
public static Vector3Int CubeRotate(HexRotation rotation, Vector3Int cc)
```

### Parameters

TYPE	NAME	DESCRIPTION
<a href="#">HexRotation</a>	rotation	
<a href="#">Vector3Int</a>	cc	

### Returns

TYPE	DESCRIPTION
<a href="#">Vector3Int</a>	

**FromCubeCords(Vector3Int, int)**

### Declaration

```
public static Vector3Int FromCubeCords(Vector3Int cc, int y = 0)
```

### Parameters

TYPE	NAME	DESCRIPTION
<a href="#">Vector3Int</a>	cc	
<a href="#">int</a>	y	

### Returns

TYPE	DESCRIPTION
<a href="#">Vector3Int</a>	

**FromSide(int)**

Declaration

```
public static HexPrismFaceDir FromSide(int side)
```

Parameters

TYPE	NAME	DESCRIPTION
int	side	

Returns

TYPE	DESCRIPTION
HexPrismFaceDir	

GetCellCenter(Vector3Int, Vector3, Vector3)

Declaration

```
public static Vector3 GetCellCenter(Vector3Int cell, Vector3 origin, Vector3 cellSize)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	cell	
Vector3	origin	
Vector3	cellSize	

Returns

TYPE	DESCRIPTION
Vector3	

Rotate(HexRotation, Vector3Int)

Declaration

```
public static Vector3Int Rotate(HexRotation rotation, Vector3Int cell)
```

Parameters

TYPE	NAME	DESCRIPTION
HexRotation	rotation	
Vector3Int	cell	

Returns

TYPE	DESCRIPTION
Vector3Int	

ToCubeCoords(Vector3Int)

## Declaration

```
public static Vector3Int ToCubeCoords(Vector3Int cell)
```

## Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	cell	

## Returns

TYPE	DESCRIPTION
Vector3Int	

# Class HexPrismDirExtensions

## NOTE

This class is available only in Tessera Pro

## Inheritance

[object](#)

[HexPrismDirExtensions](#)

Namespace: [Tessera](#)

Assembly: Tessera.dll

## Syntax

```
public static class HexPrismDirExtensions
```

## Methods

[Forward\(SylvesHexPrismDir\)](#)

### Declaration

```
public static Vector3 Forward(this SylvesHexPrismDir dir)
```

### Parameters

TYPE	NAME	DESCRIPTION
<a href="#">SylvesHexPrismDir</a>	dir	

### Returns

TYPE	DESCRIPTION
<a href="#">Vector3</a>	

[GetSide\(SylvesHexPrismDir\)](#)

### Declaration

```
public static int GetSide(this SylvesHexPrismDir dir)
```

### Parameters

TYPE	NAME	DESCRIPTION
<a href="#">SylvesHexPrismDir</a>	dir	

### Returns

TYPE	DESCRIPTION
<a href="#">int</a>	

[IsUpDown\(SylvesHexPrismDir\)](#)

### Declaration

```
public static bool IsUpDown(this SylvesHexPrismDir dir)
```

Parameters

TYPE	NAME	DESCRIPTION
SylvesHexPrismDir	dir	

Returns

TYPE	DESCRIPTION
bool	

## Up(SylvesHexPrismDir)

Declaration

```
public static Vector3 Up(this SylvesHexPrismDir dir)
```

Parameters

TYPE	NAME	DESCRIPTION
SylvesHexPrismDir	dir	

Returns

TYPE	DESCRIPTION
Vector3	

# Enum HexPrismFaceDir

Enum of the 8 faces on a hex prism.

## NOTE

This class is available only in Tessera Pro

Namespace: [Tessera](#)

Assembly: Tessera.dll

Syntax

```
public enum HexPrismFaceDir
```

Fields

NAME	DESCRIPTION
BackLeft	
BackRight	
Down	
ForwardLeft	
ForwardRight	
Left	
Right	
Up	

Extension Methods

[HexPrismFaceDirExtensions.Forward\(HexPrismFaceDir\)](#)  
[HexPrismFaceDirExtensions.ForwardInt\(HexPrismFaceDir\)](#)  
[HexPrismFaceDirExtensions.GetSide\(HexPrismFaceDir\)](#)  
[HexPrismFaceDirExtensions.IsUpDown\(HexPrismFaceDir\)](#)  
[HexPrismFaceDirExtensions.Up\(HexPrismFaceDir\)](#)

# Class HexPrismFaceDirExtensions

## NOTE

This class is available only in Tessera Pro

## Inheritance

[object](#)

[HexPrismFaceDirExtensions](#)

Namespace: [Tessera](#)

Assembly: Tessera.dll

## Syntax

```
public static class HexPrismFaceDirExtensions
```

## Methods

[Forward\(HexPrismFaceDir\)](#)

### Declaration

```
public static Vector3 Forward(this HexPrismFaceDir dir)
```

### Parameters

TYPE	NAME	DESCRIPTION
<a href="#">HexPrismFaceDir</a>	dir	

### Returns

TYPE	DESCRIPTION
Vector3	

[ForwardInt\(HexPrismFaceDir\)](#)

### Declaration

```
public static Vector3Int ForwardInt(this HexPrismFaceDir dir)
```

### Parameters

TYPE	NAME	DESCRIPTION
<a href="#">HexPrismFaceDir</a>	dir	

### Returns

TYPE	DESCRIPTION
Vector3Int	

[GetSide\(HexPrismFaceDir\)](#)

### Declaration

```
public static int GetSide(this HexPrismFaceDir dir)
```

## Parameters

TYPE	NAME	DESCRIPTION
HexPrismFaceDir	dir	

## Returns

TYPE	DESCRIPTION
int	

## IsUpDown(HexPrismFaceDir)

### Declaration

```
public static bool IsUpDown(this HexPrismFaceDir dir)
```

## Parameters

TYPE	NAME	DESCRIPTION
HexPrismFaceDir	dir	

## Returns

TYPE	DESCRIPTION
bool	

## Up(HexPrismFaceDir)

### Declaration

```
public static Vector3 Up(this HexPrismFaceDir dir)
```

## Parameters

TYPE	NAME	DESCRIPTION
HexPrismFaceDir	dir	

## Returns

TYPE	DESCRIPTION
Vector3	

# Struct HexRotation

Represents rotations / reflections of a hexagon

## NOTE

This class is available only in Tessera Pro

Inherited Members

### ValueType.ToString()

Namespace: [Tessera](#)

Assembly: Tessera.dll

Syntax

```
public struct HexRotation
```

## Properties

### All

Declaration

```
public static HexRotation[] All { get; }
```

Property Value

TYPE	DESCRIPTION
<a href="#">HexRotation</a> []	

### Identity

Declaration

```
public static HexRotation Identity { get; }
```

Property Value

TYPE	DESCRIPTION
<a href="#">HexRotation</a>	

### IsReflection

Declaration

```
public bool IsReflection { get; }
```

Property Value

TYPE	DESCRIPTION
<a href="#">bool</a>	

### ReflectForwardLeft

Declaration

```
public static HexRotation ReflectForwardLeft { get; }
```

## Property Value

TYPE	DESCRIPTION
HexRotation	

## ReflectForwardRight

### Declaration

```
public static HexRotation ReflectForwardRight { get; }
```

## Property Value

TYPE	DESCRIPTION
HexRotation	

## ReflectX

### Declaration

```
public static HexRotation ReflectX { get; }
```

## Property Value

TYPE	DESCRIPTION
HexRotation	

## ReflectZ

### Declaration

```
public static HexRotation ReflectZ { get; }
```

## Property Value

TYPE	DESCRIPTION
HexRotation	

## RotateCCW

### Declaration

```
public static HexRotation RotateCCW { get; }
```

## Property Value

TYPE	DESCRIPTION
HexRotation	

## Rotation

### Declaration

```
public int Rotation { get; }
```

## Property Value

TYPE	DESCRIPTION
int	

## Methods

### Equals(object)

Declaration

```
public override bool Equals(object obj)
```

Parameters

TYPE	NAME	DESCRIPTION
object	obj	

Returns

TYPE	DESCRIPTION
bool	

Overrides

[ValueType.Equals\(object\)](#)

### GetHashCode()

Declaration

```
public override int GetHashCode()
```

Returns

TYPE	DESCRIPTION
int	

Overrides

[ValueType.GetHashCode\(\)](#)

### Invert()

Declaration

```
public HexRotation Invert()
```

Returns

TYPE	DESCRIPTION
HexRotation	

### Rotate60(int)

Declaration

```
public static HexRotation Rotate60(int i)
```

Parameters

TYPE	NAME	DESCRIPTION
int	i	

Returns

TYPE	DESCRIPTION
HexRotation	

## Operators

**operator ==(HexRotation, HexRotation)**

Declaration

```
public static bool operator ==(HexRotation a, HexRotation b)
```

Parameters

TYPE	NAME	DESCRIPTION
HexRotation	a	
HexRotation	b	

Returns

TYPE	DESCRIPTION
bool	

**implicit operator HexRotation(CellRotation)**

Declaration

```
public static implicit operator HexRotation(CellRotation r)
```

Parameters

TYPE	NAME	DESCRIPTION
CellRotation	r	

Returns

TYPE	DESCRIPTION
HexRotation	

**implicit operator CellRotation(HexRotation)**

Declaration

```
public static implicit operator CellRotation(HexRotation r)
```

Parameters

TYPE	NAME	DESCRIPTION
HexRotation	r	

Returns

TYPE	DESCRIPTION
CellRotation	

**operator !=(HexRotation, HexRotation)**

Declaration

```
public static bool operator !=(HexRotation a, HexRotation b)
```

Parameters

TYPE	NAME	DESCRIPTION
HexRotation	a	
HexRotation	b	

Returns

TYPE	DESCRIPTION
bool	

**operator \*(HexRotation, int)**

Declaration

```
public static int operator *(HexRotation a, int side)
```

Parameters

TYPE	NAME	DESCRIPTION
HexRotation	a	
int	side	

Returns

TYPE	DESCRIPTION
int	

**operator \*(HexRotation, HexPrismFaceDir)**

Declaration

```
public static HexPrismFaceDir operator *(HexRotation rotation, HexPrismFaceDir faceDir)
```

Parameters

TYPE	NAME	DESCRIPTION
HexRotation	rotation	
HexPrismFaceDir	faceDir	

Returns

TYPE	DESCRIPTION
HexPrismFaceDir	

## operator \*(HexRotation, HexRotation)

Declaration

```
public static HexRotation operator *(HexRotation a, HexRotation b)
```

Parameters

TYPE	NAME	DESCRIPTION
HexRotation	a	
HexRotation	b	

Returns

TYPE	DESCRIPTION
HexRotation	

# Interface IEngineInterface

Namespace: [Tessera](#)

Assembly: Tessera.dll

## Syntax

```
public interface IEngineInterface
```

## Methods

### Destroy(Object)

Declaration

```
void Destroy(Object o)
```

#### Parameters

TYPE	NAME	DESCRIPTION
Object	o	

### Instantiate(GameObject, Vector3, Quaternion, Transform)

Declaration

```
GameObject Instantiate(GameObject gameObject, Vector3 position, Quaternion rotation, Transform parent)
```

#### Parameters

TYPE	NAME	DESCRIPTION
GameObject	gameObject	
Vector3	position	
Quaternion	rotation	
Transform	parent	

#### Returns

TYPE	DESCRIPTION
GameObject	

### RegisterCompleteObjectUndo(Object)

Declaration

```
void RegisterCompleteObjectUndo(Object objectToUndo)
```

#### Parameters

TYPE	NAME	DESCRIPTION
Object	objectToUndo	

### RegisterCreatedObjectUndo(Object)

## Declaration

```
void RegisterCreatedObjectUndo(Object objectToUndo)
```

## Parameters

TYPE	NAME	DESCRIPTION
Object	objectToUndo	

# Interface ISubFaceInfo

Namespace: [Tessera](#)

Assembly: Tessera.dll

Syntax

```
public interface ISubFaceInfo
```

## Methods

**GetCorner(Vector3Int, CellDir, SubFaceTuple)**

Declaration

```
CellCorner GetCorner(Vector3Int offset, CellDir dir, SubFaceTuple subFace)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	offset	
CellDir	dir	
<a href="#">SubFaceTuple</a>	subFace	

Returns

TYPE	DESCRIPTION
<a href="#">CellCorner</a>	

**GetSubFaces(Vector3Int, CellDir)**

Declaration

```
IEnumerable<SubFaceTuple> GetSubFaces(Vector3Int offset, CellDir dir)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	offset	
CellDir	dir	

Returns

TYPE	DESCRIPTION
<a href="#">IEnumerable&lt;SubFaceTuple&gt;</a>	

# Interface ITesseraInitialConstraint

Namespace: [Tessera](#)

Assembly: Tessera.dll

Syntax

```
public interface ITesseraInitialConstraint
```

Properties

Name

Declaration

```
string Name { get; }
```

Property Value

TYPE	DESCRIPTION
<a href="#">string</a>	

# Interface ITesseraTileOutput

Namespace: [Tessera](#)

Assembly: Tessera.dll

Syntax

```
public interface ITesseraTileOutput
```

Properties

IsEmpty

Is the output currently empty.

Declaration

```
bool IsEmpty { get; }
```

Property Value

TYPE	DESCRIPTION
bool	

SupportsIncremental

Is this output safe to use with AnimatedGenerator

Declaration

```
bool SupportsIncremental { get; }
```

Property Value

TYPE	DESCRIPTION
bool	

Methods

**ClearTiles(IEngineInterface)**

Clear the output

Declaration

```
void ClearTiles(IEngineInterface engine)
```

Parameters

TYPE	NAME	DESCRIPTION
IEngineInterface	engine	

**UpdateTiles(TesseraCompletion, IEngineInterface)**

Update a chunk of tiles. If incremental updates are supported, then:

- Tiles can replace other tiles, as indicated by the [Cells](#) field.
- A tile of null indicates that the tile should be erased

## Declaration

```
void UpdateTiles(TesseraCompletion completion, IEngineInterface engine)
```

## Parameters

TYPE	NAME	DESCRIPTION
TesseraCompletion	completion	
IEngineInterface	engine	

# Class InfiniteGenerator

Inheritance

[object](#)

[Object](#)

[Component](#)

[Behaviour](#)

[MonoBehaviour](#)

[InfiniteGenerator](#)

Inherited Members

[MonoBehaviour.IsInvoking\(\)](#)

[MonoBehaviour.CancelInvoke\(\)](#)

[MonoBehaviour.Invoke\(string, float\)](#)

[MonoBehaviour.InvokeRepeating\(string, float, float\)](#)

[MonoBehaviour.CancelInvoke\(string\)](#)

[MonoBehaviour.IsInvoking\(string\)](#)

[MonoBehaviour.StartCoroutine\(string\)](#)

[MonoBehaviour.StartCoroutine\(string, object\)](#)

[MonoBehaviour.StartCoroutine\(IEnumerator\)](#)

[MonoBehaviour.StartCoroutine\\_Auto\(IEnumerator\)](#)

[MonoBehaviour.StopCoroutine\(IEnumerator\)](#)

[MonoBehaviour.StopCoroutine\(Coroutine\)](#)

[MonoBehaviour.StopCoroutine\(string\)](#)

[MonoBehaviour.StopAllCoroutines\(\)](#)

[MonoBehaviour.print\(object\)](#)

[MonoBehaviour.useGUILayout](#)

[MonoBehaviour.runInEditMode](#)

[Behaviour.enabled](#)

[Behaviour.isActiveAndEnabled](#)

[Component.GetComponent\(Type\)](#)

[Component.GetComponent<T>\(\)](#)

[Component.TryGetComponent\(Type, out Component\)](#)

[Component.TryGetComponent<T>\(out T\)](#)

[Component.GetComponent\(string\)](#)

[Component.GetComponentInChildren\(Type, bool\)](#)

[Component.GetComponentInChildren\(Type\)](#)

[Component.GetComponentInChildren<T>\(bool\)](#)

[Component.GetComponentInChildren<T>\(\)](#)

[Component.GetComponentsInChildren\(Type, bool\)](#)

[Component.GetComponentsInChildren\(Type\)](#)

[Component.GetComponentsInChildren<T>\(bool\)](#)

[Component.GetComponentsInChildren<T>\(bool, List<T>\)](#)

[Component.GetComponentsInChildren<T>\(\)](#)

[Component.GetComponentsInChildren<T>\(List<T>\)](#)

[Component.GetComponentInParent\(Type\)](#)

[Component.GetComponentInParent<T>\(\)](#)

[Component.GetComponentsInParent\(Type, bool\)](#)

[Component.GetComponentsInParent\(Type\)](#)

[Component.GetComponentsInParent<T>\(bool\)](#)

[Component.GetComponentsInParent<T>\(bool, List<T>\)](#)

[Component.GetComponentsInParent<T>\(\)](#)

Component.GetComponents(Type)  
Component.GetComponents(Type, List<Component>)  
Component.GetComponents<T>(List<T>)  
Component.GetComponents<T>()  
Component.CompareTag(string)  
Component.SendMessageUpwards(string, object, SendMessageOptions)  
Component.SendMessageUpwards(string, object)  
Component.SendMessageUpwards(string)  
Component.SendMessageUpwards(string, SendMessageOptions)  
Component.SendMessage(string, object)  
Component.SendMessage(string)  
Component.SendMessage(string, object, SendMessageOptions)  
Component.SendMessage(string, SendMessageOptions)  
Component.BroadcastMessage(string, object, SendMessageOptions)  
Component.BroadcastMessage(string, object)  
Component.BroadcastMessage(string)  
Component.BroadcastMessage(string, SendMessageOptions)  
Component.transform  
Component.gameObject  
Component.tag  
Object.GetInstanceID()  
Object.GetHashCode()  
**Object.Equals(object)**  
Object.Instantiate(Object, Vector3, Quaternion)  
Object.Instantiate(Object, Vector3, Quaternion, Transform)  
Object.Instantiate(Object)  
Object.Instantiate(Object, Transform)  
**Object.Instantiate(Object, Transform, bool)**  
Object.Instantiate<T>(T)  
Object.Instantiate<T>(T, Vector3, Quaternion)  
Object.Instantiate<T>(T, Vector3, Quaternion, Transform)  
Object.Instantiate<T>(T, Transform)  
**Object.Instantiate<T>(T, Transform, bool)**  
Object.Destroy(Object, float)  
Object.Destroy(Object)  
**Object.DestroyImmediate(Object, bool)**  
Object.DestroyImmediate(Object)  
**Object.FindObjectsOfType(Type)**  
Object.DontDestroyOnLoad(Object)  
**Object.DestroyObject(Object, float)**  
Object.DestroyObject(Object)  
**Object.FindSceneObjectsOfType(Type)**  
Object.FindObjectsOfTypeIncludingAssets(Type)  
Object.FindObjectsOfType<T>()  
Object.FindObjectOfType<T>()  
**Object.FindObjectsOfTypeAll(Type)**  
**Object.FindObjectOfType(Type)**  
Object.ToString()  
Object.name  
Object.hideFlags

Namespace: **Tessera**

Assembly: Tessera.dll

## Syntax

```
public class InfiniteGenerator : MonoBehaviour
```

## Fields

### chunkCleanupType

Determines what to do with unused chunks

#### Declaration

```
public ChunkCleanupType chunkCleanupType
```

#### Field Value

TYPE	DESCRIPTION
ChunkCleanupType	

### chunkPersistTime

Time a chunk is kept around for even if not near a watched collider.

#### Declaration

```
public float chunkPersistTime
```

#### Field Value

TYPE	DESCRIPTION
float	

### cleanupInterval

Time between cleanups that remove chunks that are no longer needed.

#### Declaration

```
public float cleanupInterval
```

#### Field Value

TYPE	DESCRIPTION
float	

### generator

The generator that is used to fill the chunks. It also determines the size of each chunk.

#### Declaration

```
public TesseraGenerator generator
```

#### Field Value

TYPE	DESCRIPTION
TesseraGenerator	

## infiniteX

If true, chunks repeat infinitely on this axis. If false, you can specify the [minXChunk/maxXChunk](#) to give a limit to the amount of chunks generated.

Declaration

```
public bool infiniteX
```

Field Value

TYPE	DESCRIPTION
bool	

## infiniteY

If true, chunks repeat infinitely on this axis. If false, you can specify the [minYChunk/maxYChunk](#) to give a limit to the amount of chunks generated.

Declaration

```
public bool infiniteY
```

Field Value

TYPE	DESCRIPTION
bool	

## infiniteZ

If true, chunks repeat infinitely on this axis. If false, you can specify the [minZChunk/maxZChunk](#) to give a limit to the amount of chunks generated.

Declaration

```
public bool infiniteZ
```

Field Value

TYPE	DESCRIPTION
bool	

## maxCleanupPerUpdate

Maximum number of chunks to remove per update. Zero means unbounded.

Declaration

```
public int maxCleanupPerUpdate
```

Field Value

TYPE	DESCRIPTION
int	

## maxInstantiatePerUpdate

Maximum number of tiles to create per-update. Negative means unbounded.

Declaration

```
public float maxInstantiatePerUpdate
```

Field Value

TYPE	DESCRIPTION
float	

maxXChunk

Declaration

```
public int maxXChunk
```

Field Value

TYPE	DESCRIPTION
int	

maxYChunk

Declaration

```
public int maxYChunk
```

Field Value

TYPE	DESCRIPTION
int	

maxZChunk

Declaration

```
public int maxZChunk
```

Field Value

TYPE	DESCRIPTION
int	

minXChunk

Declaration

```
public int minXChunk
```

Field Value

TYPE	DESCRIPTION
int	

minYChunk

Declaration

```
public int minYChunk
```

Field Value

TYPE	DESCRIPTION
int	

### minZChunk

Declaration

```
public int minZChunk
```

Field Value

TYPE	DESCRIPTION
int	

### parallelism

The number of chunks that can be generated concurrently. Note that turning this up can cause slightly worse quality output.

Declaration

```
public int parallelism
```

Field Value

TYPE	DESCRIPTION
int	

### scanInterval

Time between scans that detect if new chunks need creating

Declaration

```
public float scanInterval
```

Field Value

TYPE	DESCRIPTION
float	

### seed

Fixes the seed for random number generator. If the value is zero, the seed is taken from Unity.Random. Note that generation is still non-deterministic even with a fixed seed.

Declaration

```
public int seed
```

Field Value

TYPE	DESCRIPTION
int	

## strip

Shrink the generated output by this many cells, effectively giving the generator more "lookahead".

Increasing this can help avoid contradictions generating chunks.

### Declaration

```
public int strip
```

### Field Value

TYPE	DESCRIPTION
int	

## watchedColliders

Determines the volume in which chunks should be generated. You typically want to use a large trigger collider following the player or camera, to ensure that everything nearby is generated.

### Declaration

```
public List<Collider> watchedColliders
```

### Field Value

TYPE	DESCRIPTION
List<Collider>	

## Methods

### IsOverlapCell(Vector3Int)

#### Declaration

```
public bool IsOverlapCell(Vector3Int cell)
```

#### Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	cell	

#### Returns

TYPE	DESCRIPTION
bool	

# Class InstantiateOutput

Inheritance

[object](#)

[InstantiateOutput](#)

Implements

[ITesseraTileOutput](#)

Namespace: [Tessera](#)

Assembly: Tessera.dll

Syntax

```
public class InstantiateOutput : ITesseraTileOutput
```

Constructors

[InstantiateOutput\(Transform\)](#)

Declaration

```
public InstantiateOutput(Transform transform)
```

Parameters

TYPE	NAME	DESCRIPTION
Transform	transform	

Properties

[IsEmpty](#)

Is the output currently empty.

Declaration

```
public bool IsEmpty { get; }
```

Property Value

TYPE	DESCRIPTION
<a href="#">bool</a>	

[SupportsIncremental](#)

Is this output safe to use with AnimatedGenerator

Declaration

```
public bool SupportsIncremental { get; }
```

Property Value

TYPE	DESCRIPTION
<a href="#">bool</a>	

Methods

## ClearTiles(IEngineInterface)

Clear the output

Declaration

```
public void ClearTiles(IEngineInterface engine)
```

Parameters

TYPE	NAME	DESCRIPTION
IEngineInterface	engine	

## UpdateTiles(TesseraCompletion, IEngineInterface)

Update a chunk of tiles. If incremental updates are supported, then:

- Tiles can replace other tiles, as indicated by the [Cells](#) field.
- A tile of null indicates that the tile should be erased

Declaration

```
public void UpdateTiles(TesseraCompletion completion, IEngineInterface engine)
```

Parameters

TYPE	NAME	DESCRIPTION
TesseraCompletion	completion	
IEngineInterface	engine	

Implements

[ITesseraTileOutput](#)

# Class MeshUtils

Utility for working with meshes.

## NOTE

This class is available only in Tessera Pro

Inheritance

[object](#)

[MeshUtils](#)

Namespace: [Tessera](#)

Assembly: Tessera.dll

Syntax

```
public static class MeshUtils
```

## Methods

[TransformRecursively\(GameObject, Deformation\)](#)

Applies Transform gameObject and its children. Components affected:

- MeshFilter
- MeshCollider
- BoxCollider

Declaration

```
public static void TransformRecursively(GameObject gameObject, Deformation meshDeformation)
```

Parameters

TYPE	NAME	DESCRIPTION
GameObject	gameObject	
Deformation	meshDeformation	

# Class MirrorConstraint

Ensures that the generation is symmetric when x-axis mirrored. If there are any tile constraints, they will not be mirrored.

## NOTE

This class is available only in Tessera Pro

Inheritance

[object](#)

[Object](#)

[Component](#)

[Behaviour](#)

[MonoBehaviour](#)

[TesseraConstraint](#)

[MirrorConstraint](#)

Inherited Members

[MonoBehaviour.IsInvoking\(\)](#)

[MonoBehaviour.CancelInvoke\(\)](#)

[MonoBehaviour.Invoke\(string, float\)](#)

[MonoBehaviour.InvokeRepeating\(string, float, float\)](#)

[MonoBehaviour.CancelInvoke\(string\)](#)

[MonoBehaviour.IsInvoking\(string\)](#)

[MonoBehaviour.StartCoroutine\(string\)](#)

[MonoBehaviour.StartCoroutine\(string, object\)](#)

[MonoBehaviour.StartCoroutine\(IEnumerator\)](#)

[MonoBehaviour.StartCoroutine\\_Auto\(IEnumerator\)](#)

[MonoBehaviour.StopCoroutine\(IEnumerator\)](#)

[MonoBehaviour.StopCoroutine\(Coroutine\)](#)

[MonoBehaviour.StopCoroutine\(string\)](#)

[MonoBehaviour.StopAllCoroutines\(\)](#)

[MonoBehaviour.print\(object\)](#)

[MonoBehaviour.useGUILayout](#)

[MonoBehaviour.runInEditMode](#)

[Behaviour.enabled](#)

[Behaviour.isActiveAndEnabled](#)

[Component.GetComponent\(Type\)](#)

[Component.GetComponent<T>\(\)](#)

[Component.TryGetComponent\(Type, out Component\)](#)

[Component.TryGetComponent<T>\(out T\)](#)

[Component.GetComponent\(string\)](#)

[Component.GetComponentInChildren\(Type, bool\)](#)

[Component.GetComponentInChildren\(Type\)](#)

[Component.GetComponentInChildren<T>\(bool\)](#)

[Component.GetComponentInChildren<T>\(\)](#)

[Component.GetComponentsInChildren\(Type, bool\)](#)

[Component.GetComponentsInChildren\(Type\)](#)

[Component.GetComponentsInChildren<T>\(bool\)](#)

[Component.GetComponentsInChildren<T>\(bool, List<T>\)](#)

[Component.GetComponentsInChildren<T>\(\)](#)

[Component.GetComponentsInChildren<T>\(List<T>\)](#)

Component.GetComponentInParent(Type)  
Component.GetComponentInParent<T>()  
Component.GetComponentsInParent(Type, bool)  
Component.GetComponentsInParent(Type)  
Component.GetComponentsInParent<T>(bool)  
Component.GetComponentsInParent<T>(bool, List<T>)  
Component.GetComponentsInParent<T>()  
Component.GetComponents(Type)  
Component.GetComponents(Type, List<Component>)  
Component.GetComponents<T>(List<T>)  
Component.GetComponents<T>()  
Component.CompareTag(string)  
Component.SendMessageUpwards(string, object, SendMessageOptions)  
Component.SendMessageUpwards(string, object)  
Component.SendMessageUpwards(string)  
Component.SendMessageUpwards(string, SendMessageOptions)  
Component.SendMessage(string, object)  
Component.SendMessage(string)  
Component.SendMessage(string, object, SendMessageOptions)  
Component.SendMessage(string, SendMessageOptions)  
Component.BroadcastMessage(string, object, SendMessageOptions)  
Component.BroadcastMessage(string, object)  
Component.BroadcastMessage(string)  
Component.BroadcastMessage(string, SendMessageOptions)  
Component.transform  
Component.gameObject  
Component.tag  
Object.GetInstanceID()  
Object.GetHashCode()  
**Object.Equals(object)**  
Object.Instantiate(Object, Vector3, Quaternion)  
Object.Instantiate(Object, Vector3, Quaternion, Transform)  
Object.Instantiate(Object)  
Object.Instantiate(Object, Transform)  
**Object.Instantiate(Object, Transform, bool)**  
Object.Instantiate<T>(T)  
Object.Instantiate<T>(T, Vector3, Quaternion)  
Object.Instantiate<T>(T, Vector3, Quaternion, Transform)  
Object.Instantiate<T>(T, Transform)  
**Object.Instantiate<T>(T, Transform, bool)**  
Object.Destroy(Object, float)  
Object.Destroy(Object)  
Object.DestroyImmediate(Object, bool)  
Object.DestroyImmediate(Object)  
**Object.FindObjectsOfType(Type)**  
Object.DontDestroyOnLoad(Object)  
**Object.DestroyObject(Object, float)**  
Object.DestroyObject(Object)  
**Object.FindSceneObjectsOfType(Type)**  
**Object.FindObjectsOfTypeIncludingAssets(Type)**  
Object.FindObjectsOfType<T>()  
ObjectFindObjectOfType<T>()

[Object.FindObjectsOfTypeAll\(Type\)](#)

[Object.FindObjectOfType\(Type\)](#)

[Object.ToString\(\)](#)

[Object.name](#)

[Object.hideFlags](#)

Namespace: [Tessera](#)

Assembly: Tessera.dll

## Syntax

```
[AddComponentMenu("Tessera/Mirror Constraint", 20)]  
[RequireComponent(typeof(TesseraGenerator))]  
public class MirrorConstraint : TesseraConstraint
```

## Fields

[axis](#)

Declaration

```
public MirrorConstraint.Axis axis
```

## Field Value

TYPE	DESCRIPTION
<a href="#">MirrorConstraint.Axis</a>	

# Enum MirrorConstraint.Axis

Namespace: [Tessera](#)

Assembly: Tessera.dll

## Syntax

```
public enum MirrorConstraint.Axis
```

## Fields

NAME	DESCRIPTION
W	
X	
Y	
Z	

# Struct ModelTile

Actual tiles used internally. There's a many-to-one relationship between ModelTile and TesseraTile due to rotations and "big" tile support.

Namespace: [Tessera](#)

Assembly: Tessera.dll

Syntax

```
public struct ModelTile
```

Constructors

**ModelTile(TesseraTileBase, CellRotation, Vector3Int)**

Declaration

```
public ModelTile(TesseraTileBase tile, CellRotation rotation, Vector3Int offset)
```

Parameters

TYPE	NAME	DESCRIPTION
TesseraTileBase	tile	
CellRotation	rotation	
Vector3Int	offset	

Properties

Offset

Declaration

```
public Vector3Int Offset { get; set; }
```

Property Value

TYPE	DESCRIPTION
Vector3Int	

Rotation

Declaration

```
public CellRotation Rotation { get; set; }
```

Property Value

TYPE	DESCRIPTION
CellRotation	

Tile

Declaration

```
public TesseraTileBase Tile { get; set; }
```

## Property Value

Type	Description
TesseraTileBase	

## Methods

### Equals(object)

Declaration

```
public override bool Equals(object obj)
```

#### Parameters

Type	Name	Description
object	obj	

#### Returns

Type	Description
bool	

#### Overrides

[ValueType.Equals\(object\)](#)

### Equals(ModelTile)

Declaration

```
public bool Equals(ModelTile other)
```

#### Parameters

Type	Name	Description
ModelTile	other	

#### Returns

Type	Description
bool	

### GetHashCode()

Declaration

```
public override int GetHashCode()
```

#### Returns

Type	Description
int	

#### Overrides

[ValueType.GetHashCode\(\)](#)

## ToString()

### Declaration

```
public override string ToString()
```

### Returns

TYPE	DESCRIPTION
string	

### Overrides

[ValueType.ToString\(\)](#)

# Enum ModelType

Different models Tessera supports. The model dictates how nearby tiles relate to each other.

Namespace: [Tessera](#)

Assembly: Tessera.dll

Syntax

```
public enum ModelType
```

Fields

NAME	DESCRIPTION
Adjacent	See <a href="#">overlapping</a> .
AdjacentPaint	The default model using the painting system to determine tile adjacencies.
MeshEdges	Detects where mesh edges line up.
Overlapping	See <a href="#">overlapping</a> .

# Struct OrientedFace

Legacy class used when serializing. You should use SylvesOrientedFace instead. Records the painted colors and location of single face of one cube in a [TesseraTile](#)

Inherited Members

[ValueType.Equals\(object\)](#)  
[ValueType.GetHashCode\(\)](#)  
[ValueType.ToString\(\)](#)

Namespace: [Tessera](#)

Assembly: Tessera.dll

Syntax

```
[Serializable]
public struct OrientedFace
```

Constructors

[OrientedFace\(Vector3Int, CellFaceDir, FaceDetails\)](#)

Declaration

```
public OrientedFace(Vector3Int offset, CellFaceDir faceDir, FaceDetails faceDetails)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	offset	
<a href="#">CellFaceDir</a>	faceDir	
<a href="#">FaceDetails</a>	faceDetails	

Fields

[faceDetails](#)

Declaration

```
public FaceDetails faceDetails
```

Field Value

TYPE	DESCRIPTION
<a href="#">FaceDetails</a>	

[faceDir](#)

Declaration

```
public CellFaceDir faceDir
```

Field Value

TYPE	DESCRIPTION
<a href="#">CellFaceDir</a>	

## offset

Declaration

```
public Vector3Int offset
```

Field Value

TYPE	DESCRIPTION
Vector3Int	

## Methods

Deconstruct(out Vector3Int, out CellFaceDir, out FaceDetails)

Declaration

```
public void Deconstruct(out Vector3Int offset, out CellFaceDir faceDir, out FaceDetails faceDetails)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	offset	
CellFaceDir	faceDir	
FaceDetails	faceDetails	

# Class PaletteEntry

Inheritance

[object](#)

[PaletteEntry](#)

Namespace: [Tessera](#)

Assembly: Tessera.dll

Syntax

```
[Serializable]
public class PaletteEntry
```

Fields

**color**

Declaration

```
public Color color
```

Field Value

TYPE	DESCRIPTION
Color	

**name**

Declaration

```
public string name
```

Field Value

TYPE	DESCRIPTION
string	

# Class PathConstraint

Forces a network of tiles to connect with each other, so there is always a complete path between them. Two tiles connect along the path if:

- Both tiles are in `pathTiles` (if `hasPathTiles` set); and
- The central color of the sides of the tiles leading to each other are in `pathColors` (if `pathColors` set)

## NOTE

This class is available only in Tessera Pro

Inheritance

`object`

`Object`

`Component`

`Behaviour`

`MonoBehaviour`

`TesseraConstraint`

`PathConstraint`

Inherited Members

`MonoBehaviour.IsInvoking()`

`MonoBehaviour.CancelInvoke()`

`MonoBehaviour.Invoke(string, float)`

`MonoBehaviour.InvokeRepeating(string, float, float)`

`MonoBehaviour.CancelInvoke(string)`

`MonoBehaviour.IsInvoking(string)`

`MonoBehaviour.StartCoroutine(string)`

`MonoBehaviour.StartCoroutine(string, object)`

`MonoBehaviour.StartCoroutine(IEnumerator)`

`MonoBehaviour.StartCoroutine_Auto(IEnumerator)`

`MonoBehaviour.StopCoroutine(IEnumerator)`

`MonoBehaviour.StopCoroutine(Coroutine)`

`MonoBehaviour.StopCoroutine(string)`

`MonoBehaviour.StopAllCoroutines()`

`MonoBehaviour.print(object)`

`MonoBehaviour.useGUILayout`

`MonoBehaviour.runInEditMode`

`Behaviour.enabled`

`Behaviour.isActiveAndEnabled`

`Component.GetComponent(Type)`

`Component.GetComponent<T>()`

`Component.TryGetComponent(Type, out Component)`

`Component.TryGetComponent<T>(out T)`

`Component.GetComponent(string)`

`Component.GetComponentInChildren(Type, bool)`

`Component.GetComponentInChildren(Type)`

`Component.GetComponentInChildren<T>(bool)`

`Component.GetComponentInChildren<T>()`

`Component.GetComponentsInChildren(Type, bool)`

`Component.GetComponentsInChildren(Type)`

`Component.GetComponentsInChildren<T>(bool)`

```
Component.GetComponentsInChildren<T>(bool, List<T>)
Component.GetComponentsInChildren<T>()
Component.GetComponentsInChildren<T>(List<T>)
Component.GetComponentInParent(Type)
Component.GetComponentInParent<T>()
Component.GetComponentsInParent(Type, bool)
Component.GetComponentsInParent(Type)
Component.GetComponentsInParent<T>(bool)
Component.GetComponentsInParent<T>(bool, List<T>)
Component.GetComponentsInParent<T>()
Component.GetComponents(Type)
Component.GetComponents(Type, List<Component>)
Component.GetComponents<T>(List<T>)
Component.GetComponents<T>()
Component.CompareTag(string)
Component.SendMessageUpwards(string, object, SendMessageOptions)
Component.SendMessageUpwards(string, object)
Component.SendMessageUpwards(string)
Component.SendMessageUpwards(string, SendMessageOptions)
Component.SendMessage(string, object)
Component.SendMessage(string)
Component.SendMessage(string, object, SendMessageOptions)
Component.SendMessage(string, SendMessageOptions)
Component.BroadcastMessage(string, object, SendMessageOptions)
Component.BroadcastMessage(string, object)
Component.BroadcastMessage(string)
Component.BroadcastMessage(string, SendMessageOptions)
Component.transform
Component.gameObject
Component.tag
Object.GetInstanceID()
Object.GetHashCode()
Object.Equals(object)
Object.Instantiate(Object, Vector3, Quaternion)
Object.Instantiate(Object, Vector3, Quaternion, Transform)
Object.Instantiate(Object)
Object.Instantiate(Object, Transform)
Object.Instantiate(Object, Transform, bool)
Object.Instantiate<T>(T)
Object.Instantiate<T>(T, Vector3, Quaternion)
Object.Instantiate<T>(T, Vector3, Quaternion, Transform)
Object.Instantiate<T>(T, Transform)
Object.Instantiate<T>(T, Transform, bool)
Object.Destroy(Object, float)
Object.Destroy(Object)
Object.DestroyImmediate(Object, bool)
Object.DestroyImmediate(Object)
Object.FindObjectsOfType(Type)
Object.DontDestroyOnLoad(Object)
Object.DestroyObject(Object, float)
Object.DestroyObject(Object)
Object.FindSceneObjectsOfType(Type)
```

[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#)

[Object.FindObjectsOfType<T>\(\)](#)

[Object.FindObjectOfType<T>\(\)](#)

[\*\*Object.FindObjectsOfTypeAll\(Type\)\*\*](#)

[Object FindObjectOfType\(Type\)](#)

[Object.ToString\(\)](#)

[Object.name](#)

[Object.hideFlags](#)

Namespace: [Tessera](#)

Assembly: Tessera.dll

## Syntax

```
[AddComponentMenu("Tessera/Path Constraint", 21)]
[RequireComponent(typeof(TesseraGenerator))]
public class PathConstraint : TesseraConstraint
```

## Fields

### acyclic

If set, bans all cycles, forcing a tree or forest.

#### Declaration

```
[Tooltip("If set, bans all cycles, forcing a the patch to branch like a tree or forest.")]
public bool acyclic
```

#### Field Value

TYPE	DESCRIPTION
<a href="#">bool</a>	

### connected

If Set, then the constraint forces that all path tiles must have a contiguous path between them.

#### Declaration

```
[Tooltip("If set, then the constraint forces that all path tiles must have a contiguous path between them.")]
public bool connected
```

#### Field Value

TYPE	DESCRIPTION
<a href="#">bool</a>	

### hasPathColors

If set, [pathColors](#) is used to determine path tiles and sides.

#### Declaration

```
[Tooltip("Enables a paint-based fitler for which tiles that the path can connect through")]
public bool hasPathColors
```

#### Field Value

<b>TYPE</b>	<b>DESCRIPTION</b>
bool	

## hasPathTiles

If set, [pathTiles](#) is used to determine path tiles.

Declaration

```
[Tooltip("Enables a filter for which tiles the path can connect through")]
public bool hasPathTiles
```

## Field Value

<b>TYPE</b>	<b>DESCRIPTION</b>
bool	

## loops

If set, forces there to be at least two non-overlapping valid paths between any two connected path tiles.

Declaration

```
[Tooltip("If set, forces there to be at least two non-overlapping valid paths between any two connected path
tiles.")]
public bool loops
```

## Field Value

<b>TYPE</b>	<b>DESCRIPTION</b>
bool	

## parity

Enable this if your path tileset includes no forks or junctions, it can improve the search quality.

Declaration

```
[Tooltip("Enable this if your path tileset includes no forks or junctions, it can improve the search
quality.")]
public bool parity
```

## Field Value

<b>TYPE</b>	<b>DESCRIPTION</b>
bool	

## pathColors

If [hasPathColors](#), this set filters tiles that the path can connect through. Only the central square on each face is inspected.

Declaration

```
[Tooltip("This set specifies the paint colors the path can connect through. Only the central square on each
face is inspected.")]
public List<int> pathColors
```

## Field Value

TYPE	DESCRIPTION
List<int>	

## pathTiles

If [hasPathTiles](#), this set filters tiles that the path can connect through.

### Declaration

```
[Tooltip("This set filters tiles that the path can connect through")]
public List<TesseraTileBase> pathTiles
```

## Field Value

TYPE	DESCRIPTION
List<TesseraTileBase>	

## prioritize

If set, the the generator will prefer generating tiles near the path.

### Declaration

```
[Tooltip("If set, the the generator will prefer generating tiles near the path.")]
public bool prioritize
```

## Field Value

TYPE	DESCRIPTION
bool	

# Enum PinType

Namespace: [Tessera](#)

Assembly: Tessera.dll

## Syntax

```
public enum PinType
```

## Fields

NAME	DESCRIPTION
FacesAndInterior	The faces of the pinned tile are used to constrain the cells adjacent to the location of the pinned tile and the cells covered by the pin tile are masked out so no tiles will be generated in that location.
FacesOnly	The faces of the pinned tile are used to constrain the cells adjacent to the location of the pinned tile.
Pin	Forces generation the pinned tile at the location of the pin.

# Class PrefixLookup<T>

Stores key-value pairs, with efficient searching for the longest key that is a prefix of a given string.

Inheritance

[object](#)

[PrefixLookup<T>](#)

Namespace: [Tessera](#)

Assembly: Tessera.dll

Syntax

```
public class PrefixLookup<T>
```

Type Parameters

NAME	DESCRIPTION
T	

Constructors

[PrefixLookup\(\)](#)

Declaration

```
public PrefixLookup()
```

Methods

[Add\(string, object\)](#)

Declaration

```
public void Add(string name, object value)
```

Parameters

TYPE	NAME	DESCRIPTION
<a href="#">string</a>	name	
<a href="#">object</a>	value	

[TryFindLongestPrefix\(string, out T\)](#)

Declaration

```
public bool TryFindLongestPrefix(string name, out T value)
```

Parameters

TYPE	NAME	DESCRIPTION
<a href="#">string</a>	name	
T	value	

Returns

TYPE	DESCRIPTION
bool	

# Enum RotationGroupType

Namespace: [Tessera](#)

Assembly: Tessera.dll

## Syntax

```
public enum RotationGroupType
```

## Fields

NAME	DESCRIPTION
All	
None	
XY	
XZ	
YZ	

# Class SeparationConstraint

Inheritance

[object](#)

[Object](#)

[Component](#)

[Behaviour](#)

[MonoBehaviour](#)

[TesseraConstraint](#)

[SeparationConstraint](#)

Inherited Members

[MonoBehaviour.IsInvoking\(\)](#)

[MonoBehaviour.CancelInvoke\(\)](#)

[MonoBehaviour.Invoke\(string, float\)](#)

[MonoBehaviour.InvokeRepeating\(string, float, float\)](#)

[MonoBehaviour.CancelInvoke\(string\)](#)

[MonoBehaviour.IsInvoking\(string\)](#)

[MonoBehaviour.StartCoroutine\(string\)](#)

[MonoBehaviour.StartCoroutine\(string, object\)](#)

[MonoBehaviour.StartCoroutine\(IEnumerator\)](#)

[MonoBehaviour.StartCoroutine\\_Auto\(IEnumerator\)](#)

[MonoBehaviour.StopCoroutine\(IEnumerator\)](#)

[MonoBehaviour.StopCoroutine\(Coroutine\)](#)

[MonoBehaviour.StopCoroutine\(string\)](#)

[MonoBehaviour.StopAllCoroutines\(\)](#)

[MonoBehaviour.print\(object\)](#)

[MonoBehaviour.useGUILayout](#)

[MonoBehaviour.runInEditMode](#)

[Behaviour.enabled](#)

[Behaviour.isActiveAndEnabled](#)

[Component.GetComponent\(Type\)](#)

[Component.GetComponent<T>\(\)](#)

[Component.TryGetComponent\(Type, out Component\)](#)

[Component.TryGetComponent<T>\(out T\)](#)

[Component.GetComponent\(string\)](#)

[Component.GetComponentInChildren\(Type, bool\)](#)

[Component.GetComponentInChildren\(Type\)](#)

[Component.GetComponentInChildren<T>\(bool\)](#)

[Component.GetComponentInChildren<T>\(\)](#)

[Component.GetComponentsInChildren\(Type, bool\)](#)

[Component.GetComponentsInChildren\(Type\)](#)

[Component.GetComponentsInChildren<T>\(bool\)](#)

[Component.GetComponentsInChildren<T>\(bool, List<T>\)](#)

[Component.GetComponentsInChildren<T>\(\)](#)

[Component.GetComponentsInChildren<T>\(List<T>\)](#)

[Component.GetComponentInParent\(Type\)](#)

[Component.GetComponentInParent<T>\(\)](#)

[Component.GetComponentsInParent\(Type, bool\)](#)

[Component.GetComponentsInParent\(Type\)](#)

[Component.GetComponentsInParent<T>\(bool\)](#)

[Component.GetComponentsInParent<T>\(bool, List<T>\)](#)

Component.GetComponentsInParent<T>()  
Component.GetComponents(Type)  
Component.GetComponents(Type, List<Component>)  
Component.GetComponents<T>(List<T>)  
Component.GetComponents<T>()  
Component.CompareTag(string)  
Component.SendMessageUpwards(string, object, SendMessageOptions)  
Component.SendMessageUpwards(string, object)  
Component.SendMessageUpwards(string)  
Component.SendMessageUpwards(string, SendMessageOptions)  
Component.SendMessage(string, object)  
Component.SendMessage(string)  
Component.SendMessage(string, object, SendMessageOptions)  
Component.SendMessage(string, SendMessageOptions)  
Component.BroadcastMessage(string, object, SendMessageOptions)  
Component.BroadcastMessage(string, object)  
Component.BroadcastMessage(string)  
Component.BroadcastMessage(string, SendMessageOptions)  
Component.transform  
Component.gameObject  
Component.tag  
Object.GetInstanceID()  
Object.GetHashCode()  
**Object.Equals(object)**  
Object.Instantiate(Object, Vector3, Quaternion)  
Object.Instantiate(Object, Vector3, Quaternion, Transform)  
Object.Instantiate(Object)  
Object.Instantiate(Object, Transform)  
**Object.Instantiate(Object, Transform, bool)**  
Object.Instantiate<T>(T)  
Object.Instantiate<T>(T, Vector3, Quaternion)  
Object.Instantiate<T>(T, Vector3, Quaternion, Transform)  
Object.Instantiate<T>(T, Transform)  
**Object.Instantiate<T>(T, Transform, bool)**  
Object.Destroy(Object, float)  
Object.Destroy(Object)  
Object.DestroyImmediate(Object, bool)  
Object.DestroyImmediate(Object)  
**Object.FindObjectsOfType(Type)**  
Object.DontDestroyOnLoad(Object)  
Object.DestroyObject(Object, float)  
Object.DestroyObject(Object)  
**Object.FindSceneObjectsOfType(Type)**  
**Object.FindObjectsOfTypeIncludingAssets(Type)**  
Object.FindObjectsOfType<T>()  
Object.FindObjectOfType<T>()  
**Object.FindObjectsOfTypeAll(Type)**  
**Object.FindObjectOfType(Type)**  
Object.ToString()  
Object.name  
Object.hideFlags

## Syntax

```
[AddComponentMenu("Tessera/Separation Constraint", 21)]
[RequireComponent(typeof(TesseraGenerator))]
public class SeparationConstraint : TesseraConstraint
```

## Fields

## minDistance

The count to be compared against.

## Declaration

```
public int minDistance
```

## Field Value

TYPE	DESCRIPTION
int	

## tiles

The set of tiles to count

## Declaration

```
public List<TesseraTileBase> tiles
```

## Field Value

TYPE	DESCRIPTION
List<TesseraTileBase>	

# Enum SquareFaceDir

Enum of the 4 sides of a square.

Namespace: [Tessera](#)

Assembly: Tessera.dll

Syntax

```
public enum SquareFaceDir
```

Fields

NAME	DESCRIPTION
Down	
Left	
Right	
Up	

Extension Methods

[SquareFaceDirExtensions.Forward\(SquareFaceDir\)](#)

[SquareFaceDirExtensions.GetSide\(SquareFaceDir\)](#)

[SquareFaceDirExtensions.Inverted\(SquareFaceDir\)](#)

# Class SquareFaceDirExtensions

Inheritance

[object](#)

SquareFaceDirExtensions

Namespace: [Tessera](#)

Assembly: Tessera.dll

Syntax

```
public static class SquareFaceDirExtensions
```

## Methods

[Forward\(SquareFaceDir\)](#)

Declaration

```
public static Vector3Int Forward(this SquareFaceDir faceDir)
```

Parameters

TYPE	NAME	DESCRIPTION
<a href="#">SquareFaceDir</a>	faceDir	

Returns

TYPE	DESCRIPTION
<a href="#">Vector3Int</a>	The normal vector for a given face.

[GetSide\(SquareFaceDir\)](#)

Declaration

```
public static int GetSide(this SquareFaceDir faceDir)
```

Parameters

TYPE	NAME	DESCRIPTION
<a href="#">SquareFaceDir</a>	faceDir	

Returns

TYPE	DESCRIPTION
<a href="#">int</a>	

[Inverted\(SquareFaceDir\)](#)

Declaration

```
public static SquareFaceDir Inverted(this SquareFaceDir faceDir)
```

Parameters

<b>TYPE</b>	<b>NAME</b>	<b>DESCRIPTION</b>
SquareFaceDir	faceDir	

Returns

<b>TYPE</b>	<b>DESCRIPTION</b>
SquareFaceDir	Returns the face dir with the opposite normal vector.

# Class SquareFaceExtensions

Inheritance

[object](#)

SquareFaceExtensions

Namespace: [Tessera](#)

Assembly: Tessera.dll

Syntax

```
public static class SquareFaceExtensions
```

## Methods

**Forward(SquareDir)**

Declaration

```
public static Vector3Int Forward(this SquareDir dir)
```

Parameters

TYPE	NAME	DESCRIPTION
SquareDir	dir	

Returns

TYPE	DESCRIPTION
Vector3Int	The normal vector for a given face.

**GetSide(SquareDir)**

Declaration

```
public static int GetSide(this SquareDir dir)
```

Parameters

TYPE	NAME	DESCRIPTION
SquareDir	dir	

Returns

TYPE	DESCRIPTION
int	

**Inverted(SquareDir)**

Declaration

```
public static SquareDir Inverted(this SquareDir dir)
```

Parameters

<b>TYPE</b>	<b>NAME</b>	<b>DESCRIPTION</b>
SquareDir	dir	

Returns

<b>TYPE</b>	<b>DESCRIPTION</b>
SquareDir	Returns the face dir with the opposite normal vector.

# Struct SquareRotation

Represents rotations / reflections of a square

Inherited Members

[ValueType.ToString\(\)](#)

Namespace: [Tessera](#)

Assembly: Tessera.dll

Syntax

```
public struct SquareRotation
```

## Properties

All

Declaration

```
public static SquareRotation[] All { get; }
```

## Property Value

TYPE	DESCRIPTION
<a href="#">SquareRotation[]</a>	

## Identity

Declaration

```
public static SquareRotation Identity { get; }
```

## Property Value

TYPE	DESCRIPTION
<a href="#">SquareRotation</a>	

## IsReflection

Declaration

```
public bool IsReflection { get; }
```

## Property Value

TYPE	DESCRIPTION
<a href="#">bool</a>	

## ReflectX

Declaration

```
public static SquareRotation ReflectX { get; }
```

## Property Value

TYPE	DESCRIPTION
SquareRotation	

## ReflectY

### Declaration

```
public static SquareRotation ReflectY { get; }
```

### Property Value

TYPE	DESCRIPTION
SquareRotation	

## RotateCCW

### Declaration

```
public static SquareRotation RotateCCW { get; }
```

### Property Value

TYPE	DESCRIPTION
SquareRotation	

## Rotation

### Declaration

```
public int Rotation { get; }
```

### Property Value

TYPE	DESCRIPTION
int	

## Methods

### Equals(object)

#### Declaration

```
public override bool Equals(object obj)
```

#### Parameters

TYPE	NAME	DESCRIPTION
object	obj	

#### Returns

TYPE	DESCRIPTION
bool	

## Overrides

[ValueType.Equals\(object\)](#)

[GetHashCode\(\)](#)

Declaration

```
public override int GetHashCode()
```

Returns

TYPE	DESCRIPTION
int	

Overrides

[ValueType.GetHashCode\(\)](#)

[Invert\(\)](#)

Declaration

```
public SquareRotation Invert()
```

Returns

TYPE	DESCRIPTION
SquareRotation	

[Rotate90\(int\)](#)

Declaration

```
public static SquareRotation Rotate90(int i)
```

Parameters

TYPE	NAME	DESCRIPTION
int	i	

Returns

TYPE	DESCRIPTION
SquareRotation	

Operators

[operator ==\(SquareRotation, SquareRotation\)](#)

Declaration

```
public static bool operator ==(SquareRotation a, SquareRotation b)
```

Parameters

TYPE	NAME	DESCRIPTION
SquareRotation	a	

TYPE	NAME	DESCRIPTION
SquareRotation	b	

Returns

TYPE	DESCRIPTION
bool	

### implicit operator SquareRotation(CellRotation)

Declaration

```
public static implicit operator SquareRotation(CellRotation r)
```

Parameters

TYPE	NAME	DESCRIPTION
CellRotation	r	

Returns

TYPE	DESCRIPTION
SquareRotation	

### implicit operator CellRotation(SquareRotation)

Declaration

```
public static implicit operator CellRotation(SquareRotation r)
```

Parameters

TYPE	NAME	DESCRIPTION
SquareRotation	r	

Returns

TYPE	DESCRIPTION
CellRotation	

### operator !=(SquareRotation, SquareRotation)

Declaration

```
public static bool operator !=(SquareRotation a, SquareRotation b)
```

Parameters

TYPE	NAME	DESCRIPTION
SquareRotation	a	
SquareRotation	b	

Returns

TYPE	DESCRIPTION
bool	

operator \*(SquareRotation, int)

Declaration

```
public static int operator *(SquareRotation a, int side)
```

Parameters

TYPE	NAME	DESCRIPTION
SquareRotation	a	
int	side	

Returns

TYPE	DESCRIPTION
int	

operator \*(SquareRotation, SquareFaceDir)

Declaration

```
public static SquareFaceDir operator *(SquareRotation rotation, SquareFaceDir faceDir)
```

Parameters

TYPE	NAME	DESCRIPTION
SquareRotation	rotation	
SquareFaceDir	faceDir	

Returns

TYPE	DESCRIPTION
SquareFaceDir	

operator \*(SquareRotation, SquareRotation)

Declaration

```
public static SquareRotation operator *(SquareRotation a, SquareRotation b)
```

Parameters

TYPE	NAME	DESCRIPTION
SquareRotation	a	
SquareRotation	b	

Returns

TYPE	DESCRIPTION
SquareRotation	

operator \*(SquareRotation, BoundsInt)

Declaration

```
public static BoundsInt operator *(SquareRotation r, BoundsInt bounds)
```

Parameters

TYPE	NAME	DESCRIPTION
SquareRotation	r	
BoundsInt	bounds	

Returns

TYPE	DESCRIPTION
BoundsInt	

operator \*(SquareRotation, Vector3Int)

Declaration

```
public static Vector3Int operator *(SquareRotation r, Vector3Int v)
```

Parameters

TYPE	NAME	DESCRIPTION
SquareRotation	r	
Vector3Int	v	

Returns

TYPE	DESCRIPTION
Vector3Int	

# Struct SubFaceTuple

Each face of a tessera tile is split up into several sub faces that are independently paintable. There's 9 for a square face - one center, 4 corners and 4 "dirs". The corner / dir values come from Sylves.SquareCellType, and similarly for faces with another number of sides. SubFaceTuple is used for address a specific subface.

It's an improvement on the earlier enum SubFace, which covers the same thing and has yet to be deprecated.

Namespace: [Tessera](#)

Assembly: Tessera.dll

Syntax

```
public struct SubFaceTuple
```

## Constructors

**SubFaceTuple(SubFaceType, int)**

Declaration

```
public SubFaceTuple(SubFaceType type, int value)
```

## Parameters

TYPE	NAME	DESCRIPTION
<a href="#">SubFaceType</a>	type	
<a href="#">int</a>	value	

## Fields

**type**

Declaration

```
public SubFaceType type
```

## Field Value

TYPE	DESCRIPTION
<a href="#">SubFaceType</a>	

**value**

Declaration

```
public int value
```

## Field Value

TYPE	DESCRIPTION
<a href="#">int</a>	

## Methods

**Equals(object)**

Declaration

```
public override bool Equals(object obj)
```

Parameters

TYPE	NAME	DESCRIPTION
object	obj	

Returns

TYPE	DESCRIPTION
bool	

Overrides

[ValueType.Equals\(object\)](#)

**Equals(SubFaceTuple)**

Declaration

```
public bool Equals(SubFaceTuple other)
```

Parameters

TYPE	NAME	DESCRIPTION
SubFaceTuple	other	

Returns

TYPE	DESCRIPTION
bool	

**GetHashCode()**

Declaration

```
public override int GetHashCode()
```

Returns

TYPE	DESCRIPTION
int	

Overrides

[ValueType.GetHashCode\(\)](#)

**Rotate(Connection)**

Declaration

```
public SubFaceTuple Rotate(Connection connection)
```

Parameters

TYPE	NAME	DESCRIPTION
Connection	connection	

Returns

TYPE	DESCRIPTION
SubFaceTuple	

## ToString()

Declaration

```
public override string ToString()
```

Returns

TYPE	DESCRIPTION
string	

Overrides

[ValueType.ToString\(\)](#)

## Operators

### operator ==(SubFaceTuple, SubFaceTuple)

Declaration

```
public static bool operator ==(SubFaceTuple a, SubFaceTuple b)
```

Parameters

TYPE	NAME	DESCRIPTION
SubFaceTuple	a	
SubFaceTuple	b	

Returns

TYPE	DESCRIPTION
bool	

### operator !=(SubFaceTuple, SubFaceTuple)

Declaration

```
public static bool operator !=(SubFaceTuple a, SubFaceTuple b)
```

Parameters

TYPE	NAME	DESCRIPTION
SubFaceTuple	a	

<b>TYPE</b>	<b>NAME</b>	<b>DESCRIPTION</b>
SubFaceTuple	b	

Returns

<b>TYPE</b>	<b>DESCRIPTION</b>
bool	

# Enum SubFaceType

Namespace: [Tessera](#)

Assembly: Tessera.dll

## Syntax

```
public enum SubFaceType
```

## Fields

NAME	DESCRIPTION
Center	
Corner	
Dir	

# Class SylvesConversions

Utilities for converting legacy Tessera enumerations to Sylves equivalents

Inheritance

[object](#)

SylvesConversions

Namespace: [Tessera](#)

Assembly: Tessera.dll

Syntax

```
public static class SylvesConversions
```

## Methods

[CubeOffset\(Vector3Int\)](#)

Declaration

```
public static Vector3Int CubeOffset(Vector3Int o)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	o	

Returns

TYPE	DESCRIPTION
Vector3Int	

[CubeOrientedFace\(OrientedFace\)](#)

Declaration

```
public static SylvesOrientedFace CubeOrientedFace(OrientedFace x)
```

Parameters

TYPE	NAME	DESCRIPTION
OrientedFace	x	

Returns

TYPE	DESCRIPTION
SylvesOrientedFace	

[HexOffset\(Vector3Int\)](#)

Declaration

```
public static Vector3Int HexOffset(Vector3Int o)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	o	

Returns

TYPE	DESCRIPTION
Vector3Int	

## HexOrientedFace(OrientedFace)

Declaration

```
public static SylvesOrientedFace HexOrientedFace(OrientedFace x)
```

Parameters

TYPE	NAME	DESCRIPTION
OrientedFace	x	

Returns

TYPE	DESCRIPTION
SylvesOrientedFace	

## SquareOffset(Vector3Int)

Declaration

```
public static Vector3Int SquareOffset(Vector3Int o)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	o	

Returns

TYPE	DESCRIPTION
Vector3Int	

## SquareOrientedFace(OrientedFace)

Declaration

```
public static SylvesOrientedFace SquareOrientedFace(OrientedFace x)
```

Parameters

TYPE	NAME	DESCRIPTION
OrientedFace	x	

Returns

<b>TYPE</b>	<b>DESCRIPTION</b>
SylvesOrientedFace	

### TriangleOffset(Vector3Int)

Declaration

```
public static Vector3Int TriangleOffset(Vector3Int o)
```

Parameters

<b>TYPE</b>	<b>NAME</b>	<b>DESCRIPTION</b>
Vector3Int	o	

Returns

<b>TYPE</b>	<b>DESCRIPTION</b>
Vector3Int	

### TrianglePrismOrientedFace(OrientedFace)

Declaration

```
public static SylvesOrientedFace TrianglePrismOrientedFace(OrientedFace x)
```

Parameters

<b>TYPE</b>	<b>NAME</b>	<b>DESCRIPTION</b>
OrientedFace	x	

Returns

<b>TYPE</b>	<b>DESCRIPTION</b>
SylvesOrientedFace	

### UndoCubeOffset(Vector3Int)

Declaration

```
public static Vector3Int UndoCubeOffset(Vector3Int o)
```

Parameters

<b>TYPE</b>	<b>NAME</b>	<b>DESCRIPTION</b>
Vector3Int	o	

Returns

<b>TYPE</b>	<b>DESCRIPTION</b>
Vector3Int	

### UndoCubeOrientedFace(SylvesOrientedFace)

Declaration

```
public static OrientedFace UndoCubeOrientedFace(SylvesOrientedFace x)
```

Parameters

TYPE	NAME	DESCRIPTION
SylvesOrientedFace	x	

Returns

TYPE	DESCRIPTION
OrientedFace	

### UndoHexOffset(Vector3Int)

Declaration

```
public static Vector3Int UndoHexOffset(Vector3Int o)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	o	

Returns

TYPE	DESCRIPTION
Vector3Int	

### UndoHexOrientedFace(SylvesOrientedFace)

Declaration

```
public static OrientedFace UndoHexOrientedFace(SylvesOrientedFace x)
```

Parameters

TYPE	NAME	DESCRIPTION
SylvesOrientedFace	x	

Returns

TYPE	DESCRIPTION
OrientedFace	

### UndoSquareOffset(Vector3Int)

Declaration

```
public static Vector3Int UndoSquareOffset(Vector3Int o)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	o	

Returns

TYPE	DESCRIPTION
Vector3Int	

## UndoSquareOrientedFace(SylvesOrientedFace)

Declaration

```
public static OrientedFace UndoSquareOrientedFace(SylvesOrientedFace x)
```

Parameters

TYPE	NAME	DESCRIPTION
SylvesOrientedFace	x	

Returns

TYPE	DESCRIPTION
OrientedFace	

## UndoTriangleOffset(Vector3Int)

Declaration

```
public static Vector3Int UndoTriangleOffset(Vector3Int o)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	o	

Returns

TYPE	DESCRIPTION
Vector3Int	

## UndoTrianglePrismOrientedFace(SylvesOrientedFace)

Declaration

```
public static OrientedFace UndoTrianglePrismOrientedFace(SylvesOrientedFace x)
```

Parameters

TYPE	NAME	DESCRIPTION
SylvesOrientedFace	x	

Returns

TYPE	DESCRIPTION
OrientedFace	

# Class SylvesExtensions

Inheritance

[object](#)

SylvesExtensions

Namespace: [Tessera](#)

Assembly: Tessera.dll

Syntax

```
public static class SylvesExtensions
```

Fields

[CubeCellType](#)

Declaration

```
public static readonly ICellType CubeCellType
```

Field Value

TYPE	DESCRIPTION
ICellType	

[CubeGridInstance](#)

Declaration

```
public static readonly IGrid CubeGridInstance
```

Field Value

TYPE	DESCRIPTION
IGrid	

[HexPrismCellType](#)

Declaration

```
public static readonly ICellType HexPrismCellType
```

Field Value

TYPE	DESCRIPTION
ICellType	

[HexPrismGridInstance](#)

Declaration

```
public static readonly IGrid HexPrismGridInstance
```

Field Value

TYPE	DESCRIPTION
IGrid	

## SquareCellType

Declaration

```
public static readonly ICellType SquareCellType
```

Field Value

TYPE	DESCRIPTION
ICellType	

## SquareGridInstance

Declaration

```
public static readonly IGrid SquareGridInstance
```

Field Value

TYPE	DESCRIPTION
IGrid	

## TrianglePrismCellType

Declaration

```
public static readonly ICellType TrianglePrismCellType
```

Field Value

TYPE	DESCRIPTION
ICellType	

## Properties

### TrianglePrismGridInstance

Declaration

```
public static IGrid TrianglePrismGridInstance { get; }
```

Property Value

TYPE	DESCRIPTION
IGrid	

## Methods

### FindCell(IGrid, Vector3, Matrix4x4, out Cell, out CellRotation)

Declaration

```
public static bool FindCell(this IGrid grid, Vector3 tileCenter, Matrix4x4 tileLocalToGridMatrix, out Cell cell, out CellRotation rotation)
```

Parameters

TYPE	NAME	DESCRIPTION
IGrid	grid	
Vector3	tileCenter	
Matrix4x4	tileLocalToGridMatrix	
Cell	cell	
CellRotation	rotation	

Returns

TYPE	DESCRIPTION
bool	

### GetCellCenter(IGrid, Vector3Int, Vector3, Vector3)

Declaration

```
public static Vector3 GetCellCenter(this IGrid sylvesCellGrid, Vector3Int offset, Vector3 center, Vector3 cellSize)
```

Parameters

TYPE	NAME	DESCRIPTION
IGrid	sylvesCellGrid	
Vector3Int	offset	
Vector3	center	
Vector3	cellSize	

Returns

TYPE	DESCRIPTION
Vector3	

### GetCellCorner(IGrid, Vector3Int, CellCorner, Vector3, Vector3)

Declaration

```
public static Vector3 GetCellCorner(this IGrid sylvesCellGrid, Vector3Int offset, CellCorner corner, Vector3 center, Vector3 cellSize)
```

Parameters

TYPE	NAME	DESCRIPTION
IGrid	sylvesCellGrid	
Vector3Int	offset	

TYPE	NAME	DESCRIPTION
CellCorner	corner	
Vector3	center	
Vector3	cellSize	

Returns

TYPE	DESCRIPTION
Vector3	

### GetCellGrid(ICellType)

Declaration

```
public static IGrid GetCellGrid(this ICellType cellType)
```

Parameters

TYPE	NAME	DESCRIPTION
ICellType	cellType	

Returns

TYPE	DESCRIPTION
IGrid	

### GetCellSizeTransform(IGrid, Vector3, Vector3)

Returns a transform from the canonical cell (of unit size) to a specific cell size and center.

Declaration

```
public static Matrix4x4 GetCellSizeTransform(this IGrid sylvesCellGrid, Vector3 center, Vector3 cellSize)
```

Parameters

TYPE	NAME	DESCRIPTION
IGrid	sylvesCellGrid	
Vector3	center	
Vector3	cellSize	

Returns

TYPE	DESCRIPTION
Matrix4x4	

### GetCellType(IGrid)

Declaration

```
public static ICellType GetCellType(this IGrid grid)
```

#### Parameters

TYPE	NAME	DESCRIPTION
IGrid	grid	

#### Returns

TYPE	DESCRIPTION
ICellType	

### GetDirPairs(ICellType)

#### Declaration

```
public static IEnumerable<(CellDir, CellDir> GetDirPairs(this ICellType cellType)
```

#### Parameters

TYPE	NAME	DESCRIPTION
ICellType	cellType	

#### Returns

TYPE	DESCRIPTION
IEnumerable<(CellDir, CellDir)>	

### GetDisplayName(ICellType, CellDir)

#### Declaration

```
public static string GetDisplayName(this ICellType cellType, CellDir cellDir)
```

#### Parameters

TYPE	NAME	DESCRIPTION
ICellType	cellType	
CellDir	cellDir	

#### Returns

TYPE	DESCRIPTION
string	

### GetReflectX(ICellType)

#### Declaration

```
public static CellRotation GetReflectX(this ICellType cellType)
```

#### Parameters

TYPE	NAME	DESCRIPTION
ICellType	cellType	

Returns

TYPE	DESCRIPTION
CellRotation	

### GetRotations(ICellType, bool, bool, RotationGroupType)

Declaration

```
public static IList<CellRotation> GetRotations(this ICellType cellType, bool rotatable = true, bool reflectable = true, RotationGroupType rotationGroupType = RotationGroupType.All)
```

Parameters

TYPE	NAME	DESCRIPTION
ICellType	cellType	
bool	rotatable	
bool	reflectable	
RotationGroupType	rotationGroupType	

Returns

TYPE	DESCRIPTION
IList<CellRotation>	

### RotateBy(ICellType, CellDir, FaceDetails, CellRotation)

Declaration

```
public static (CellDir, FaceDetails) RotateBy(this ICellType cellType, CellDir dir, FaceDetails faceDetails, CellRotation rot)
```

Parameters

TYPE	NAME	DESCRIPTION
ICellType	cellType	
CellDir	dir	
FaceDetails	faceDetails	
CellRotation	rot	

Returns

TYPE	DESCRIPTION
(CellDir, <a href="#">FaceDetails</a> )	

## ToTesseraMatrix(Connection)

### Declaration

```
public static Matrix4x4 ToTesseraMatrix(this Connection connection)
```

### Parameters

TYPE	NAME	DESCRIPTION
Connection	connection	

### Returns

TYPE	DESCRIPTION
Matrix4x4	

# Enum SylvesHexPrismCorner

Namespace: [Tessera](#)

Assembly: Tessera.dll

## Syntax

```
public enum SylvesHexPrismCorner
```

## Fields

NAME	DESCRIPTION
DownBack	
DownBackLeft	
DownBackRight	
DownForward	
DownForwardLeft	
DownForwardRight	
UpBack	
UpBackLeft	
UpBackRight	
UpForward	
UpForwardLeft	
UpForwardRight	

# Enum SylvesHexPrismDir

Namespace: [Tessera](#)

Assembly: Tessera.dll

## Syntax

```
public enum SylvesHexPrismDir
```

## Fields

NAME	DESCRIPTION
BackLeft	
BackRight	
Down	
ForwardLeft	
ForwardRight	
Left	
Right	
Up	

## Extension Methods

[HexPrismDirExtensions.Forward\(SylvesHexPrismDir\)](#)

[HexPrismDirExtensions.GetSide\(SylvesHexPrismDir\)](#)

[HexPrismDirExtensions.IsUpDown\(SylvesHexPrismDir\)](#)

[HexPrismDirExtensions.Up\(SylvesHexPrismDir\)](#)

# Struct SylvesOrientedFace

Records the painted colors and location of single face of one cube in a [TesseraTile](#)

Inherited Members

[ValueType.Equals\(object\)](#)

[ValueType.GetHashCode\(\)](#)

[ValueType.ToString\(\)](#)

Namespace: [Tessera](#)

Assembly: Tessera.dll

Syntax

```
[Serializable]
public struct SylvesOrientedFace
```

Constructors

[SylvesOrientedFace\(Vector3Int, CellDir, FaceDetails\)](#)

Declaration

```
public SylvesOrientedFace(Vector3Int offset, CellDir dir, FaceDetails faceDetails)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	offset	
CellDir	dir	
<a href="#">FaceDetails</a>	faceDetails	

Fields

**dir**

Declaration

```
public CellDir dir
```

Field Value

TYPE	DESCRIPTION
CellDir	

**faceDetails**

Declaration

```
public FaceDetails faceDetails
```

Field Value

TYPE	DESCRIPTION
<a href="#">FaceDetails</a>	

## offset

Declaration

```
public Vector3Int offset
```

Field Value

TYPE	DESCRIPTION
Vector3Int	

## Methods

Deconstruct(out Vector3Int, out CellDir, out FaceDetails)

Declaration

```
public void Deconstruct(out Vector3Int offset, out CellDir dir, out FaceDetails faceDetails)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	offset	
CellDir	dir	
FaceDetails	faceDetails	

# Enum SylvesTrianglePrismCorner

Namespace: [Tessera](#)

Assembly: Tessera.dll

## Syntax

```
public enum SylvesTrianglePrismCorner
```

## Fields

NAME	DESCRIPTION
DownBack	
DownBackLeft	
DownBackRight	
DownForward	
DownForwardLeft	
DownForwardRight	
UpBack	
UpBackLeft	
UpBackRight	
UpForward	
UpForwardLeft	
UpForwardRight	

# Enum SylvesTrianglePrismDir

Namespace: [Tessera](#)

Assembly: Tessera.dll

## Syntax

```
public enum SylvesTrianglePrismDir
```

## Fields

NAME	DESCRIPTION
Back	
BackLeft	
BackRight	
Down	
Forward	
ForwardLeft	
ForwardRight	
Up	

## Extension Methods

[TrianglePrismDirExtensions.Forward\(SylvesTrianglePrismDir\)](#)  
[TrianglePrismDirExtensions.GetSide\(SylvesTrianglePrismDir\)](#)  
[TrianglePrismDirExtensions.IsUpDown\(SylvesTrianglePrismDir\)](#)  
[TrianglePrismDirExtensions.IsValid\(SylvesTrianglePrismDir, bool\)](#)  
[TrianglePrismDirExtensions.IsValid\(SylvesTrianglePrismDir, Vector3Int\)](#)  
[TrianglePrismDirExtensions.SylvesOffsetDelta\(SylvesTrianglePrismDir\)](#)  
[TrianglePrismDirExtensions.Up\(SylvesTrianglePrismDir\)](#)

# Class TRS

Represents a position / rotation and scale. Much like a Transform, but without the association with a unity object.

Inheritance

[object](#)

[TRS](#)

Namespace: [Tessera](#)

Assembly: Tessera.dll

Syntax

```
public class TRS
```

Constructors

[TRS\(Matrix4x4\)](#)

Declaration

```
public TRS(Matrix4x4 m)
```

Parameters

TYPE	NAME	DESCRIPTION
Matrix4x4	m	

[TRS\(Vector3\)](#)

Declaration

```
public TRS(Vector3 position)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3	position	

[TRS\(Vector3, Quaternion, Vector3\)](#)

Declaration

```
public TRS(Vector3 position, Quaternion rotation, Vector3 scale)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3	position	
Quaternion	rotation	
Vector3	scale	

Properties

[Position](#)

Declaration

```
public Vector3 Position { get; }
```

#### Property Value

TYPE	DESCRIPTION
Vector3	

#### Rotation

##### Declaration

```
public Quaternion Rotation { get; }
```

#### Property Value

TYPE	DESCRIPTION
Quaternion	

#### Scale

##### Declaration

```
public Vector3 Scale { get; }
```

#### Property Value

TYPE	DESCRIPTION
Vector3	

#### Methods

##### Local(Transform)

##### Declaration

```
public static TRS Local(Transform t)
```

##### Parameters

TYPE	NAME	DESCRIPTION
Transform	t	

##### Returns

TYPE	DESCRIPTION
TRS	

##### ToMatrix()

##### Declaration

```
public Matrix4x4 ToMatrix()
```

##### Returns

<b>TYPE</b>	<b>DESCRIPTION</b>
Matrix4x4	

## World(Transform)

### Declaration

```
public static TRS World(Transform t)
```

### Parameters

<b>TYPE</b>	<b>NAME</b>	<b>DESCRIPTION</b>
Transform	t	

### Returns

<b>TYPE</b>	<b>DESCRIPTION</b>
TRS	

## Operators

### operator \*(TRS, TRS)

### Declaration

```
public static TRS operator *(TRS a, TRS b)
```

### Parameters

<b>TYPE</b>	<b>NAME</b>	<b>DESCRIPTION</b>
TRS	a	
TRS	b	

### Returns

<b>TYPE</b>	<b>DESCRIPTION</b>
TRS	

# Class TesseraBaseSceneConstraintComponent

Marks any component in the scene that can be interpreted by Tessera as a constraint. These eventually become ITesseraInitialConstraint objects

Inheritance

[object](#)

[Object](#)

[Component](#)

[Behaviour](#)

[MonoBehaviour](#)

[TesseraBaseSceneConstraintComponent](#)

[TesseraPinned](#)

[TesseraVolume](#)

Inherited Members

[MonoBehaviour.IsInvoking\(\)](#)

[MonoBehaviour.CancelInvoke\(\)](#)

[MonoBehaviour.Invoke\(string, float\)](#)

[MonoBehaviour.InvokeRepeating\(string, float, float\)](#)

[MonoBehaviour.CancelInvoke\(string\)](#)

[MonoBehaviour.IsInvoking\(string\)](#)

[MonoBehaviour.StartCoroutine\(string\)](#)

[MonoBehaviour.StartCoroutine\(string, object\)](#)

[MonoBehaviour.StartCoroutine\(IEnumerator\)](#)

[MonoBehaviour.StartCoroutine\\_Auto\(IEnumerator\)](#)

[MonoBehaviour.StopCoroutine\(IEnumerator\)](#)

[MonoBehaviour.StopCoroutine\(Coroutine\)](#)

[MonoBehaviour.StopCoroutine\(string\)](#)

[MonoBehaviour.StopAllCoroutines\(\)](#)

[MonoBehaviour.print\(object\)](#)

[MonoBehaviour.useGUILayout](#)

[MonoBehaviour.runInEditMode](#)

[Behaviour.enabled](#)

[Behaviour.isActiveAndEnabled](#)

[Component.GetComponent\(Type\)](#)

[Component.GetComponent<T>\(\)](#)

[Component.TryGetComponent\(Type, out Component\)](#)

[Component.TryGetComponent<T>\(out T\)](#)

[Component.GetComponent\(string\)](#)

[Component.GetComponentInChildren\(Type, bool\)](#)

[Component.GetComponentInChildren\(Type\)](#)

[Component.GetComponentInChildren<T>\(bool\)](#)

[Component.GetComponentInChildren<T>\(\)](#)

[Component.GetComponentsInChildren\(Type, bool\)](#)

[Component.GetComponentsInChildren\(Type\)](#)

[Component.GetComponentsInChildren<T>\(bool\)](#)

[Component.GetComponentsInChildren<T>\(bool, List<T>\)](#)

[Component.GetComponentsInChildren<T>\(\)](#)

[Component.GetComponentsInChildren<T>\(List<T>\)](#)

[Component.GetComponentInParent\(Type\)](#)

[Component.GetComponentInParent<T>\(\)](#)

[Component.GetComponentsInParent\(Type, bool\)](#)

Component.GetComponentsInParent(Type)  
Component.GetComponentsInParent<T>(bool)  
Component.GetComponentsInParent<T>(bool, List<T>)  
Component.GetComponentsInParent<T>()  
Component.GetComponents(Type)  
Component.GetComponents(Type, List<Component>)  
Component.GetComponents<T>(List<T>)  
Component.GetComponents<T>()  
Component.CompareTag(string)  
Component.SendMessageUpwards(string, object, SendMessageOptions)  
Component.SendMessageUpwards(string, object)  
Component.SendMessageUpwards(string)  
Component.SendMessageUpwards(string, SendMessageOptions)  
Component.SendMessage(string, object)  
Component.SendMessage(string)  
Component.SendMessage(string, object, SendMessageOptions)  
Component.SendMessage(string, SendMessageOptions)  
Component.BroadcastMessage(string, object, SendMessageOptions)  
Component.BroadcastMessage(string, object)  
Component.BroadcastMessage(string)  
Component.BroadcastMessage(string, SendMessageOptions)  
Component.transform  
Component.gameObject  
Component.tag  
Object.GetInstanceID()  
Object.GetHashCode()  
**Object.Equals(object)**  
Object.Instantiate(Object, Vector3, Quaternion)  
Object.Instantiate(Object, Vector3, Quaternion, Transform)  
Object.Instantiate(Object)  
Object.Instantiate(Object, Transform)  
**Object.Instantiate(Object, Transform, bool)**  
Object.Instantiate<T>(T)  
Object.Instantiate<T>(T, Vector3, Quaternion)  
Object.Instantiate<T>(T, Vector3, Quaternion, Transform)  
Object.Instantiate<T>(T, Transform)  
**Object.Instantiate<T>(T, Transform, bool)**  
Object.Destroy(Object, float)  
Object.Destroy(Object)  
Object.DestroyImmediate(Object, bool)  
Object.DestroyImmediate(Object)  
**Object.FindObjectsOfType(Type)**  
Object.DontDestroyOnLoad(Object)  
Object.DestroyObject(Object, float)  
Object.DestroyObject(Object)  
**Object.FindSceneObjectsOfType(Type)**  
**Object.FindObjectsOfTypeIncludingAssets(Type)**  
Object.FindObjectsOfType<T>()  
ObjectFindObjectOfType<T>()  
**Object.FindObjectsOfTypeAll(Type)**  
**Object.FindObjectOfType(Type)**  
Object.ToString()

Object.name

Object.hideFlags

Namespace: [Tessera](#)

Assembly: Tessera.dll

Syntax

```
public class TesseraBaseSceneConstraintComponent : MonoBehaviour
```

# Class TesseraCompletion

Returned by TesseraGenerator after generation finishes

Inheritance

[object](#)

TesseraCompletion

Namespace: [Tessera](#)

Assembly: Tessera.dll

Syntax

```
public class TesseraCompletion
```

Properties

**backtrackCount**

The number of times the generation process backtracked.

Declaration

```
public int backtrackCount { get; set; }
```

Property Value

TYPE	DESCRIPTION
int	

**contradictionLocation**

If success is false, indicates where the generation failed.

Declaration

```
public Vector3Int? contradictionLocation { get; set; }
```

Property Value

TYPE	DESCRIPTION
Vector3Int?	

**contradictionReason**

If success is false, indicates why the generation failed. This is not always helpful due to the nature of WFC. See the manual for more advice.

Declaration

```
public string contradictionReason { get; set; }
```

Property Value

TYPE	DESCRIPTION
string	

grid

Describes the geometry and layout of the cells. See separate Sylves documentations for more details.

<https://www.boristhebrave.com/docs/sylves/1>

Declaration

```
public IGrid grid { get; set; }
```

Property Value

TYPE	DESCRIPTION
IGrid	

### gridTransform

The position of the grid in world space. (Sylves grids always operate in local space).

Declaration

```
public TRS gridTransform { get; set; }
```

Property Value

TYPE	DESCRIPTION
TRS	

### isIncremental

Indicates these instances should be added to the previous set of instances.

Declaration

```
public bool isIncremental { get; set; }
```

Property Value

TYPE	DESCRIPTION
bool	

### retries

The number of times the generation process was restarted.

Declaration

```
public int retries { get; set; }
```

Property Value

TYPE	DESCRIPTION
int	

### success

True if all tiles were successfully found.

Declaration

```
public bool success { get; set; }
```

#### Property Value

TYPE	DESCRIPTION
bool	

#### tileData

The raw tile data, describing which tile is located in each cell.

#### Declaration

```
public IDictionary<Vector3Int, ModelTile> tileData { get; set; }
```

#### Property Value

TYPE	DESCRIPTION
IDictionary<Vector3Int, ModelTile>	

#### tileInstances

The list of tiles to create. Big tiles will be listed a single time, and each [TesseraTileInstance](#) has the world position set.

#### Declaration

```
public IList<TesseraTileInstance> tileInstances { get; }
```

#### Property Value

TYPE	DESCRIPTION
IList<TesseraTileInstance>	

#### Methods

##### LogError()

Writes error information to Unity's log.

#### Declaration

```
public void LogError()
```

# Class TesseraConstraint

Abstract class for all generator constraint components.

## NOTE

This class is available only in Tessera Pro

Inheritance

[object](#)

[Object](#)

[Component](#)

[Behaviour](#)

[MonoBehaviour](#)

[TesseraConstraint](#)

[BorderConstraint](#)

[CountConstraint](#)

[MirrorConstraint](#)

[PathConstraint](#)

[SeparationConstraint](#)

Inherited Members

[MonoBehaviour.IsInvoking\(\)](#)

[MonoBehaviour.CancelInvoke\(\)](#)

[MonoBehaviour.Invoke\(string, float\)](#)

[MonoBehaviour.InvokeRepeating\(string, float, float\)](#)

[MonoBehaviour.CancelInvoke\(string\)](#)

[MonoBehaviour.IsInvoking\(string\)](#)

[MonoBehaviour.StartCoroutine\(string\)](#)

[MonoBehaviour.StartCoroutine\(string, object\)](#)

[MonoBehaviour.StartCoroutine\(IEnumerator\)](#)

[MonoBehaviour.StartCoroutine\\_Auto\(IEnumerator\)](#)

[MonoBehaviour.StopCoroutine\(IEnumerator\)](#)

[MonoBehaviour.StopCoroutine\(Coroutine\)](#)

[MonoBehaviour.StopCoroutine\(string\)](#)

[MonoBehaviour.StopAllCoroutines\(\)](#)

[MonoBehaviour.print\(object\)](#)

[MonoBehaviour.useGUILayout](#)

[MonoBehaviour.runInEditMode](#)

[Behaviour.enabled](#)

[Behaviour.isActiveAndEnabled](#)

[Component.GetComponent\(Type\)](#)

[Component.GetComponent<T>\(\)](#)

[Component.TryGetComponent\(Type, out Component\)](#)

[Component.TryGetComponent<T>\(out T\)](#)

[Component.GetComponent\(string\)](#)

[Component.GetComponentInChildren\(Type, bool\)](#)

[Component.GetComponentInChildren\(Type\)](#)

[Component.GetComponentInChildren<T>\(bool\)](#)

[Component.GetComponentInChildren<T>\(\)](#)

[Component.GetComponentsInChildren\(Type, bool\)](#)

[Component.GetComponentsInChildren\(Type\)](#)

```
Component.GetComponentInChildren<T>(bool)
Component.GetComponentInChildren<T>(bool, List<T>)
Component.GetComponentInChildren<T>()
Component.GetComponentInChildren<T>(List<T>)
Component.GetComponentInParent(Type)
Component.GetComponentInParent<T>()
Component.GetComponentInParent(Type, bool)
Component.GetComponentInParent(Type)
Component.GetComponentInParent<T>(bool)
Component.GetComponentInParent<T>(bool, List<T>)
Component.GetComponentInParent<T>()
Component.GetComponents(Type)
Component.GetComponents(Type, List<Component>)
Component.GetComponents<T>(List<T>)
Component.GetComponents<T>()
Component.CompareTag(string)
Component.SendMessageUpwards(string, object, SendMessageOptions)
Component.SendMessageUpwards(string, object)
Component.SendMessageUpwards(string)
Component.SendMessageUpwards(string, SendMessageOptions)
Component.SendMessage(string, object)
Component.SendMessage(string)
Component.SendMessage(string, object, SendMessageOptions)
Component.SendMessage(string, SendMessageOptions)
Component.BroadcastMessage(string, object, SendMessageOptions)
Component.BroadcastMessage(string, object)
Component.BroadcastMessage(string)
Component.BroadcastMessage(string, SendMessageOptions)
Component.transform
Component.gameObject
Component.tag
Object.GetInstanceID()
Object.GetHashCode()
Object.Equals(object)
Object.Instantiate(Object, Vector3, Quaternion)
Object.Instantiate(Object, Vector3, Quaternion, Transform)
Object.Instantiate(Object)
Object.Instantiate(Object, Transform)
Object.Instantiate(Object, Transform, bool)
Object.Instantiate<T>(T)
Object.Instantiate<T>(T, Vector3, Quaternion)
Object.Instantiate<T>(T, Vector3, Quaternion, Transform)
Object.Instantiate<T>(T, Transform)
Object.Instantiate<T>(T, Transform, bool)
Object.Destroy(Object, float)
Object.Destroy(Object)
Object.DestroyImmediate(Object, bool)
Object.DestroyImmediate(Object)
Object.FindObjectsOfType(Type)
Object.DontDestroyOnLoad(Object)
Object.DestroyObject(Object, float)
Object.DestroyObject(Object)
```

[Object.FindSceneObjectsOfType\(Type\)](#)  
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#)  
Object.FindObjectsOfType<T>()  
Object.FindObjectOfType<T>()  
[Object.FindObjectsOfTypeAll\(Type\)](#)  
[Object.FindObjectOfType\(Type\)](#)  
Object.ToString()  
Object.name  
Object.hideFlags

Namespace: [Tessera](#)

Assembly: Tessera.dll

#### Syntax

```
public abstract class TesseraConstraint : MonoBehaviour
```

# Class TesseraGenerateOptions

Additional settings to customize the generation at runtime.

Inheritance

[object](#)

TesseraGenerateOptions

Namespace: [Tessera](#)

Assembly: Tessera.dll

Syntax

```
public class TesseraGenerateOptions
```

Fields

cancellationToken

Allows interuption of the calculations

Declaration

```
public CancellationToken cancellationToken
```

Field Value

TYPE	DESCRIPTION
<a href="#">CancellationToken</a>	

initialConstraints

If set, overrides TesseraGenerator.initialConstraints and TesseraGenerator.searchInitialConstraints.

Declaration

```
public List<ITesseraInitialConstraint> initialConstraints
```

Field Value

TYPE	DESCRIPTION
<a href="#">List&lt;ITesseraInitialConstraint&gt;</a>	

multithreaded

If set, then generation is offloaded to another thread stopping Unity from freezing. Requires you to use StartGenerate in a coroutine. Multithreaded is ignored in the WebGL player, as it doesn't support threads.

Declaration

```
public bool multithreaded
```

Field Value

TYPE	DESCRIPTION
<a href="#">bool</a>	

onComplete

Called when the generation is complete. By default, checks for success then invokes `onCreate` on each instance.

Declaration

```
public Action<TesseraCompletion> onComplete
```

Field Value

TYPE	DESCRIPTION
Action<TesseraCompletion>	

`onCreate`

Called for each newly generated tile. By default,  
`Tessera.TesseraGenerator.Instantiate(Tessera.TesseraTileInstance, Transform, IEngineInterface)` is used.

Declaration

```
public Action<TesseraTileInstance> onCreate
```

Field Value

TYPE	DESCRIPTION
Action<TesseraTileInstance>	

`progress`

Called with a string describing the current phase of the calculations, and the progress from 0 to 1. Progress can move backwards for retries or backtracing. Note progress can be called from threads other than the main thread.

Declaration

```
public Action<string, float> progress
```

Field Value

TYPE	DESCRIPTION
Action<string, float>	

`seed`

Fixes the seed for random number generator. By defult, random numbers from from `Unity.Random`.

Declaration

```
public int? seed
```

Field Value

TYPE	DESCRIPTION
int?	

# Class TesseraGenerator

GameObjects with this behaviour contain utilities to generate tile based levels using Wave Function Collapse (WFC). Call [Generate\(TesseraGenerateOptions\)](#) or [StartGenerate\(TesseraGenerateOptions\)](#) to run. The generation takes the following steps:

- Inspect the tiles in `tiles` and work out how they rotate and connect to each other.
- Setup any initial constraints that fix parts of the generation ([initialConstraints](#)).
- Fix the boundary of the generation if `skyBox` is set.
- Generate a set of tile instances that fits the above tiles and constraints.
- Optionally [retries](#) or [backtrack](#).
- Instantiates the tile instances.

Inheritance

`object`

`Object`

`Component`

`Behaviour`

`MonoBehaviour`

`TesseraGenerator`

Inherited Members

`MonoBehaviour.IsInvoking()`

`MonoBehaviour.CancelInvoke()`

`MonoBehaviour.Invoke(string, float)`

`MonoBehaviour.InvokeRepeating(string, float, float)`

`MonoBehaviour.CancelInvoke(string)`

`MonoBehaviour.IsInvoking(string)`

`MonoBehaviour.StartCoroutine(string)`

`MonoBehaviour.StartCoroutine(string, object)`

`MonoBehaviour.StartCoroutine(IEnumerator)`

`MonoBehaviour.StartCoroutine_Auto(IEnumerator)`

`MonoBehaviour.StopCoroutine(IEnumerator)`

`MonoBehaviour.StopCoroutine(Coroutine)`

`MonoBehaviour.StopCoroutine(string)`

`MonoBehaviour.StopAllCoroutines()`

`MonoBehaviour.print(object)`

`MonoBehaviour.useGUILayout`

`MonoBehaviour.runInEditMode`

`Behaviour.enabled`

`Behaviour.isActiveAndEnabled`

`Component.GetComponent(Type)`

`Component.GetComponent<T>()`

`Component.TryGetComponent(Type, out Component)`

`Component.TryGetComponent<T>(out T)`

`Component.GetComponent(string)`

`Component.GetComponentInChildren(Type, bool)`

`Component.GetComponentInChildren(Type)`

`Component.GetComponentInChildren<T>(bool)`

`Component.GetComponentInChildren<T>()`

`Component.GetComponentsInChildren(Type, bool)`

`Component.GetComponentsInChildren(Type)`

`Component.GetComponentsInChildren<T>(bool)`

`Component.GetComponentsInChildren<T>(bool, List<T>)`

```
Component.GetComponentsInChildren<T>()
Component.GetComponentsInChildren<T>(List<T>)
Component.GetComponentInParent(Type)
Component.GetComponentInParent<T>()
Component.GetComponentsInParent(Type, bool)
Component.GetComponentsInParent(Type)
Component.GetComponentsInParent<T>(bool)
Component.GetComponentsInParent<T>(bool, List<T>)
Component.GetComponentsInParent<T>()
Component.GetComponents(Type)
Component.GetComponents(Type, List<Component>)
Component.GetComponents<T>(List<T>)
Component.GetComponents<T>()
Component.CompareTag(string)
Component.SendMessageUpwards(string, object, SendMessageOptions)
Component.SendMessageUpwards(string, object)
Component.SendMessageUpwards(string)
Component.SendMessageUpwards(string, SendMessageOptions)
Component.SendMessage(string, object)
Component.SendMessage(string)
Component.SendMessage(string, object, SendMessageOptions)
Component.SendMessage(string, SendMessageOptions)
Component.BroadcastMessage(string, object, SendMessageOptions)
Component.BroadcastMessage(string, object)
Component.BroadcastMessage(string)
Component.BroadcastMessage(string, SendMessageOptions)
Component.transform
Component.gameObject
Component.tag
Object.GetInstanceID()
Object.GetHashCode()
Object.Equals(object)
Object.Instantiate(Object, Vector3, Quaternion)
Object.Instantiate(Object, Vector3, Quaternion, Transform)
Object.Instantiate(Object)
Object.Instantiate(Object, Transform)
Object.Instantiate(Object, Transform, bool)
Object.Instantiate<T>(T)
Object.Instantiate<T>(T, Vector3, Quaternion)
Object.Instantiate<T>(T, Vector3, Quaternion, Transform)
Object.Instantiate<T>(T, Transform)
Object.Instantiate<T>(T, Transform, bool)
Object.Destroy(Object, float)
Object.Destroy(Object)
Object.DestroyImmediate(Object, bool)
Object.DestroyImmediate(Object)
Object.FindObjectsOfType(Type)
Object.DontDestroyOnLoad(Object)
Object.DestroyObject(Object, float)
Object.DestroyObject(Object)
Object.FindSceneObjectsOfType(Type)
Object.FindObjectsOfTypeIncludingAssets(Type)
```

Object.FindObjectsOfType<T>()  
Object.FindObjectOfType<T>()  
**Object.FindObjectsOfTypeAll(Type)**  
**Object FindObjectOfType(Type)**  
Object.ToString()  
Object.name  
Object.hideFlags

Namespace: [Tessera](#)

Assembly: Tessera.dll

#### Syntax

```
[AddComponentMenu("Tessera/Tessera Generator")]
public class TesseraGenerator : MonoBehaviour
```

#### Fields

##### algorithm

Controls the algorithm used internally for Wave Function Collapse.

##### Declaration

```
[Tooltip("Controls the algorithm used internally for Wave Function Collapse.")]
public TesseraWfcAlgorithm algorithm
```

##### Field Value

TYPE	DESCRIPTION
<a href="#">TesseraWfcAlgorithm</a>	

##### backtrack

If set, backtracking will be used during generation. Backtracking can find solutions that would otherwise be failures, but can take a long time.

##### Declaration

```
[Tooltip("If set, backtracking will be used during generation.\nBacktracking can find solutions that would otherwise be failures, but can take a long time.")]
public bool backtrack
```

##### Field Value

TYPE	DESCRIPTION
<code>bool</code>	

##### contradictionTile

Game object to show in cells that cannot be solved.

##### Declaration

```
public GameObject contradictionTile
```

##### Field Value

<b>TYPE</b>	<b>DESCRIPTION</b>
GameObject	

## failureMode

Controls what is output when the generation fails.

### Declaration

```
public FailureMode failureMode
```

### Field Value

<b>TYPE</b>	<b>DESCRIPTION</b>
FailureMode	

## filterSurfaceSubmeshTiles

If true, and a [surfaceMesh](#) is set with multiple submeshes (materials), then use [surfaceSubmeshTiles](#).

### Declaration

```
[Tooltip("If true, filters which tiles appear on each material (submesh) of the surface mesh.")]
public bool filterSurfaceSubmeshTiles
```

### Field Value

<b>TYPE</b>	<b>DESCRIPTION</b>
bool	

## meshEdgesTolerance

Controls the tolerance (snap size) of comparing vertices for [MeshEdges](#)

### Declaration

```
public float meshEdgesTolerance
```

### Field Value

<b>TYPE</b>	<b>DESCRIPTION</b>
float	

## modelType

Sets which sort of model the generator uses. The model dictates how nearby tiles relate to each other.

### Declaration

```
public ModelType modelType
```

### Field Value

<b>TYPE</b>	<b>DESCRIPTION</b>
ModelType	

## overlapSize

The size of the overlap parameter for the overlapping model. [Overlapping](#)

Declaration

```
public Vector3Int overlapSize
```

Field Value

TYPE	DESCRIPTION
Vector3Int	

## recordUndo

Records undo/redo when run by pressing the Generate button in the Inspector.

Declaration

```
[Tooltip("Records undo/redo when run in the editor.")]  
public bool recordUndo
```

Field Value

TYPE	DESCRIPTION
bool	

## retries

If backtracking is off, how many times to retry generation if a solution cannot be found.

Declaration

```
[Tooltip("How many times to retry generation if a solution cannot be found.")]  
public int retries
```

Field Value

TYPE	DESCRIPTION
int	

## samples

For overlapping models, a list of objects to use as input samples. Each one will have its children inspected and read out.

[Overlapping](#)

Declaration

```
public List<GameObject> samples
```

Field Value

TYPE	DESCRIPTION
List<GameObject>	

## scaleUncertaintyTile

If true, the uncertainty tiles shrink as the solver gets more certain.

#### Declaration

```
public bool scaleUncertaintyTile
```

#### Field Value

TYPE	DESCRIPTION
bool	

### searchInitialConstraints

If true, then active tiles in the scene will be taken as initial constraints. If false, then no initial constraints are used. Using [initialConstraints](#) overrides either outcome.

#### Declaration

```
public bool searchInitialConstraints
```

#### Field Value

TYPE	DESCRIPTION
bool	

### seed

Fixes the seed for random number generator. If the value is zero, the seed is taken from Unity.Random

#### Declaration

```
[Tooltip("Fixes the seed for random number generator.\n If the value is zero, the seed is taken from\nUnity.Random.")]\npublic int seed
```

#### Field Value

TYPE	DESCRIPTION
int	

### skyBox

If set, this tile is used to define extra initial constraints for the boundary.

#### Declaration

```
[Tooltip("If set, this tile is used to define extra initial constraints for the boundary.")]\npublic TesseraTileBase skyBox
```

#### Field Value

TYPE	DESCRIPTION
TesseraTileBase	

### stepLimit

How many steps to take before retrying from the start.

## Declaration

```
[Tooltip("How many steps to take before retrying from the start.")]
public int stepLimit
```

## Field Value

TYPE	DESCRIPTION
int	

## surfaceMesh

If set, then tiles are generated on the surface of this mesh instead of a regular grid.

## Declaration

```
[Tooltip("If set, then tiles are generated on the surface of this mesh instead of a regular grid.")]
public Mesh surfaceMesh
```

## Field Value

TYPE	DESCRIPTION
Mesh	

## surfaceOffset

Height above the surface mesh that the bottom layer of tiles is generated at.

## Declaration

```
[Tooltip("Height above the surface mesh that the bottom layer of tiles is generated at.")]
public float surfaceOffset
```

## Field Value

TYPE	DESCRIPTION
float	

## surfaceSmoothNormals

Controls how normals are treated for meshes deformed to fit the surfaceMesh.

## Declaration

```
[Tooltip("Controls how normals are treated for meshes deformed to fit the surfaceMesh.")]
public bool surfaceSmoothNormals
```

## Field Value

TYPE	DESCRIPTION
bool	

## surfaceSubmeshTiles

A list of tiles to filter each submesh of [surfaceMesh](#) to. Ignored unless [filterSurfaceSubmeshTiles](#) is true.

## Declaration

```
public List<TileList> surfaceSubmeshTiles
```

#### Field Value

TYPE	DESCRIPTION
List<TileList>	

#### tileSelectionType

##### Declaration

```
public TileSelectionType tileSelectionType
```

#### Field Value

TYPE	DESCRIPTION
TileSelectionType	

#### tiles

The list of tiles eligible for generation.

##### Declaration

```
[Tooltip("The list of tiles eligible for generation.")]  
public List<TileEntry> tiles
```

#### Field Value

TYPE	DESCRIPTION
List<TileEntry>	

#### uncertaintyTile

Game object to show in cells that have yet to be fully solved.

##### Declaration

```
public GameObject uncertaintyTile
```

#### Field Value

TYPE	DESCRIPTION
GameObject	

#### Properties

##### CellGrid

Indicates the cell type of the tiles set up.

##### Declaration

```
public IGrid CellGrid { get; }
```

#### Property Value

<b>TYPE</b>	<b>DESCRIPTION</b>
IGrid	

## CellType

Indicates the cell type of the tiles set up.

Declaration

```
public ICellType CellType { get; }
```

Property Value

<b>TYPE</b>	<b>DESCRIPTION</b>
ICellType	

## bounds

The area of generation. Setting this will cause the size to be rounded to a multiple of [cellSize](#)

Declaration

```
public Bounds bounds { get; set; }
```

Property Value

<b>TYPE</b>	<b>DESCRIPTION</b>
Bounds	

## cellSize

The stride between each cell in the generation. "big" tiles may occupy a multiple of this cell size.

Declaration

```
public Vector3 cellSize { get; set; }
```

Property Value

<b>TYPE</b>	<b>DESCRIPTION</b>
Vector3	

## center

The local position of the center of the area to generate.

Declaration

```
public Vector3 center { get; set; }
```

Property Value

<b>TYPE</b>	<b>DESCRIPTION</b>
Vector3	

## origin

Declaration

```
public Vector3 origin { get; set; }
```

Property Value

TYPE	DESCRIPTION
Vector3	

## palette

Inherited from the first tile in [tiles](#).

Declaration

```
public TesseraPalette palette { get; }
```

Property Value

TYPE	DESCRIPTION
TesseraPalette	

## size

The size of the generator area, counting in cells each of size [cellSize](#).

Declaration

```
public Vector3Int size { get; set; }
```

Property Value

TYPE	DESCRIPTION
Vector3Int	

## Methods

### Clear()

Clears previously generated content.

Declaration

```
public void Clear()
```

### CloneSample(int, IEngineInterface)

Dumps out the internal view of the sample. Can be used for diagnosing issues with reading samples.

Declaration

```
public void CloneSample(int index = 0, IEngineInterface engineInterface = null)
```

Parameters

<b>TYPE</b>	<b>NAME</b>	<b>DESCRIPTION</b>
int	index	
IEngineInterface	engineInterface	

### Generate(TesseraGenerateOptions)

Synchronously runs the generation process described in the class docs.

Declaration

```
public TesseraCompletion Generate(TesseraGenerateOptions options = null)
```

Parameters

<b>TYPE</b>	<b>NAME</b>	<b>DESCRIPTION</b>
TesseraGenerateOptions	options	

Returns

<b>TYPE</b>	<b>DESCRIPTION</b>
TesseraCompletion	

### GetCellTypes()

For validation purposes

Declaration

```
public IList<ICellType> GetCellTypes()
```

Returns

<b>TYPE</b>	<b>DESCRIPTION</b>
IList<ICellType>	

### GetGrid()

Describes the geometry and layout of the cells. See separate Sylves documentations for more details.

<https://www.boristhebrave.com/docs/sylves/1> Note: If you are using a Mesh Surface based grid, then GetGrid() can be quite slow, and you are recommended to cache the result of it.

Declaration

```
public IGrid GetGrid()
```

Returns

<b>TYPE</b>	<b>DESCRIPTION</b>
IGrid	

### GetInitialConstraintBuilder()

Declaration

```
public TesseraInitialConstraintBuilder GetInitialConstraintBuilder()
```

Returns

TYPE	DESCRIPTION
TesseraInitialConstraintBuilder	

## GetTileOutput(bool)

Declaration

```
public ITesseraTileOutput GetTileOutput(bool forceIncremental = false)
```

Parameters

TYPE	NAME	DESCRIPTION
bool	forceIncremental	

Returns

TYPE	DESCRIPTION
ITesseraTileOutput	

## Instantiate(TesseraTileInstance, Transform, IEngineInterface)

Utility function that instantiates a tile instance in the scene. This is the default function used when you do not pass `onCreate` to the Generate method. It is essentially the same as Unity's normal Instantiate method with extra features:

- respects `InstantiateChildrenOnly`
- applies mesh transformations (Pro only)

Declaration

```
public static GameObject[] Instantiate(TesseraTileInstance instance, Transform parent, IEngineInterface engine = null)
```

Parameters

TYPE	NAME	DESCRIPTION
TesseraTileInstance	instance	The instance being created.
Transform	parent	The game object to parent the new game object to. This does not affect the world position of the instance
IEngineInterface	engine	

Returns

TYPE	DESCRIPTION
GameObject[]	The game objects created.

TYPE	DESCRIPTION

### Instantiate(TesseraTileInstance, Transform, GameObject, bool, IEngineInterface)

Utility function that instantiates a tile instance in the scene. This is the default function used when you do not pass `onCreate` to the Generate method. It is essentially the same as Unity's normal Instantiate method with extra features:

- respects `instantiateChildrenOnly`
- applies mesh transformations (Pro only)

Declaration

```
public static GameObject[] Instantiate(TesseraTileInstance instance, Transform parent, GameObject gameObject,
bool instantiateChildrenOnly, IEngineInterface engine = null)
```

Parameters

TYPE	NAME	DESCRIPTION
TesseraTileInstance	instance	The instance being created.
Transform	parent	The game object to parent the new game object to. This does not affect the world position of the instance
GameObject	gameObject	The game object to actually instantiate. Usually this is <code>instance.Tile.gameObject</code>
bool	instantiateChildrenOnly	Should <code>gameObject</code> be created, or just its children.
IEngineInterface	engine	

Returns

TYPE	DESCRIPTION
GameObject[]	The game objects created.

### Regenerate(TesseraGenerateOptions)

Runs Clear, then Generate

Declaration

```
public TesseraCompletion Regenerate(TesseraGenerateOptions options = null)
```

Parameters

<b>TYPE</b>	<b>NAME</b>	<b>DESCRIPTION</b>
TesseraGenerateOptions	options	

Returns

<b>TYPE</b>	<b>DESCRIPTION</b>
TesseraCompletion	

### StartGenerate(TesseraGenerateOptions)

Asynchronously runs the generation process described in the class docs, for use with StartCoroutine.

Declaration

```
public EnumeratorWithResult<TesseraCompletion> StartGenerate(TesseraGenerateOptions options = null)
```

Parameters

<b>TYPE</b>	<b>NAME</b>	<b>DESCRIPTION</b>
TesseraGenerateOptions	options	

Returns

<b>TYPE</b>	<b>DESCRIPTION</b>
EnumeratorWithResult<TesseraCompletion>	

Remarks

The default instantiation is still synchronous, so this can still cause frame glitches unless you override onCreate.

# Class TesseraHexTile

GameObjects with this behaviour record adjacency information for use with a [TesseraGenerator](#).

Inheritance

[object](#)

[Object](#)

[Component](#)

[Behaviour](#)

[MonoBehaviour](#)

[TesseraTileBase](#)

[TesseraHexTile](#)

Implements

[ISerializationCallbackReceiver](#)

Inherited Members

[TesseraTileBase.palette](#)

[TesseraTileBase.sylvesFaceDetails](#)

[TesseraTileBase.sylvesOffsets](#)

[TesseraTileBase.center](#)

[TesseraTileBase.cellSize](#)

[TesseraTileBase.rotatable](#)

[TesseraTileBase.reflectable](#)

[TesseraTileBase.rotationGroupType](#)

[TesseraTileBase.symmetric](#)

[TesseraTileBase.instantiateChildrenOnly](#)

[TesseraTileBase.Get\(Vector3Int, CellFaceDir\)](#)

[TesseraTileBase.Get\(Vector3Int, CellDir\)](#)

[TesseraTileBase.TryGet\(Vector3Int, CellFaceDir, out FaceDetails\)](#)

[TesseraTileBase.TryGet\(Vector3Int, CellDir, out FaceDetails\)](#)

[TesseraTileBase.GetSharedMesh\(\)](#)

[TesseraTileBase.AddOffset\(Vector3Int\)](#)

[TesseraTileBase.RemoveOffset\(Vector3Int\)](#)

[TesseraTileBase.OnBeforeSerialize\(\)](#)

[TesseraTileBase.OnAfterDeserialize\(\)](#)

[MonoBehaviour.IsInvoking\(\)](#)

[MonoBehaviour.CancelInvoke\(\)](#)

[MonoBehaviour.Invoke\(string, float\)](#)

[MonoBehaviour.InvokeRepeating\(string, float, float\)](#)

[MonoBehaviour.CancelInvoke\(string\)](#)

[MonoBehaviour.IsInvoking\(string\)](#)

[MonoBehaviour.StartCoroutine\(string\)](#)

[MonoBehaviour.StartCoroutine\(string, object\)](#)

[MonoBehaviour.StartCoroutine\(IEnumerator\)](#)

[MonoBehaviour.StartCoroutine\\_Auto\(IEnumerator\)](#)

[MonoBehaviour.StopCoroutine\(IEnumerator\)](#)

[MonoBehaviour.StopCoroutine\(Coroutine\)](#)

[MonoBehaviour.StopCoroutine\(string\)](#)

[MonoBehaviour.StopAllCoroutines\(\)](#)

[MonoBehaviour.print\(object\)](#)

[MonoBehaviour.useGUILayout](#)

[MonoBehaviour.RunWithEditMode](#)

Behaviour.enabled  
Behaviour.isActiveAndEnabled  
**Component.GetComponent(Type)**  
Component.GetComponent<T>()  
**Component.TryGetComponent(Type, out Component)**  
Component.TryGetComponent<T>(out T)  
Component.GetComponent(string)  
Component.GetComponentInChildren(Type, bool)  
Component.GetComponentInChildren(Type)  
**Component.GetComponentInChildren<T>(bool)**  
Component.GetComponentInChildren<T>()  
**Component.GetComponentsInChildren(Type, bool)**  
Component.GetComponentsInChildren(Type)  
Component.GetComponentsInChildren<T>(bool)  
Component.GetComponentsInChildren<T>(bool, List<T>)  
Component.GetComponentsInChildren<T>()  
**Component.GetComponentsInChildren<T>(List<T>)**  
**Component.GetComponentInParent(Type)**  
Component.GetComponentInParent<T>()  
**Component.GetComponentsInParent(Type, bool)**  
Component.GetComponentsInParent(Type)  
Component.GetComponentsInParent<T>(bool)  
Component.GetComponentsInParent<T>(bool, List<T>)  
Component.GetComponentsInParent<T>()  
**Component.GetComponents(Type)**  
**Component.GetComponents(Type, List<Component>)**  
**Component.GetComponents<T>(List<T>)**  
Component.GetComponents<T>()  
**Component.CompareTag(string)**  
Component.SendMessageUpwards(string, object, SendMessageOptions)  
Component.SendMessageUpwards(string, object)  
Component.SendMessageUpwards(string)  
Component.SendMessageUpwards(string, SendMessageOptions)  
Component.SendMessage(string, object)  
Component.SendMessage(string)  
Component.SendMessage(string, object, SendMessageOptions)  
Component.SendMessage(string, SendMessageOptions)  
Component.BroadcastMessage(string, object, SendMessageOptions)  
Component.BroadcastMessage(string, object)  
Component.BroadcastMessage(string)  
**Component.BroadcastMessage(string, SendMessageOptions)**  
Component.transform  
Component.gameObject  
Component.tag  
Object.GetInstanceID()  
Object.GetHashCode()  
**Object.Equals(object)**  
Object.Instantiate(Object, Vector3, Quaternion)  
Object.Instantiate(Object, Vector3, Quaternion, Transform)  
Object.Instantiate(Object)  
Object.Instantiate(Object, Transform)  
**Object.Instantiate(Object, Transform, bool)**

Object.Instantiate<T>(T)  
Object.Instantiate<T>(T, Vector3, Quaternion)  
Object.Instantiate<T>(T, Vector3, Quaternion, Transform)  
Object.Instantiate<T>(T, Transform)  
[Object.Instantiate<T>\(T, Transform, bool\)](#)  
[Object.Destroy\(Object, float\)](#)  
Object.Destroy(Object)  
[Object.DestroyImmediate\(Object, bool\)](#)  
Object.DestroyImmediate(Object)  
[Object.FindObjectsOfType\(Type\)](#)  
Object.DontDestroyOnLoad(Object)  
[Object.DestroyObject\(Object, float\)](#)  
Object.DestroyObject(Object)  
[Object.FindSceneObjectsOfType\(Type\)](#)  
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#)  
Object.FindObjectsOfType<T>()  
Object.FindObjectOfType<T>()  
[Object.FindObjectsOfTypeAll\(Type\)](#)  
[Object.FindObjectOfType\(Type\)](#)  
Object.ToString()  
Object.name  
Object.hideFlags

Namespace: [Tessera](#)

Assembly: Tessera.dll

## Syntax

```
[AddComponentMenu("Tessera/Tessera Hex Tile")]
public class TesseraHexTile : TesseraTileBase, ISerializationCallbackReceiver
```

## Constructors

[TesseraHexTile\(\)](#)

Declaration

```
public TesseraHexTile()
```

## Properties

[SubFaceInfo](#)

Declaration

```
public override ISubFaceInfo SubFaceInfo { get; }
```

Property Value

TYPE	DESCRIPTION
<a href="#">ISubFaceInfo</a>	

Overrides

[TesseraTileBase.SubFaceInfo](#)

[SylvesCellGrid](#)

Declaration

```
public override IGrid SylvesCellGrid { get; }
```

Property Value

TYPE	DESCRIPTION
IGrid	

Overrides

[TesseraTileBase.SylvesCellGrid](#)

## SylvesCellType

Declaration

```
public override ICellType SylvesCellType { get; }
```

Property Value

TYPE	DESCRIPTION
ICellType	

Overrides

[TesseraTileBase.SylvesCellType](#)

## Methods

### GetBounds()

Declaration

```
public BoundsInt GetBounds()
```

Returns

TYPE	DESCRIPTION
BoundsInt	

## Implements

UnityEngine.ISerializationCallbackReceiver

# Class TesseraInitialConstraint

Initial constraint objects fix parts of the generation process in places. Use the utility methods on [TesseraGenerator](#) to create these objects.

Inheritance

[object](#)

TesseraInitialConstraint

Implements

[ITesseraInitialConstraint](#)

Namespace: [Tessera](#)

Assembly: Tessera.dll

Syntax

```
[Serializable]
public class TesseraInitialConstraint : ITesseraInitialConstraint
```

## Properties

Name

Declaration

```
public string Name { get; }
```

Property Value

TYPE	DESCRIPTION
<a href="#">string</a>	

Implements

[ITesseraInitialConstraint](#)

# Class TesseraInitialConstraintBuilder

Utility for creating [ITesseraInitialConstraint](#) objects, which describe a variety of different constraints.

Inheritance

[object](#)

TesseraInitialConstraintBuilder

Namespace: [Tessera](#)

Assembly: Tessera.dll

Syntax

```
public class TesseraInitialConstraintBuilder
```

Constructors

[TesseraInitialConstraintBuilder\(Transform, IGrid\)](#)

Declaration

```
public TesseraInitialConstraintBuilder(Transform transform, IGrid grid)
```

Parameters

TYPE	NAME	DESCRIPTION
Transform	transform	
IGrid	grid	

Properties

Grid

Declaration

```
public IGrid Grid { get; }
```

Property Value

TYPE	DESCRIPTION
IGrid	

Methods

[CreateVolume\(string, List<TesseraTileBase>, List<Cell>\)](#)

Declaration

```
public TesseraVolumeFilter CreateVolume(string name, List<TesseraTileBase> tiles, List<Cell> cells)
```

Parameters

TYPE	NAME	DESCRIPTION
string	name	
List<TesseraTileBase>	tiles	

TYPE	NAME	DESCRIPTION
List<Cell>	cells	

Returns

TYPE	DESCRIPTION
TesseraVolumeFilter	

### GetInitialConstraint(TesseraPinned)

Gets the initial constraint from a given pin at a given position. It should be aligned with the grid defined by this generator.

Declaration

```
public ITesseraInitialConstraint GetInitialConstraint(TesseraPinned pin)
```

Parameters

TYPE	NAME	DESCRIPTION
TesseraPinned	pin	The pin to inspect

Returns

TYPE	DESCRIPTION
ITesseraInitialConstraint	

### GetInitialConstraint(TesseraPinned, Matrix4x4)

Gets the initial constraint from a given pin at a given position. It should be aligned with the grid defined by this generator.

Declaration

```
public ITesseraInitialConstraint GetInitialConstraint(TesseraPinned pin, Matrix4x4 localToWorldMatrix)
```

Parameters

TYPE	NAME	DESCRIPTION
TesseraPinned	pin	The pin to inspect
Matrix4x4	localToWorldMatrix	The matrix indicating the position and rotation of the tile

Returns

TYPE	DESCRIPTION
ITesseraInitialConstraint	

### GetInitialConstraint(TesseraTileBase)

Gets the initial constraint from a given tile. The tile should be aligned with the grid defined by this generator.

Declaration

```
public TesseraInitialConstraint GetInitialConstraint(TesseraTileBase tile)
```

Parameters

TYPE	NAME	DESCRIPTION
TesseraTileBase	tile	The tile to inspect

Returns

TYPE	DESCRIPTION
TesseralInitialConstraint	

**GetInitialConstraint(TesseraTileBase, Cell, CellRotation?)**

Gets the initial constraint from a given tile at a given cell.

Declaration

```
public TesseraInitialConstraint GetInitialConstraint(TesseraTileBase tile, Cell cell, CellRotation? cellRotation)
```

Parameters

TYPE	NAME	DESCRIPTION
TesseraTileBase	tile	The tile to inspect
Cell	cell	The location of the tile. For big tiles, this is the location of the (0, 0, 0) offset.
CellRotation?	cellRotation	The rotation of the tile. See Sylves docs for more info.

Returns

TYPE	DESCRIPTION
TesseralInitialConstraint	

**GetInitialConstraint(TesseraTileBase, Matrix4x4)**

Gets the initial constraint from a given tile at a given position. The tile should be aligned with the grid defined by this generator.

Declaration

```
public TesseraInitialConstraint GetInitialConstraint(TesseraTileBase tile, Matrix4x4 localToWorldMatrix)
```

Parameters

TYPE	NAME	DESCRIPTION

TYPE	NAME	DESCRIPTION
TesseraTileBase	tile	The tile to inspect
Matrix4x4	localToWorldMatrix	The matrix indicating the position and rotation of the tile

Returns

TYPE	DESCRIPTION
TesseraInitialConstraint	

### GetInitialConstraint(TesseraTileInstance, PinType)

Converts a TesseraTileInstance to a ITesseraInitialConstraint. This allows you to easily use the output of one generation for later generations

Declaration

```
public ITesseraInitialConstraint GetInitialConstraint(TesseraTileInstance tileInstance, PinType pinType = PinType.Pin)
```

Parameters

TYPE	NAME	DESCRIPTION
TesseraTileInstance	tileInstance	
PinType	pinType	

Returns

TYPE	DESCRIPTION
ITesseraInitialConstraint	

### GetInitialConstraint(TesseraVolume)

Gets the initial constraint from a given tile. The tile should be aligned with the grid defined by this generator.

Declaration

```
public TesseraVolumeFilter GetInitialConstraint(TesseraVolume volume)
```

Parameters

TYPE	NAME	DESCRIPTION
TesseraVolume	volume	

Returns

TYPE	DESCRIPTION
TesseraVolumeFilter	

## GetInitialConstraint(GameObject)

Gets the initial constraint for a given game object. It checks for a TesseraPinned, TesseraTile or TesseraVolume component.

Declaration

```
public ITesseraInitialConstraint GetInitialConstraint(GameObject gameObject)
```

Parameters

TYPE	NAME	DESCRIPTION
GameObject	gameObject	

Returns

TYPE	DESCRIPTION
ITesseralInitialConstraint	

## SearchInitialConstraints()

Searches the scene for all applicable game objects and converts them to ITesseralInitialConstraint

Declaration

```
public List<ITesseraInitialConstraint> SearchInitialConstraints()
```

Returns

TYPE	DESCRIPTION
List<ITesseralInitialConstraint>	

# Enum TesseraInstantiateBehaviour

Namespace: [Tessera](#)

Assembly: Tessera.dll

## Syntax

```
public enum TesseraInstantiateBehaviour
```

## Fields

NAME	DESCRIPTION
Constraint	
Default	
Function	
InstantiateAs	
Nothing	

# Class TesseraInstantiateEvent

Inheritance

[object](#)

[UnityEventBase](#)

[UnityEvent<TesseraCompletion, TesseraTileInstance, Action<GameObject>>](#)

[TesseraInstantiateEvent](#)

Implements

[ISerializationCallbackReceiver](#)

Inherited Members

[UnityEvent<TesseraCompletion, TesseraTileInstance, Action<GameObject>>.AddListener\(UnityAction<TesseraCompletion, TesseraTileInstance, Action<GameObject>>\)](#)

[UnityEvent<TesseraCompletion, TesseraTileInstance, Action<GameObject>>.RemoveListener\(UnityAction<TesseraCompletion, TesseraTileInstance, Action<GameObject>>\)](#)

[UnityEvent<TesseraCompletion, TesseraTileInstance, Action<GameObject>>.FindMethod\\_Impl\(string, object\)](#)

[UnityEvent<TesseraCompletion, TesseraTileInstance, Action<GameObject>>.Invoke\(TesseraCompletion, TesseraTileInstance, Action<GameObject>\)](#)

[UnityEventBase.FindMethod\\_Impl\(string, object\)](#)

[UnityEventBase.GetPersistentEventCount\(\)](#)

[UnityEventBase.GetPersistentTarget\(int\)](#)

[UnityEventBase.GetPersistentMethodName\(int\)](#)

[UnityEventBase.SetPersistentListenerState\(int, UnityEventCallState\)](#)

[UnityEventBase.AddListener\(object, MethodInfo\)](#)

[UnityEventBase.RemoveListener\(object, MethodInfo\)](#)

[UnityEventBase.RemoveAllListeners\(\)](#)

[UnityEventBase.Invoke\(object\[\]\)](#)

[UnityEventBase.ToString\(\)](#)

[UnityEventBase.GetValidMethodInfo\(object, string, Type\[\]\)](#)

[UnityEventBase.ValidateRegistration\(MethodInfo, object, PersistentListenerMode\)](#)

[UnityEventBase.ValidateRegistration\(MethodInfo, object, PersistentListenerMode, Type\)](#)

[UnityEventBase.RegisterPersistentListener\(int, object, MethodInfo\)](#)

Namespace: [Tessera](#)

Assembly: [Tessera.dll](#)

Syntax

```
[Serializable]
public class TesseraInstantiateEvent : UnityEvent<TesseraCompletion, TesseraTileInstance, Action<GameObject>>, ISerializationCallbackReceiver
```

Implements

[UnityEngine.ISerializationCallbackReceiver](#)

# Class TesseraInstantiateOutput

Attach this to a TesseraGenerator to control how tiles are instantiated.

## NOTE

This class is available only in Tessera Pro

Inheritance

[object](#)

[Object](#)

[Component](#)

[Behaviour](#)

[MonoBehaviour](#)

[TesseraInstantiateOutput](#)

Implements

[ITesseraTileOutput](#)

Inherited Members

[MonoBehaviour.IsInvoking\(\)](#)

[MonoBehaviour.CancelInvoke\(\)](#)

[MonoBehaviour.Invoke\(string, float\)](#)

[MonoBehaviour.InvokeRepeating\(string, float, float\)](#)

[MonoBehaviour.CancelInvoke\(string\)](#)

[MonoBehaviour.IsInvoking\(string\)](#)

[MonoBehaviour.StartCoroutine\(string\)](#)

[MonoBehaviour.StartCoroutine\(string, object\)](#)

[MonoBehaviour.StartCoroutine\(IEnumerator\)](#)

[MonoBehaviour.StartCoroutine\\_Auto\(IEnumerator\)](#)

[MonoBehaviour.StopCoroutine\(IEnumerator\)](#)

[MonoBehaviour.StopCoroutine\(Coroutine\)](#)

[MonoBehaviour.StopCoroutine\(string\)](#)

[MonoBehaviour.StopAllCoroutines\(\)](#)

[MonoBehaviour.print\(object\)](#)

[MonoBehaviour.useGUILayout](#)

[MonoBehaviour.runInEditMode](#)

[Behaviour.enabled](#)

[Behaviour.isActiveAndEnabled](#)

[Component.GetComponent\(Type\)](#)

[Component.GetComponent<T>\(\)](#)

[Component.TryGetComponent\(Type, out Component\)](#)

[Component.TryGetComponent<T>\(out T\)](#)

[Component.GetComponent\(string\)](#)

[Component.GetComponentInChildren\(Type, bool\)](#)

[Component.GetComponentInChildren\(Type\)](#)

[Component.GetComponentInChildren<T>\(bool\)](#)

[Component.GetComponentInChildren<T>\(\)](#)

[Component.GetComponentsInChildren\(Type, bool\)](#)

[Component.GetComponentsInChildren\(Type\)](#)

[Component.GetComponentsInChildren<T>\(bool\)](#)

[Component.GetComponentsInChildren<T>\(bool, List<T>\)](#)

[Component.GetComponentsInChildren<T>\(\)](#)

```
Component.GetComponentsInChildren<T>(List<T>)
Component.GetComponentInParent(Type)
Component.GetComponentInParent<T>()
Component.GetComponentsInParent(Type, bool)
Component.GetComponentsInParent(Type)
Component.GetComponentsInParent<T>(bool)
Component.GetComponentsInParent<T>(bool, List<T>)
Component.GetComponentsInParent<T>()
Component.GetComponents(Type)
Component.GetComponents(Type, List<Component>)
Component.GetComponents<T>(List<T>)
Component.GetComponents<T>()
Component.CompareTag(string)
Component.SendMessageUpwards(string, object, SendMessageOptions)
Component.SendMessageUpwards(string, object)
Component.SendMessageUpwards(string)
Component.SendMessageUpwards(string, SendMessageOptions)
Component.SendMessage(string, object)
Component.SendMessage(string)
Component.SendMessage(string, object, SendMessageOptions)
Component.SendMessage(string, SendMessageOptions)
Component.BroadcastMessage(string, object, SendMessageOptions)
Component.BroadcastMessage(string, object)
Component.BroadcastMessage(string)
Component.BroadcastMessage(string, SendMessageOptions)
Component.transform
Component.gameObject
Component.tag
Object.GetInstanceID()
Object.GetHashCode()
Object.Equals(object)
Object.Instantiate(Object, Vector3, Quaternion)
Object.Instantiate(Object, Vector3, Quaternion, Transform)
Object.Instantiate(Object)
Object.Instantiate(Object, Transform)
Object.Instantiate(Object, Transform, bool)
Object.Instantiate<T>(T)
Object.Instantiate<T>(T, Vector3, Quaternion)
Object.Instantiate<T>(T, Vector3, Quaternion, Transform)
Object.Instantiate<T>(T, Transform)
Object.Instantiate<T>(T, Transform, bool)
Object.Destroy(Object, float)
Object.Destroy(Object)
Object.DestroyImmediate(Object)
Object.DestroyImmediate(Object)
Object.FindObjectsOfType(Type)
Object.DontDestroyOnLoad(Object)
Object.DestroyObject(Object, float)
Object.DestroyObject(Object)
Object.FindSceneObjectsOfType(Type)
Object.FindObjectsOfTypeIncludingAssets(Type)
Object.FindObjectsOfType<T>()
```

`Object.FindObjectOfType<T>()`  
`Object.FindObjectsOfTypeAll(Type)`  
`Object.FindObjectOfType(Type)`  
`Object.ToString()`  
`Object.name`  
`Object.hideFlags`

Namespace: [Tessera](#)  
Assembly: Tessera.dll

#### Syntax

```
[RequireComponent(typeof(TesseraGenerator))]  
[AddComponentMenu("Tessera/Tessera Instantiate Output", 40)]  
public class TesseraInstantiateOutput : MonoBehaviour, ITesseraTileOutput
```

#### Constructors

##### `TesseraInstantiateOutput()`

Declaration

```
public TesseraInstantiateOutput()
```

#### Fields

##### `parent`

Declaration

```
public Transform parent
```

#### Field Value

TYPE	DESCRIPTION
Transform	

##### `tileMappings`

Declaration

```
public TesseraInstantiateOutputTileMapping[] tileMappings
```

#### Field Value

TYPE	DESCRIPTION
<a href="#">TesseraInstantiateOutputTileMapping[]</a>	

#### Properties

##### `IsEmpty`

Is the output currently empty.

Declaration

```
public bool IsEmpty { get; }
```

#### Property Value

TYPE	DESCRIPTION
bool	

## SupportsIncremental

Is this output safe to use with `AnimatedGenerator`

Declaration

```
public bool SupportsIncremental { get; }
```

## Property Value

TYPE	DESCRIPTION
bool	

## Methods

### `ClearTiles(IEngineInterface)`

Clear the output

Declaration

```
public void ClearTiles(IEngineInterface engine)
```

## Parameters

TYPE	NAME	DESCRIPTION
IEngineInterface	engine	

### `UpdateTiles(TesseraCompletion, IEngineInterface)`

Update a chunk of tiles. If incremental updates are supported, then:

- Tiles can replace other tiles, as indicated by the `Cells` field.
- A tile of null indicates that the tile should be erased

Declaration

```
public void UpdateTiles(TesseraCompletion completion, IEngineInterface engine)
```

## Parameters

TYPE	NAME	DESCRIPTION
TesseraCompletion	completion	
IEngineInterface	engine	

## Implements

`ITesseraTileOutput`

# Class TesseraInstantiateOutputTileMapping

Inheritance

[object](#)

TesseraInstantiateOutputTileMapping

Namespace: [Tessera](#)

Assembly: Tessera.dll

Syntax

```
[Serializable]
public class TesseraInstantiateOutputTileMapping
```

Fields

**behaviour**

Declaration

```
public TesseraInstantiateBehaviour behaviour
```

Field Value

TYPE	DESCRIPTION
<a href="#">TesseraInstantiateBehaviour</a>	

**constraint**

Declaration

```
public TesseraBaseSceneConstraintComponent constraint
```

Field Value

TYPE	DESCRIPTION
<a href="#">TesseraBaseSceneConstraintComponent</a>	

**from**

Declaration

```
public TesseraTileBase from
```

Field Value

TYPE	DESCRIPTION
<a href="#">TesseraTileBase</a>	

**instantiateChildrenOnly**

Declaration

```
public bool instantiateChildrenOnly
```

Field Value

TYPE	DESCRIPTION
bool	

## instantiateEvent

Declaration

```
public TesseraInstantiateEvent instantiateEvent
```

Field Value

TYPE	DESCRIPTION
TesseraInstantiateEvent	

## to

Declaration

```
public GameObject to
```

Field Value

TYPE	DESCRIPTION
GameObject	

# Class TesseraMeshOutput

Attach this to a TesseraGenerator to output the tiles to a single mesh instead of instantiating them.

## NOTE

This class is available only in Tessera Pro

Inheritance

[object](#)

[Object](#)

[Component](#)

[Behaviour](#)

[MonoBehaviour](#)

[TesseraMeshOutput](#)

Implements

[ITesseraTileOutput](#)

Inherited Members

[MonoBehaviour.IsInvoking\(\)](#)

[MonoBehaviour.CancelInvoke\(\)](#)

[MonoBehaviour.Invoke\(string, float\)](#)

[MonoBehaviour.InvokeRepeating\(string, float, float\)](#)

[MonoBehaviour.CancelInvoke\(string\)](#)

[MonoBehaviour.IsInvoking\(string\)](#)

[MonoBehaviour.StartCoroutine\(string\)](#)

[MonoBehaviour.StartCoroutine\(string, object\)](#)

[MonoBehaviour.StartCoroutine\(IEnumerator\)](#)

[MonoBehaviour.StartCoroutine\\_Auto\(IEnumerator\)](#)

[MonoBehaviour.StopCoroutine\(IEnumerator\)](#)

[MonoBehaviour.StopCoroutine\(Coroutine\)](#)

[MonoBehaviour.StopCoroutine\(string\)](#)

[MonoBehaviour.StopAllCoroutines\(\)](#)

[MonoBehaviour.print\(object\)](#)

[MonoBehaviour.useGUILayout](#)

[MonoBehaviour.runInEditMode](#)

[Behaviour.enabled](#)

[Behaviour.isActiveAndEnabled](#)

[Component.GetComponent\(Type\)](#)

[Component.GetComponent<T>\(\)](#)

[Component.TryGetComponent\(Type, out Component\)](#)

[Component.TryGetComponent<T>\(out T\)](#)

[Component.GetComponent\(string\)](#)

[Component.GetComponentInChildren\(Type, bool\)](#)

[Component.GetComponentInChildren\(Type\)](#)

[Component.GetComponentInChildren<T>\(bool\)](#)

[Component.GetComponentInChildren<T>\(\)](#)

[Component.GetComponentsInChildren\(Type, bool\)](#)

[Component.GetComponentsInChildren\(Type\)](#)

[Component.GetComponentsInChildren<T>\(bool\)](#)

[Component.GetComponentsInChildren<T>\(bool, List<T>\)](#)

[Component.GetComponentsInChildren<T>\(\)](#)

```
Component.GetComponentsInChildren<T>(List<T>)
Component.GetComponentInParent(Type)
Component.GetComponentInParent<T>()
Component.GetComponentsInParent(Type, bool)
Component.GetComponentsInParent(Type)
Component.GetComponentsInParent<T>(bool)
Component.GetComponentsInParent<T>(bool, List<T>)
Component.GetComponentsInParent<T>()
Component.GetComponents(Type)
Component.GetComponents(Type, List<Component>)
Component.GetComponents<T>(List<T>)
Component.GetComponents<T>()
Component.CompareTag(string)
Component.SendMessageUpwards(string, object, SendMessageOptions)
Component.SendMessageUpwards(string, object)
Component.SendMessageUpwards(string)
Component.SendMessageUpwards(string, SendMessageOptions)
Component.SendMessage(string, object)
Component.SendMessage(string)
Component.SendMessage(string, object, SendMessageOptions)
Component.SendMessage(string, SendMessageOptions)
Component.BroadcastMessage(string, object, SendMessageOptions)
Component.BroadcastMessage(string, object)
Component.BroadcastMessage(string)
Component.BroadcastMessage(string, SendMessageOptions)
Component.transform
Component.gameObject
Component.tag
Object.GetInstanceID()
Object.GetHashCode()
Object.Equals(object)
Object.Instantiate(Object, Vector3, Quaternion)
Object.Instantiate(Object, Vector3, Quaternion, Transform)
Object.Instantiate(Object)
Object.Instantiate(Object, Transform)
Object.Instantiate(Object, Transform, bool)
Object.Instantiate<T>(T)
Object.Instantiate<T>(T, Vector3, Quaternion)
Object.Instantiate<T>(T, Vector3, Quaternion, Transform)
Object.Instantiate<T>(T, Transform)
Object.Instantiate<T>(T, Transform, bool)
Object.Destroy(Object, float)
Object.Destroy(Object)
Object.DestroyImmediate(Object, bool)
Object.DestroyImmediate(Object)
Object.FindObjectsOfType(Type)
Object.DontDestroyOnLoad(Object)
Object.DestroyObject(Object, float)
Object.DestroyObject(Object)
Object.FindSceneObjectsOfType(Type)
Object.FindObjectsOfTypeIncludingAssets(Type)
Object.FindObjectsOfType<T>()
```

`Object.FindObjectOfType<T>()`  
`Object.FindObjectsOfTypeAll(Type)`  
`Object.FindObjectOfType(Type)`  
`Object.ToString()`  
`Object.name`  
`Object.hideFlags`

Namespace: [Tessera](#)

Assembly: Tessera.dll

#### Syntax

```
[RequireComponent(typeof(TesseraGenerator))]  
[AddComponentMenu("Tessera/Tessera Mesh Output", 40)]  
public class TesseraMeshOutput : MonoBehaviour, ITesseraTileOutput
```

#### Fields

##### chunkSize

Declaration

```
public Vector3 chunkSize
```

##### Field Value

TYPE	DESCRIPTION
Vector3	

##### colliders

Declaration

```
public TesseraMeshOutputCollider colliders
```

##### Field Value

TYPE	DESCRIPTION
TesseraMeshOutputCollider	

##### materialGrouping

Declaration

```
public TesseraMeshOutputMaterialGrouping materialGrouping
```

##### Field Value

TYPE	DESCRIPTION
TesseraMeshOutputMaterialGrouping	

##### singleMaterial

Declaration

```
public Material singleMaterial
```

##### Field Value

TYPE	DESCRIPTION
Material	

## useChunks

Declaration

```
public bool useChunks
```

Field Value

TYPE	DESCRIPTION
bool	

## Properties

### ChunkGrid

Declaration

```
public CubeGrid ChunkGrid { get; }
```

Property Value

TYPE	DESCRIPTION
CubeGrid	

### IsEmpty

Is the output currently empty.

Declaration

```
public bool IsEmpty { get; }
```

Property Value

TYPE	DESCRIPTION
bool	

### SupportsIncremental

Is this output safe to use with AnimatedGenerator

Declaration

```
public bool SupportsIncremental { get; }
```

Property Value

TYPE	DESCRIPTION
bool	

## Methods

### ClearTiles(IEngineInterface)

Clear the output

Declaration

```
public void ClearTiles(IEngineInterface engine)
```

Parameters

TYPE	NAME	DESCRIPTION
IEngineInterface	engine	

GetCellsInChunk(IGrid, Cell)

Declaration

```
public IEnumerable<Cell> GetCellsInChunk(IGrid grid, Cell chunk)
```

Parameters

TYPE	NAME	DESCRIPTION
IGrid	grid	
Cell	chunk	

Returns

TYPE	DESCRIPTION
IEnumerable<Cell>	

GetChunk(IGrid, Cell)

Declaration

```
public Cell GetChunk(IGrid grid, Cell cell)
```

Parameters

TYPE	NAME	DESCRIPTION
IGrid	grid	
Cell	cell	

Returns

TYPE	DESCRIPTION
Cell	

UpdateTiles(TesseraCompletion, IEngineInterface)

Update a chunk of tiles. If incremental updates are supported, then:

- Tiles can replace other tiles, as indicated by the [Cells](#) field.
- A tile of null indicates that the tile should be erased

Declaration

```
public void UpdateTiles(TesseraCompletion completion, IEngineInterface engine)
```

Parameters

TYPE	NAME	DESCRIPTION
TesseraCompletion	completion	
IEngineInterface	engine	

Implements

[ITesseraTileOutput](#)

# Enum TesseraMeshOutputCollider

Namespace: [Tessera](#)

Assembly: Tessera.dll

## Syntax

```
public enum TesseraMeshOutputCollider
```

## Fields

NAME	DESCRIPTION
Merge	Merge all MeshColliders, other colliders ignored.
None	Don't do anything.
ReuseRenderMesh	Use the same mesh as MeshFilter, in a MeshCollider

# Enum TesseraMeshOutputMaterialGrouping

Namespace: [Tessera](#)

Assembly: Tessera.dll

Syntax

```
public enum TesseraMeshOutputMaterialGrouping
```

Fields

NAME	DESCRIPTION
Single	Merge everything to a single material
Unique	Every material in the input becomes a material in the output
UniqueByName	Every name of material in the input becomes a material in the output. (useful when you have duplicate materials of the same name)

# Class TesseraMultipassGenerator

Inheritance

[object](#)  
[Object](#)  
[Component](#)  
[Behaviour](#)  
[MonoBehaviour](#)  
[TesseraMultipassGenerator](#)

Inherited Members

[MonoBehaviour.IsInvoking\(\)](#)  
[MonoBehaviour.CancelInvoke\(\)](#)  
[MonoBehaviour.Invoke\(string, float\)](#)  
[MonoBehaviour.InvokeRepeating\(string, float, float\)](#)  
[MonoBehaviour.CancelInvoke\(string\)](#)  
[MonoBehaviour.IsInvoking\(string\)](#)  
[MonoBehaviour.StartCoroutine\(string\)](#)  
[MonoBehaviour.StartCoroutine\(string, object\)](#)  
[MonoBehaviour.StartCoroutine\(IEnumerator\)](#)  
[MonoBehaviour.StartCoroutine\\_Auto\(IEnumerator\)](#)  
[MonoBehaviour.StopCoroutine\(IEnumerator\)](#)  
[MonoBehaviour.StopCoroutine\(Coroutine\)](#)  
[MonoBehaviour.StopCoroutine\(string\)](#)  
[MonoBehaviour.StopAllCoroutines\(\)](#)  
[MonoBehaviour.print\(object\)](#)  
[MonoBehaviour.useGUILayout](#)  
[MonoBehaviour.runInEditMode](#)  
[Behaviour.enabled](#)  
[Behaviour.isActiveAndEnabled](#)  
[Component.GetComponent\(Type\)](#)  
[Component.GetComponent<T>\(\)](#)  
[Component.TryGetComponent\(Type, out Component\)](#)  
[Component.TryGetComponent<T>\(out T\)](#)  
[Component.GetComponent\(string\)](#)  
[Component.GetComponentInChildren\(Type, bool\)](#)  
[Component.GetComponentInChildren\(Type\)](#)  
[Component.GetComponentInChildren<T>\(bool\)](#)  
[Component.GetComponentInChildren<T>\(\)](#)  
[Component.GetComponentsInChildren\(Type, bool\)](#)  
[Component.GetComponentsInChildren\(Type\)](#)  
[Component.GetComponentsInChildren<T>\(bool\)](#)  
[Component.GetComponentsInChildren<T>\(bool, List<T>\)](#)  
[Component.GetComponentsInChildren<T>\(\)](#)  
[Component.GetComponentsInChildren<T>\(List<T>\)](#)  
[Component.GetComponentInParent\(Type\)](#)  
[Component.GetComponentInParent<T>\(\)](#)  
[Component.GetComponentsInParent\(Type, bool\)](#)  
[Component.GetComponentsInParent\(Type\)](#)  
[Component.GetComponentsInParent<T>\(bool\)](#)  
[Component.GetComponentsInParent<T>\(bool, List<T>\)](#)  
[Component.GetComponentsInParent<T>\(\)](#)

Component.GetComponents(Type)  
Component.GetComponents(Type, List<Component>)  
Component.GetComponents<T>(List<T>)  
Component.GetComponents<T>()  
Component.CompareTag(string)  
Component.SendMessageUpwards(string, object, SendMessageOptions)  
Component.SendMessageUpwards(string, object)  
Component.SendMessageUpwards(string)  
Component.SendMessageUpwards(string, SendMessageOptions)  
Component.SendMessage(string, object)  
Component.SendMessage(string)  
Component.SendMessage(string, object, SendMessageOptions)  
Component.SendMessage(string, SendMessageOptions)  
Component.BroadcastMessage(string, object, SendMessageOptions)  
Component.BroadcastMessage(string, object)  
Component.BroadcastMessage(string)  
Component.BroadcastMessage(string, SendMessageOptions)  
Component.transform  
Component.gameObject  
Component.tag  
Object.GetInstanceID()  
Object.GetHashCode()  
**Object.Equals(object)**  
Object.Instantiate(Object, Vector3, Quaternion)  
Object.Instantiate(Object, Vector3, Quaternion, Transform)  
Object.Instantiate(Object)  
Object.Instantiate(Object, Transform)  
**Object.Instantiate(Object, Transform, bool)**  
Object.Instantiate<T>(T)  
Object.Instantiate<T>(T, Vector3, Quaternion)  
Object.Instantiate<T>(T, Vector3, Quaternion, Transform)  
Object.Instantiate<T>(T, Transform)  
**Object.Instantiate<T>(T, Transform, bool)**  
Object.Destroy(Object, float)  
Object.Destroy(Object)  
**Object.DestroyImmediate(Object, bool)**  
Object.DestroyImmediate(Object)  
**Object.FindObjectsOfType(Type)**  
Object.DontDestroyOnLoad(Object)  
**Object.DestroyObject(Object, float)**  
Object.DestroyObject(Object)  
**Object.FindSceneObjectsOfType(Type)**  
Object.FindObjectsOfTypeIncludingAssets(Type)  
Object.FindObjectsOfType<T>()  
Object.FindObjectOfType<T>()  
**Object.FindObjectsOfTypeAll(Type)**  
**Object.FindObjectOfType(Type)**  
Object.ToString()  
Object.name  
Object.hideFlags

Namespace: **Tessera**

Assembly: Tessera.dll

## Syntax

```
public class TesseraMultipassGenerator : MonoBehaviour
```

## Fields

### passes

Declaration

```
public TesseraMultipassPass[] passes
```

#### Field Value

TYPE	DESCRIPTION
TesseraMultipassPass[]	

## Methods

### Clear()

Declaration

```
public void Clear()
```

### Generate()

Declaration

```
public void Generate()
```

### StartGenerate()

Declaration

```
public IEnumerator StartGenerate()
```

#### Returns

TYPE	DESCRIPTION
IEnumerator	

# Class TesseraMultipassPass

Inheritance

[object](#)

TesseraMultipassPass

Namespace: [Tessera](#)

Assembly: Tessera.dll

Syntax

```
[Serializable]
public class TesseraMultipassPass
```

Fields

**clearEvent**

Declaration

```
public UnityEvent clearEvent
```

Field Value

TYPE	DESCRIPTION
UnityEvent	

**generateEvent**

Declaration

```
public UnityEvent generateEvent
```

Field Value

TYPE	DESCRIPTION
UnityEvent	

**generator**

Declaration

```
public TesseraGenerator generator
```

Field Value

TYPE	DESCRIPTION
<a href="#">TesseraGenerator</a>	

**passType**

Declaration

```
public TesseraMultipassPassType passType
```

Field Value

TYPE	DESCRIPTION
TesseraMultipassPassType	

# Enum TesseraMultipassPassType

Namespace: [Tessera](#)

Assembly: Tessera.dll

## Syntax

```
public enum TesseraMultipassPassType
```

## Fields

NAME	DESCRIPTION
Event	
Generator	

# Class TesseraPalette

Inheritance

[object](#)

[Object](#)

[ScriptableObject](#)

[TesseraPalette](#)

Implements

[ISerializationCallbackReceiver](#)

Inherited Members

[ScriptableObject.SetDirty\(\)](#)

[ScriptableObject.CreateInstance\(string\)](#)

[ScriptableObject.CreateInstance\(Type\)](#)

[ScriptableObject.CreateInstance<T>\(\)](#)

[Object.GetInstanceID\(\)](#)

[Object.GetHashCode\(\)](#)

[Object.Equals\(object\)](#)

[Object.Instantiate\(Object, Vector3, Quaternion\)](#)

[Object.Instantiate\(Object, Vector3, Quaternion, Transform\)](#)

[Object.Instantiate\(Object\)](#)

[Object.Instantiate\(Object, Transform\)](#)

[Object.Instantiate\(Object, Transform, bool\)](#)

[Object.Instantiate<T>\(T\)](#)

[Object.Instantiate<T>\(T, Vector3, Quaternion\)](#)

[Object.Instantiate<T>\(T, Vector3, Quaternion, Transform\)](#)

[Object.Instantiate<T>\(T, Transform\)](#)

[Object.Instantiate<T>\(T, Transform, bool\)](#)

[Object.Destroy\(Object, float\)](#)

[Object.Destroy\(Object\)](#)

[Object.DestroyImmediate\(Object, bool\)](#)

[Object.DestroyImmediate\(Object\)](#)

[Object.FindObjectsOfType\(Type\)](#)

[Object.DontDestroyOnLoad\(Object\)](#)

[Object.DestroyObject\(Object, float\)](#)

[Object.DestroyObject\(Object\)](#)

[Object.FindSceneObjectsOfType\(Type\)](#)

[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#)

[Object.FindObjectsOfType<T>\(\)](#)

[Object.FindObjectOfType<T>\(\)](#)

[Object.FindObjectsOfTypeAll\(Type\)](#)

[Object.FindObjectOfType\(Type\)](#)

[Object.ToString\(\)](#)

[Object.name](#)

[Object.hideFlags](#)

Namespace: [Tessera](#)

Assembly: Tessera.dll

Syntax

```
[CreateAssetMenu(fileName = "Palette", menuName = "Tessera Palette", order = 1)]
public class TesseraPalette : ScriptableObject, ISerializationCallbackReceiver
```

## Constructors

### TesseraPalette()

Declaration

```
public TesseraPalette()
```

## Fields

### entries

Declaration

```
public List<PaletteEntry> entries
```

Field Value

TYPE	DESCRIPTION
List<PaletteEntry>	

### matchOverrides

Declaration

```
public Dictionary<(int, int), bool> matchOverrides
```

Field Value

TYPE	DESCRIPTION
Dictionary<(int, int), bool>	

## Properties

### defaultPalette

Declaration

```
public static TesseraPalette defaultPalette { get; }
```

Property Value

TYPE	DESCRIPTION
TesseraPalette	

### entryCount

Declaration

```
public int entryCount { get; }
```

Property Value

TYPE	DESCRIPTION
int	

## Methods

### GetColor(int)

Declaration

```
public Color GetColor(int i)
```

Parameters

TYPE	NAME	DESCRIPTION
int	i	

Returns

TYPE	DESCRIPTION
Color	

**GetEntry(int)**

Declaration

```
public PaletteEntry GetEntry(int i)
```

Parameters

TYPE	NAME	DESCRIPTION
int	i	

Returns

TYPE	DESCRIPTION
PaletteEntry	

**Match(int, int)**

Declaration

```
public bool Match(int a, int b)
```

Parameters

TYPE	NAME	DESCRIPTION
int	a	
int	b	

Returns

TYPE	DESCRIPTION
bool	

**Match(FaceDetails, FaceDetails)**

Declaration

```
public bool Match(FaceDetails a, FaceDetails b)
```

Parameters

TYPE	NAME	DESCRIPTION
FaceDetails	a	
FaceDetails	b	

Returns

TYPE	DESCRIPTION
bool	

### OnAfterDeserialize()

Implement this method to receive a callback after Unity deserializes your object.

Declaration

```
public void OnAfterDeserialize()
```

### OnBeforeSerialize()

Implement this method to receive a callback before Unity serializes your object.

Declaration

```
public void OnBeforeSerialize()
```

Implements

UnityEngine.ISerializationCallbackReceiver

# Class TesseraPinConstraint

Inheritance

[object](#)

TesseraPinConstraint

Implements

[ITesseraiInitialConstraint](#)

Namespace: [Tessera](#)

Assembly: Tessera.dll

Syntax

```
public class TesseraPinConstraint : ITesseraiInitialConstraint
```

## Properties

Name

Declaration

```
public string Name { get; }
```

Property Value

TYPE	DESCRIPTION
<a href="#">string</a>	

Implements

[ITesseraiInitialConstraint](#)

# Class TesseraPinned

Inheritance

[object](#)

[Object](#)

[Component](#)

[Behaviour](#)

[MonoBehaviour](#)

[TesseraBaseSceneConstraintComponent](#)

[TesseraPinned](#)

Inherited Members

[MonoBehaviour.IsInvoking\(\)](#)

[MonoBehaviour.CancelInvoke\(\)](#)

[MonoBehaviour.Invoke\(string, float\)](#)

[MonoBehaviour.InvokeRepeating\(string, float, float\)](#)

[MonoBehaviour.CancelInvoke\(string\)](#)

[MonoBehaviour.IsInvoking\(string\)](#)

[MonoBehaviour.StartCoroutine\(string\)](#)

[MonoBehaviour.StartCoroutine\(string, object\)](#)

[MonoBehaviour.StartCoroutine\(IEnumerator\)](#)

[MonoBehaviour.StartCoroutine\\_Auto\(IEnumerator\)](#)

[MonoBehaviour.StopCoroutine\(IEnumerator\)](#)

[MonoBehaviour.StopCoroutine\(Coroutine\)](#)

[MonoBehaviour.StopCoroutine\(string\)](#)

[MonoBehaviour.StopAllCoroutines\(\)](#)

[MonoBehaviour.print\(object\)](#)

[MonoBehaviour.useGUILayout](#)

[MonoBehaviour.runInEditMode](#)

[Behaviour.enabled](#)

[Behaviour.isActiveAndEnabled](#)

[Component.GetComponent\(Type\)](#)

[Component.GetComponent<T>\(\)](#)

[Component.TryGetComponent\(Type, out Component\)](#)

[Component.TryGetComponent<T>\(out T\)](#)

[Component.GetComponent\(string\)](#)

[Component.GetComponentInChildren\(Type, bool\)](#)

[Component.GetComponentInChildren\(Type\)](#)

[Component.GetComponentInChildren<T>\(bool\)](#)

[Component.GetComponentInChildren<T>\(\)](#)

[Component.GetComponentsInChildren\(Type, bool\)](#)

[Component.GetComponentsInChildren\(Type\)](#)

[Component.GetComponentsInChildren<T>\(bool\)](#)

[Component.GetComponentsInChildren<T>\(bool, List<T>\)](#)

[Component.GetComponentsInChildren<T>\(\)](#)

[Component.GetComponentsInChildren<T>\(List<T>\)](#)

[Component.GetComponentInParent\(Type\)](#)

[Component.GetComponentInParent<T>\(\)](#)

[Component.GetComponentsInParent\(Type, bool\)](#)

[Component.GetComponentsInParent\(Type\)](#)

[Component.GetComponentsInParent<T>\(bool\)](#)

[Component.GetComponentsInParent<T>\(bool, List<T>\)](#)

Component.GetComponentsInParent<T>()  
Component.GetComponents(Type)  
Component.GetComponents(Type, List<Component>)  
Component.GetComponents<T>(List<T>)  
Component.GetComponents<T>()  
Component.CompareTag(string)  
Component.SendMessageUpwards(string, object, SendMessageOptions)  
Component.SendMessageUpwards(string, object)  
Component.SendMessageUpwards(string)  
Component.SendMessageUpwards(string, SendMessageOptions)  
Component.SendMessage(string, object)  
Component.SendMessage(string)  
Component.SendMessage(string, object, SendMessageOptions)  
Component.SendMessage(string, SendMessageOptions)  
Component.BroadcastMessage(string, object, SendMessageOptions)  
Component.BroadcastMessage(string, object)  
Component.BroadcastMessage(string)  
Component.BroadcastMessage(string, SendMessageOptions)  
Component.transform  
Component.gameObject  
Component.tag  
Object.GetInstanceID()  
Object.GetHashCode()  
**Object.Equals(object)**  
Object.Instantiate(Object, Vector3, Quaternion)  
Object.Instantiate(Object, Vector3, Quaternion, Transform)  
Object.Instantiate(Object)  
Object.Instantiate(Object, Transform)  
**Object.Instantiate(Object, Transform, bool)**  
Object.Instantiate<T>(T)  
Object.Instantiate<T>(T, Vector3, Quaternion)  
Object.Instantiate<T>(T, Vector3, Quaternion, Transform)  
Object.Instantiate<T>(T, Transform)  
**Object.Instantiate<T>(T, Transform, bool)**  
Object.Destroy(Object, float)  
Object.Destroy(Object)  
Object.DestroyImmediate(Object, bool)  
Object.DestroyImmediate(Object)  
**Object.FindObjectsOfType(Type)**  
Object.DontDestroyOnLoad(Object)  
Object.DestroyObject(Object, float)  
Object.DestroyObject(Object)  
**Object.FindSceneObjectsOfType(Type)**  
**Object.FindObjectsOfTypeIncludingAssets(Type)**  
Object.FindObjectsOfType<T>()  
Object.FindObjectOfType<T>()  
**Object.FindObjectsOfTypeAll(Type)**  
**Object.FindObjectOfType(Type)**  
Object.ToString()  
Object.name  
Object.hideFlags

## Syntax

```
[AddComponentMenu("Tessera/Tessera Pinned", 21)]  
public class TesseraPinned : TesseraBaseSceneConstraintComponent
```

## Fields

**pinType**

Sets the type of pin to apply.

## Declaration

```
public PinType pinType
```

## Field Value

TYPE	DESCRIPTION
PinType	

**tile**

The tile to pin. Defaults to a tile component found on the same GameObject

## Declaration

```
public TesseraTileBase tile
```

## Field Value

TYPE	DESCRIPTION
TesseraTileBase	

# Class TesseraSquareTile

GameObjects with this behaviour record adjacency information for use with a [TesseraGenerator](#).

Inheritance

[object](#)

[Object](#)

[Component](#)

[Behaviour](#)

[MonoBehaviour](#)

[TesseraTileBase](#)

[TesseraSquareTile](#)

Implements

[ISerializationCallbackReceiver](#)

Inherited Members

[TesseraTileBase.palette](#)

[TesseraTileBase.sylvesFaceDetails](#)

[TesseraTileBase.sylvesOffsets](#)

[TesseraTileBase.center](#)

[TesseraTileBase.cellSize](#)

[TesseraTileBase.rotatable](#)

[TesseraTileBase.reflectable](#)

[TesseraTileBase.rotationGroupType](#)

[TesseraTileBase.symmetric](#)

[TesseraTileBase.instantiateChildrenOnly](#)

[TesseraTileBase.Get\(Vector3Int, CellFaceDir\)](#)

[TesseraTileBase.Get\(Vector3Int, CellDir\)](#)

[TesseraTileBase.TryGet\(Vector3Int, CellFaceDir, out FaceDetails\)](#)

[TesseraTileBase.TryGet\(Vector3Int, CellDir, out FaceDetails\)](#)

[TesseraTileBase.GetSharedMesh\(\)](#)

[TesseraTileBase.AddOffset\(Vector3Int\)](#)

[TesseraTileBase.RemoveOffset\(Vector3Int\)](#)

[TesseraTileBase.OnBeforeSerialize\(\)](#)

[TesseraTileBase.OnAfterDeserialize\(\)](#)

[MonoBehaviour.IsInvoking\(\)](#)

[MonoBehaviour.CancelInvoke\(\)](#)

[MonoBehaviour.Invoke\(string, float\)](#)

[MonoBehaviour.InvokeRepeating\(string, float, float\)](#)

[MonoBehaviour.CancelInvoke\(string\)](#)

[MonoBehaviour.IsInvoking\(string\)](#)

[MonoBehaviour.StartCoroutine\(string\)](#)

[MonoBehaviour.StartCoroutine\(string, object\)](#)

[MonoBehaviour.StartCoroutine\(IEnumerator\)](#)

[MonoBehaviour.StartCoroutine\\_Auto\(IEnumerator\)](#)

[MonoBehaviour.StopCoroutine\(IEnumerator\)](#)

[MonoBehaviour.StopCoroutine\(Coroutine\)](#)

[MonoBehaviour.StopCoroutine\(string\)](#)

[MonoBehaviour.StopAllCoroutines\(\)](#)

[MonoBehaviour.print\(object\)](#)

[MonoBehaviour.useGUILayout](#)

[MonoBehaviour.RunWithEditMode](#)

Behaviour.enabled  
Behaviour.isActiveAndEnabled  
**Component.GetComponent(Type)**  
Component.GetComponent<T>()  
**Component.TryGetComponent(Type, out Component)**  
Component.TryGetComponent<T>(out T)  
Component.GetComponent(string)  
Component.GetComponentInChildren(Type, bool)  
Component.GetComponentInChildren(Type)  
**Component.GetComponentInChildren<T>(bool)**  
Component.GetComponentInChildren<T>()  
**Component.GetComponentsInChildren(Type, bool)**  
Component.GetComponentsInChildren(Type)  
Component.GetComponentsInChildren<T>(bool)  
Component.GetComponentsInChildren<T>(bool, List<T>)  
Component.GetComponentsInChildren<T>()  
**Component.GetComponentsInChildren<T>(List<T>)**  
**Component.GetComponentInParent(Type)**  
Component.GetComponentInParent<T>()  
**Component.GetComponentsInParent(Type, bool)**  
Component.GetComponentsInParent(Type)  
Component.GetComponentsInParent<T>(bool)  
Component.GetComponentsInParent<T>(bool, List<T>)  
Component.GetComponentsInParent<T>()  
**Component.GetComponents(Type)**  
**Component.GetComponents(Type, List<Component>)**  
**Component.GetComponents<T>(List<T>)**  
Component.GetComponents<T>()  
**Component.CompareTag(string)**  
Component.SendMessageUpwards(string, object, SendMessageOptions)  
Component.SendMessageUpwards(string, object)  
Component.SendMessageUpwards(string)  
Component.SendMessageUpwards(string, SendMessageOptions)  
Component.SendMessage(string, object)  
Component.SendMessage(string)  
Component.SendMessage(string, object, SendMessageOptions)  
Component.SendMessage(string, SendMessageOptions)  
Component.BroadcastMessage(string, object, SendMessageOptions)  
Component.BroadcastMessage(string, object)  
Component.BroadcastMessage(string)  
**Component.BroadcastMessage(string, SendMessageOptions)**  
Component.transform  
Component.gameObject  
Component.tag  
Object.GetInstanceID()  
Object.GetHashCode()  
**Object.Equals(object)**  
Object.Instantiate(Object, Vector3, Quaternion)  
Object.Instantiate(Object, Vector3, Quaternion, Transform)  
Object.Instantiate(Object)  
Object.Instantiate(Object, Transform)  
**Object.Instantiate(Object, Transform, bool)**

Object.Instantiate<T>(T)  
Object.Instantiate<T>(T, Vector3, Quaternion)  
Object.Instantiate<T>(T, Vector3, Quaternion, Transform)  
Object.Instantiate<T>(T, Transform)  
**Object.Instantiate<T>(T, Transform, bool)**  
**Object.Destroy(Object, float)**  
Object.Destroy(Object)  
**Object.DestroyImmediate(Object, bool)**  
Object.DestroyImmediate(Object)  
**Object.FindObjectsOfType(Type)**  
Object.DontDestroyOnLoad(Object)  
**Object.DestroyObject(Object, float)**  
Object.DestroyObject(Object)  
**Object.FindSceneObjectsOfType(Type)**  
**Object.FindObjectsOfTypeIncludingAssets(Type)**  
Object.FindObjectsOfType<T>()  
Object.FindObjectOfType<T>()  
**Object.FindObjectsOfTypeAll(Type)**  
**Object.FindObjectOfType(Type)**  
Object.ToString()  
Object.name  
Object.hideFlags

Namespace: **Tessera**

Assembly: Tessera.dll

## Syntax

```
[AddComponentMenu("Tessera/Tessera Square Tile")]
public class TesseraSquareTile : TesseraTileBase, ISerializationCallbackReceiver
```

## Constructors

**TesseraSquareTile()**

Declaration

```
public TesseraSquareTile()
```

## Properties

**SubFaceInfo**

Declaration

```
public override ISubFaceInfo SubFaceInfo { get; }
```

Property Value

TYPE	DESCRIPTION
<b>ISubFaceInfo</b>	

Overrides

**TesseraTileBase.SubFaceInfo**

**SylvesCellGrid**

Declaration

```
public override IGrid SylvesCellGrid { get; }
```

Property Value

TYPE	DESCRIPTION
IGrid	

Overrides

[TesseraTileBase.SylvesCellGrid](#)

## SylvesCellType

Declaration

```
public override ICellType SylvesCellType { get; }
```

Property Value

TYPE	DESCRIPTION
ICellType	

Overrides

[TesseraTileBase.SylvesCellType](#)

## Methods

### GetBounds()

Declaration

```
public BoundsInt GetBounds()
```

Returns

TYPE	DESCRIPTION
BoundsInt	

## Implements

UnityEngine.ISerializationCallbackReceiver

# Class TesseraTile

GameObjects with this behaviour record adjacency information for use with a [TesseraGenerator](#).

Inheritance

[object](#)

[Object](#)

[Component](#)

[Behaviour](#)

[MonoBehaviour](#)

[TesseraTileBase](#)

[TesseraTile](#)

Implements

[ISerializationCallbackReceiver](#)

Inherited Members

[TesseraTileBase.palette](#)

[TesseraTileBase.sylvesFaceDetails](#)

[TesseraTileBase.sylvesOffsets](#)

[TesseraTileBase.center](#)

[TesseraTileBase.cellSize](#)

[TesseraTileBase.rotatable](#)

[TesseraTileBase.reflectable](#)

[TesseraTileBase.rotationGroupType](#)

[TesseraTileBase.symmetric](#)

[TesseraTileBase.instantiateChildrenOnly](#)

[TesseraTileBase.Get\(Vector3Int, CellFaceDir\)](#)

[TesseraTileBase.Get\(Vector3Int, CellDir\)](#)

[TesseraTileBase.TryGet\(Vector3Int, CellFaceDir, out FaceDetails\)](#)

[TesseraTileBase.TryGet\(Vector3Int, CellDir, out FaceDetails\)](#)

[TesseraTileBase.GetSharedMesh\(\)](#)

[TesseraTileBase.AddOffset\(Vector3Int\)](#)

[TesseraTileBase.RemoveOffset\(Vector3Int\)](#)

[TesseraTileBase.OnBeforeSerialize\(\)](#)

[TesseraTileBase.OnAfterDeserialize\(\)](#)

[MonoBehaviour.IsInvoking\(\)](#)

[MonoBehaviour.CancelInvoke\(\)](#)

[MonoBehaviour.Invoke\(string, float\)](#)

[MonoBehaviour.InvokeRepeating\(string, float, float\)](#)

[MonoBehaviour.CancelInvoke\(string\)](#)

[MonoBehaviour.IsInvoking\(string\)](#)

[MonoBehaviour.StartCoroutine\(string\)](#)

[MonoBehaviour.StartCoroutine\(string, object\)](#)

[MonoBehaviour.StartCoroutine\(IEnumerator\)](#)

[MonoBehaviour.StartCoroutine\\_Auto\(IEnumerator\)](#)

[MonoBehaviour.StopCoroutine\(IEnumerator\)](#)

[MonoBehaviour.StopCoroutine\(Coroutine\)](#)

[MonoBehaviour.StopCoroutine\(string\)](#)

[MonoBehaviour.StopAllCoroutines\(\)](#)

[MonoBehaviour.print\(object\)](#)

[MonoBehaviour.useGUILayout](#)

[MonoBehaviour.RunWithEditMode](#)

Behaviour.enabled  
Behaviour.isActiveAndEnabled  
**Component.GetComponent(Type)**  
Component.GetComponent<T>()  
**Component.TryGetComponent(Type, out Component)**  
Component.TryGetComponent<T>(out T)  
Component.GetComponent(string)  
Component.GetComponentInChildren(Type, bool)  
Component.GetComponentInChildren(Type)  
**Component.GetComponentInChildren<T>(bool)**  
Component.GetComponentInChildren<T>()  
**Component.GetComponentsInChildren(Type, bool)**  
Component.GetComponentsInChildren(Type)  
Component.GetComponentsInChildren<T>(bool)  
Component.GetComponentsInChildren<T>(bool, List<T>)  
Component.GetComponentsInChildren<T>()  
**Component.GetComponentsInChildren<T>(List<T>)**  
**Component.GetComponentInParent(Type)**  
Component.GetComponentInParent<T>()  
**Component.GetComponentsInParent(Type, bool)**  
Component.GetComponentsInParent(Type)  
Component.GetComponentsInParent<T>(bool)  
Component.GetComponentsInParent<T>(bool, List<T>)  
Component.GetComponentsInParent<T>()  
**Component.GetComponents(Type)**  
**Component.GetComponents(Type, List<Component>)**  
**Component.GetComponents<T>(List<T>)**  
Component.GetComponents<T>()  
**Component.CompareTag(string)**  
Component.SendMessageUpwards(string, object, SendMessageOptions)  
Component.SendMessageUpwards(string, object)  
Component.SendMessageUpwards(string)  
Component.SendMessageUpwards(string, SendMessageOptions)  
Component.SendMessage(string, object)  
Component.SendMessage(string)  
Component.SendMessage(string, object, SendMessageOptions)  
Component.SendMessage(string, SendMessageOptions)  
Component.BroadcastMessage(string, object, SendMessageOptions)  
Component.BroadcastMessage(string, object)  
Component.BroadcastMessage(string)  
**Component.BroadcastMessage(string, SendMessageOptions)**  
Component.transform  
Component.gameObject  
Component.tag  
Object.GetInstanceID()  
Object.GetHashCode()  
**Object.Equals(object)**  
Object.Instantiate(Object, Vector3, Quaternion)  
Object.Instantiate(Object, Vector3, Quaternion, Transform)  
Object.Instantiate(Object)  
Object.Instantiate(Object, Transform)  
**Object.Instantiate(Object, Transform, bool)**

Object.Instantiate<T>(T)  
Object.Instantiate<T>(T, Vector3, Quaternion)  
Object.Instantiate<T>(T, Vector3, Quaternion, Transform)  
Object.Instantiate<T>(T, Transform)  
**Object.Instantiate<T>(T, Transform, bool)**  
**Object.Destroy(Object, float)**  
Object.Destroy(Object)  
**Object.DestroyImmediate(Object, bool)**  
Object.DestroyImmediate(Object)  
**Object.FindObjectsOfType(Type)**  
Object.DontDestroyOnLoad(Object)  
**Object.DestroyObject(Object, float)**  
Object.DestroyObject(Object)  
**Object.FindSceneObjectsOfType(Type)**  
**Object.FindObjectsOfTypeIncludingAssets(Type)**  
Object.FindObjectsOfType<T>()  
Object.FindObjectOfType<T>()  
**Object.FindObjectsOfTypeAll(Type)**  
**Object.FindObjectOfType(Type)**  
Object.ToString()  
Object.name  
Object.hideFlags

Namespace: **Tessera**

Assembly: Tessera.dll

## Syntax

```
[AddComponentMenu("Tessera/Tessera Tile")]
public class TesseraTile : TesseraTileBase, ISerializationCallbackReceiver
```

## Constructors

**TesseraTile()**

Declaration

```
public TesseraTile()
```

## Properties

**SubFaceInfo**

Declaration

```
public override ISubFaceInfo SubFaceInfo { get; }
```

Property Value

TYPE	DESCRIPTION
<b>ISubFaceInfo</b>	

Overrides

**TesseraTileBase.SubFaceInfo**

**SylvesCellGrid**

Declaration

```
public override IGrid SylvesCellGrid { get; }
```

#### Property Value

TYPE	DESCRIPTION
IGrid	

#### Overrides

[TesseraTileBase.SylvesCellGrid](#)

### SylvesCellType

#### Declaration

```
public override ICellType SylvesCellType { get; }
```

#### Property Value

TYPE	DESCRIPTION
ICellType	

#### Overrides

[TesseraTileBase.SylvesCellType](#)

### Methods

#### GetBounds()

#### Declaration

```
public BoundsInt GetBounds()
```

#### Returns

TYPE	DESCRIPTION
BoundsInt	

### Implements

UnityEngine.ISerializationCallbackReceiver

# Class TesseraTileBase

Inheritance

[object](#)  
[Object](#)  
[Component](#)  
[Behaviour](#)  
[MonoBehaviour](#)  
[TesseraTileBase](#)  
[TesseraHexTile](#)  
[TesseraSquareTile](#)  
[TesseraTile](#)  
[TesseraTrianglePrismTile](#)

Implements

[ISerializationCallbackReceiver](#)

Inherited Members

[MonoBehaviour.IsInvoking\(\)](#)  
[MonoBehaviour.CancelInvoke\(\)](#)  
[MonoBehaviour.Invoke\(string, float\)](#)  
[MonoBehaviour.InvokeRepeating\(string, float, float\)](#)  
[MonoBehaviour.CancelInvoke\(string\)](#)  
[MonoBehaviour.IsInvoking\(string\)](#)  
[MonoBehaviour.StartCoroutine\(string\)](#)  
[MonoBehaviour.StartCoroutine\(string, object\)](#)  
[MonoBehaviour.StartCoroutine\(IEnumerator\)](#)  
[MonoBehaviour.StartCoroutine\\_Auto\(IEnumerator\)](#)  
[MonoBehaviour.StopCoroutine\(IEnumerator\)](#)  
[MonoBehaviour.StopCoroutine\(Coroutine\)](#)  
[MonoBehaviour.StopCoroutine\(string\)](#)  
[MonoBehaviour.StopAllCoroutines\(\)](#)  
[MonoBehaviour.print\(object\)](#)  
[MonoBehaviour.useGUILayout](#)  
[MonoBehaviour.runInEditMode](#)  
[Behaviour.enabled](#)  
[Behaviour.isActiveAndEnabled](#)  
[Component.GetComponent\(Type\)](#)  
[Component.GetComponent<T>\(\)](#)  
[Component.TryGetComponent\(Type, out Component\)](#)  
[Component.TryGetComponent<T>\(out T\)](#)  
[Component.GetComponent\(string\)](#)  
[Component.GetComponentInChildren\(Type, bool\)](#)  
[Component.GetComponentInChildren\(Type\)](#)  
[Component.GetComponentInChildren<T>\(bool\)](#)  
[Component.GetComponentInChildren<T>\(\)](#)  
[Component.GetComponentsInChildren\(Type, bool\)](#)  
[Component.GetComponentsInChildren\(Type\)](#)  
[Component.GetComponentsInChildren<T>\(bool\)](#)  
[Component.GetComponentsInChildren<T>\(bool, List<T>\)](#)  
[Component.GetComponentsInChildren<T>\(\)](#)  
[Component.GetComponentsInChildren<T>\(List<T>\)](#)  
[Component.GetComponentInParent\(Type\)](#)

```
Component.GetComponentInParent<T>()
Component.GetComponentsInParent(Type, bool)
Component.GetComponentsInParent(Type)
Component.GetComponentsInParent<T>(bool)
Component.GetComponentsInParent<T>(bool, List<T>)
Component.GetComponentsInParent<T>()
Component.GetComponents(Type)
Component.GetComponents(Type, List<Component>)
Component.GetComponents<T>(List<T>)
Component.GetComponents<T>()
Component.CompareTag(string)
Component.SendMessageUpwards(string, object, SendMessageOptions)
Component.SendMessageUpwards(string, object)
Component.SendMessageUpwards(string)
Component.SendMessageUpwards(string, SendMessageOptions)
Component.SendMessage(string, object)
Component.SendMessage(string)
Component.SendMessage(string, object, SendMessageOptions)
Component.SendMessage(string, SendMessageOptions)
Component.BroadcastMessage(string, object, SendMessageOptions)
Component.BroadcastMessage(string, object)
Component.BroadcastMessage(string)
Component.BroadcastMessage(string, SendMessageOptions)
Component.transform
Component.gameObject
Component.tag
Object.GetInstanceID()
Object.GetHashCode()
Object.Equals(object)
Object.Instantiate(Object, Vector3, Quaternion)
Object.Instantiate(Object, Vector3, Quaternion, Transform)
Object.Instantiate(Object)
Object.Instantiate(Object, Transform)
Object.Instantiate(Object, Transform, bool)
Object.Instantiate<T>(T)
Object.Instantiate<T>(T, Vector3, Quaternion)
Object.Instantiate<T>(T, Vector3, Quaternion, Transform)
Object.Instantiate<T>(T, Transform)
Object.Instantiate<T>(T, Transform, bool)
Object.Destroy(Object, float)
Object.Destroy(Object)
Object.DestroyImmediate(Object, bool)
Object.DestroyImmediate(Object)
Object.FindObjectsOfType(Type)
Object.DontDestroyOnLoad(Object)
Object.DestroyObject(Object, float)
Object.DestroyObject(Object)
Object.FindSceneObjectsOfType(Type)
Object.FindObjectsOfTypeIncludingAssets(Type)
Object.FindObjectsOfType<T>()
ObjectFindObjectOfType<T>()
Object.FindObjectsOfTypeAll(Type)
```

`Object.FindObjectOfType(Type)`

`Object.ToString()`

`Object.name`

`Object.hideFlags`

Namespace: [Tessera](#)

Assembly: Tessera.dll

Syntax

```
public abstract class TesseraTileBase : MonoBehaviour, ISerializationCallbackReceiver
```

Fields

`cellSize`

The size of one cell in the tile. NB: This field is only used in the Editor - you must set `cellSize` to match.

Declaration

```
[Tooltip("The size of one cell in the tile.")]
[FormerlySerializedAs("tileSize")]
public Vector3 cellSize
```

Field Value

TYPE	DESCRIPTION
Vector3	

`center`

Where the center of tile is. For big tiles that occupy more than one cell, it's the center of the cell with offset (0, 0, 0). The tile may not actually occupy that cell!

Declaration

```
[Tooltip("Where the center of tile is.")]
public Vector3 center
```

Field Value

TYPE	DESCRIPTION
Vector3	

`instantiateChildrenOnly`

If set, when being instantiated by a Generator, only children will get constructed. If there are no children, then this effectively disables the tile from instantiation.

Declaration

```
[Tooltip("If set, when being instantiated by a Generator, only children will get constructed.")]
public bool instantiateChildrenOnly
```

Field Value

TYPE	DESCRIPTION
bool	

## palette

Set this to control the colors and names used for painting on the tile. Defaults to [defaultPalette](#).

Declaration

```
[Tooltip("Set this to control the colors and names used for painting on the tile.")]  
public TesseraPalette palette
```

Field Value

TYPE	DESCRIPTION
TesseraPalette	

## reflectable

If true, when generating, reflections in the x-axis will be used.

Declaration

```
[Tooltip("If true, when generating, reflections in the x-axis will be used.")]  
public bool reflectable
```

Field Value

TYPE	DESCRIPTION
bool	

## rotatable

If true, when generating, rotations of the tile will be used.

Declaration

```
[Tooltip("If true, when generating, rotations of the tile will be used.")]  
public bool rotatable
```

Field Value

TYPE	DESCRIPTION
bool	

## rotationGroupType

If rotatable is on, specifies what sorts of rotations are used.

Declaration

```
[Tooltip("If rotatable is on, specifies what sorts of rotations are used.")]  
public RotationGroupType rotationGroupType
```

Field Value

TYPE	DESCRIPTION
RotationGroupType	

## sylvesFaceDetails

A list of outward facing faces. For a normal cube tile, there are 6 faces. Each face contains adjacency information that indicates what other tiles can connect to it. It is recommended you only edit this via the Unity Editor, or [Get\(Vector3Int, CellFaceDir\)](#) and [AddOffset\(Vector3Int\)](#)

Declaration

```
[NonSerialized]
public List<SylvesOrientedFace> sylvesFaceDetails
```

Field Value

TYPE	DESCRIPTION
List<SylvesOrientedFace>	

sylvesOffsets

A list of cells that this tile occupies. For a normal cube tile, this just contains Vector3Int.zero, but it will be more for "big" tiles. It is recommended you only edit this via the Unity Editor, or [AddOffset\(Vector3Int\)](#) and [RemoveOffset\(Vector3Int\)](#)

Declaration

```
[NonSerialized]
public List<Vector3Int> sylvesOffsets
```

Field Value

TYPE	DESCRIPTION
List<Vector3Int>	

symmetric

If true, Tessera assumes that the tile paint matches the symmetry of the tile. Disable this if there are important details of your tile that the paint doesn't show. Turning symmetric on can have some performance benefits, and affects the behaviour of the mirror constraint.

Declaration

```
[Tooltip("If true, Tessera assumes that the tile paint matches the the symmetry of the tile.\nIf false, assumes every possible rotation is unique and not interchangeable.")]
public bool symmetric
```

Field Value

TYPE	DESCRIPTION
bool	

Properties

SubFaceInfo

Declaration

```
public abstract ISubFaceInfo SubFaceInfo { get; }
```

Property Value

TYPE	DESCRIPTION
ISubFaceInfo	

## SylvesCellGrid

### Declaration

```
public abstract IGrid SylvesCellGrid { get; }
```

### Property Value

TYPE	DESCRIPTION
IGrid	

## SylvesCellType

### Declaration

```
public abstract ICellType SylvesCellType { get; }
```

### Property Value

TYPE	DESCRIPTION
ICellType	

## Methods

### AddOffset(Vector3Int)

Configures the tile as a "big" tile that occupies several cells. Keeps [sylvesOffsets](#) and [sylvesFaceDetails](#) in sync.

### Declaration

```
public void AddOffset(Vector3Int o)
```

### Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	o	

### Get(Vector3Int, CellDir)

Finds the face details for a cell with a given offset.

### Declaration

```
public FaceDetails Get(Vector3Int offset, CellDir dir)
```

### Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	offset	
CellDir	dir	

Returns

TYPE	DESCRIPTION
FaceDetails	

## Get(Vector3Int, CellFaceDir)

Finds the face details for a cell with a given offset.

Declaration

```
public FaceDetails Get(Vector3Int offset, CellFaceDir faceDir)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	offset	
CellFaceDir	faceDir	

Returns

TYPE	DESCRIPTION
FaceDetails	

## GetSharedMesh()

Finds the mesh associated with this tile, or null

Declaration

```
public Mesh GetSharedMesh()
```

Returns

TYPE	DESCRIPTION
Mesh	

## OnAfterDeserialize()

Implement this method to receive a callback after Unity deserializes your object.

Declaration

```
public void OnAfterDeserialize()
```

## OnBeforeSerialize()

Implement this method to receive a callback before Unity serializes your object.

Declaration

```
public void OnBeforeSerialize()
```

## RemoveOffset(Vector3Int)

Configures the tile as a "big" tile that occupies several cells. Keeps [sylvesOffsets](#) and [sylvesFaceDetails](#) in sync.

## Declaration

```
public void RemoveOffset(Vector3Int o)
```

## Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	o	

## TryGet(Vector3Int, CellDir, out FaceDetails)

Finds the face details for a cell with a given offset.

## Declaration

```
public bool TryGet(Vector3Int offset, CellDir faceDir, out FaceDetails details)
```

## Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	offset	
CellDir	faceDir	
FaceDetails	details	

## Returns

TYPE	DESCRIPTION
bool	

## TryGet(Vector3Int, CellFaceDir, out FaceDetails)

Finds the face details for a cell with a given offset.

## Declaration

```
public bool TryGet(Vector3Int offset, CellFaceDir faceDir, out FaceDetails details)
```

## Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	offset	
CellFaceDir	faceDir	
FaceDetails	details	

## Returns

TYPE	DESCRIPTION
bool	

## Implements

UnityEngine.ISerializationCallbackReceiver

# Class TesseraTileInstance

Represents a request to instantiate a TesseraTile, post generation.

Inheritance

[object](#)

TesseraTileInstance

Namespace: [Tessera](#)

Assembly: Tessera.dll

Syntax

```
public class TesseraTileInstance
```

## Properties

### Cell

The grid cell this instance fills. (for big tiles, this is the cell of the first offset)

Declaration

```
public Vector3Int Cell { get; }
```

Property Value

TYPE	DESCRIPTION
Vector3Int	

### CellRotation

The rotation this instance is placed at (for big tiles, this is the cell of the first offset)

Declaration

```
public CellRotation CellRotation { get; }
```

Property Value

TYPE	DESCRIPTION
CellRotation	

### CellRotations

The rotations this instance instance, in the same order as the tile offsets. Most grids will have same rotation for all offsets.

Declaration

```
public CellRotation[] CellRotations { get; }
```

Property Value

TYPE	DESCRIPTION
CellRotation[]	

### Cells

The cells this instance fills, in the same order as the tile offsets.

## Declaration

```
public Vector3Int[] Cells { get; }
```

## Property Value

TYPE	DESCRIPTION
Vector3Int[]	

## LocalPosition

### Declaration

```
public Vector3 LocalPosition { get; }
```

## Property Value

TYPE	DESCRIPTION
Vector3	

## LocalRotation

### Declaration

```
public Quaternion LocalRotation { get; }
```

## Property Value

TYPE	DESCRIPTION
Quaternion	

## LocalScale

### Declaration

```
public Vector3 LocalScale { get; }
```

## Property Value

TYPE	DESCRIPTION
Vector3	

## LossyScale

### Declaration

```
public Vector3 LossyScale { get; }
```

## Property Value

TYPE	DESCRIPTION
Vector3	

## MeshDeformation

Gives a mesh deformation from tile space to generator space. Null for grids that do not have deformed tiles.

## Declaration

```
public Deformation MeshDeformation { get; }
```

## Property Value

TYPE	DESCRIPTION
Deformation	

## Position

### Declaration

```
public Vector3 Position { get; }
```

## Property Value

TYPE	DESCRIPTION
Vector3	

## Rotation

### Declaration

```
public Quaternion Rotation { get; }
```

## Property Value

TYPE	DESCRIPTION
Quaternion	

## Tile

### Declaration

```
public TesseraTileBase Tile { get; }
```

## Property Value

TYPE	DESCRIPTION
TesseraTileBase	

## Methods

### Align(TRS)

Sets Position/Rotation/Scale from the local versions and a given transform

### Declaration

```
public void Align(TRS transform)
```

## Parameters

TYPE	NAME	DESCRIPTION
TRS	transform	

## Clone()

### Declaration

```
public TesseraTileInstance Clone()
```

### Returns

TYPE	DESCRIPTION
TesseraTileInstance	

# Class TesseraTilemapOutput

Attach this to a TesseraGenerator to output the tiles to a Unity Tilemap component instead of directly instantiating them.

## NOTE

This class is available only in Tessera Pro

Inheritance

[object](#)

[Object](#)

[Component](#)

[Behaviour](#)

[MonoBehaviour](#)

[TesseraTilemapOutput](#)

Implements

[ITesseraTileOutput](#)

Inherited Members

[MonoBehaviour.IsInvoking\(\)](#)

[MonoBehaviour.CancelInvoke\(\)](#)

[MonoBehaviour.Invoke\(string, float\)](#)

[MonoBehaviour.InvokeRepeating\(string, float, float\)](#)

[MonoBehaviour.CancelInvoke\(string\)](#)

[MonoBehaviour.IsInvoking\(string\)](#)

[MonoBehaviour.StartCoroutine\(string\)](#)

[MonoBehaviour.StartCoroutine\(string, object\)](#)

[MonoBehaviour.StartCoroutine\(IEnumerator\)](#)

[MonoBehaviour.StartCoroutine\\_Auto\(IEnumerator\)](#)

[MonoBehaviour.StopCoroutine\(IEnumerator\)](#)

[MonoBehaviour.StopCoroutine\(Coroutine\)](#)

[MonoBehaviour.StopCoroutine\(string\)](#)

[MonoBehaviour.StopAllCoroutines\(\)](#)

[MonoBehaviour.print\(object\)](#)

[MonoBehaviour.useGUILayout](#)

[MonoBehaviour.runInEditMode](#)

[Behaviour.enabled](#)

[Behaviour.isActiveAndEnabled](#)

[Component.GetComponent\(Type\)](#)

[Component.GetComponent<T>\(\)](#)

[Component.TryGetComponent\(Type, out Component\)](#)

[Component.TryGetComponent<T>\(out T\)](#)

[Component.GetComponent\(string\)](#)

[Component.GetComponentInChildren\(Type, bool\)](#)

[Component.GetComponentInChildren\(Type\)](#)

[Component.GetComponentInChildren<T>\(bool\)](#)

[Component.GetComponentInChildren<T>\(\)](#)

[Component.GetComponentsInChildren\(Type, bool\)](#)

[Component.GetComponentsInChildren\(Type\)](#)

[Component.GetComponentsInChildren<T>\(bool\)](#)

[Component.GetComponentsInChildren<T>\(bool, List<T>\)](#)

[Component.GetComponentsInChildren<T>\(\)](#)

```
Component.GetComponentsInChildren<T>(List<T>)
Component.GetComponentInParent(Type)
Component.GetComponentInParent<T>()
Component.GetComponentsInParent(Type, bool)
Component.GetComponentsInParent(Type)
Component.GetComponentsInParent<T>(bool)
Component.GetComponentsInParent<T>(bool, List<T>)
Component.GetComponentsInParent<T>()
Component.GetComponents(Type)
Component.GetComponents(Type, List<Component>)
Component.GetComponents<T>(List<T>)
Component.GetComponents<T>()
Component.CompareTag(string)
Component.SendMessageUpwards(string, object, SendMessageOptions)
Component.SendMessageUpwards(string, object)
Component.SendMessageUpwards(string)
Component.SendMessageUpwards(string, SendMessageOptions)
Component.SendMessage(string, object)
Component.SendMessage(string)
Component.SendMessage(string, object, SendMessageOptions)
Component.SendMessage(string, SendMessageOptions)
Component.BroadcastMessage(string, object, SendMessageOptions)
Component.BroadcastMessage(string, object)
Component.BroadcastMessage(string)
Component.BroadcastMessage(string, SendMessageOptions)
Component.transform
Component.gameObject
Component.tag
Object.GetInstanceID()
Object.GetHashCode()
Object.Equals(object)
Object.Instantiate(Object, Vector3, Quaternion)
Object.Instantiate(Object, Vector3, Quaternion, Transform)
Object.Instantiate(Object)
Object.Instantiate(Object, Transform)
Object.Instantiate(Object, Transform, bool)
Object.Instantiate<T>(T)
Object.Instantiate<T>(T, Vector3, Quaternion)
Object.Instantiate<T>(T, Vector3, Quaternion, Transform)
Object.Instantiate<T>(T, Transform)
Object.Instantiate<T>(T, Transform, bool)
Object.Destroy(Object, float)
Object.Destroy(Object)
Object.DestroyImmediate(Object, bool)
Object.DestroyImmediate(Object)
Object.FindObjectsOfType(Type)
Object.DontDestroyOnLoad(Object)
Object.DestroyObject(Object, float)
Object.DestroyObject(Object)
Object.FindSceneObjectsOfType(Type)
Object.FindObjectsOfTypeIncludingAssets(Type)
Object.FindObjectsOfType<T>()
```

`Object.FindObjectOfType<T>()`  
`Object.FindObjectsOfTypeAll(Type)`  
`Object.FindObjectOfType(Type)`

`Object.ToString()`  
`Object.name`  
`Object.hideFlags`

Namespace: [Tessera](#)  
Assembly: Tessera.dll

#### Syntax

```
[RequireComponent(typeof(TesseraGenerator))]  
[AddComponentMenu("Tessera/Tessera Tilemap Output", 40)]  
public class TesseraTilemapOutput : MonoBehaviour, ITesseraTileOutput
```

#### Fields

##### `tilemap`

The tilemap to write results to.

##### Declaration

```
public Tilemap tilemap
```

##### Field Value

TYPE	DESCRIPTION
Tilemap	

##### `useSprites`

If true, TesseraTiles that have a SpriteRenderer will be recorded to the Tilemap as that sprite. This is more efficient, but you will lose any other components on the object.

##### Declaration

```
public bool useSprites
```

##### Field Value

TYPE	DESCRIPTION
bool	

##### `useWorld`

If true, tiles will be transformed to align with the world space position of the generator.

##### Declaration

```
public bool useWorld
```

##### Field Value

TYPE	DESCRIPTION
bool	

#### Properties

## IsEmpty

Is the output currently empty.

Declaration

```
public bool IsEmpty { get; }
```

Property Value

TYPE	DESCRIPTION
bool	

## SupportsIncremental

Is this output safe to use with AnimatedGenerator

Declaration

```
public bool SupportsIncremental { get; }
```

Property Value

TYPE	DESCRIPTION
bool	

## Methods

### ClearTiles(IEngineInterface)

Clear the output

Declaration

```
public void ClearTiles(IEngineInterface engine)
```

Parameters

TYPE	NAME	DESCRIPTION
IEngineInterface	engine	

### UpdateTiles(TesseraCompletion, IEngineInterface)

Update a chunk of tiles. If incremental updates are supported, then:

- Tiles can replace other tiles, as indicated by the [Cells](#) field.
- A tile of null indicates that the tile should be erased

Declaration

```
public void UpdateTiles(TesseraCompletion completion, IEngineInterface engine)
```

Parameters

TYPE	NAME	DESCRIPTION
TesseraCompletion	completion	

TYPE	NAME	DESCRIPTION
IEngineInterface	engine	

Implements

ITesseraTileOutput

# Class TesseraTransformedTile

Defines a Unity tile that has a specific transform applied to it. Used by [TesseraTilemapOutput](#)

Inheritance

[object](#)

[Object](#)

[ScriptableObject](#)

[TileBase](#)

[Tile](#)

[TesseraTransformedTile](#)

Inherited Members

[Tile.GetTileData\(Vector3Int, ITilemap, ref TileData\)](#)

[Tile.sprite](#)

[Tile.color](#)

[Tile.transform](#)

[Tile.gameObject](#)

[Tile.flags](#)

[Tile.colliderType](#)

[TileBase.RefreshTile\(Vector3Int, ITilemap\)](#)

[TileBase.GetTileAnimationData\(Vector3Int, ITilemap, ref TileAnimationData\)](#)

[ScriptableObject.SetDirty\(\)](#)

[ScriptableObject.CreateInstance\(string\)](#)

[ScriptableObject.CreateInstance\(Type\)](#)

[ScriptableObject.CreateInstance<T>\(\)](#)

[Object.GetInstanceID\(\)](#)

[Object.GetHashCode\(\)](#)

[Object.Equals\(object\)](#)

[Object.Instantiate\(Object, Vector3, Quaternion\)](#)

[Object.Instantiate\(Object, Vector3, Quaternion, Transform\)](#)

[Object.Instantiate\(Object\)](#)

[Object.Instantiate\(Object, Transform\)](#)

[Object.Instantiate\(Object, Transform, bool\)](#)

[Object.Instantiate<T>\(T\)](#)

[Object.Instantiate<T>\(T, Vector3, Quaternion\)](#)

[Object.Instantiate<T>\(T, Vector3, Quaternion, Transform\)](#)

[Object.Instantiate<T>\(T, Transform\)](#)

[Object.Instantiate<T>\(T, Transform, bool\)](#)

[Object.Destroy\(Object, float\)](#)

[Object.Destroy\(Object\)](#)

[Object.DestroyImmediate\(Object, bool\)](#)

[Object.DestroyImmediate\(Object\)](#)

[Object.FindObjectsOfType\(Type\)](#)

[Object.DontDestroyOnLoad\(Object\)](#)

[Object.DestroyObject\(Object, float\)](#)

[Object.DestroyObject\(Object\)](#)

[Object.FindSceneObjectsOfType\(Type\)](#)

[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#)

[Object.FindObjectsOfType<T>\(\)](#)

[ObjectFindObjectOfType<T>\(\)](#)

[Object.FindObjectsOfTypeAll\(Type\)](#)

[Object.FindObjectOfType\(Type\)](#)

Object.ToString()

Object.name

Object.hideFlags

Namespace: [Tessera](#)

Assembly: Tessera.dll

Syntax

```
public class TesseraTransformedTile : Tile
```

## Fields

**localScale**

Declaration

```
public Vector3 localScale
```

Field Value

TYPE	DESCRIPTION
Vector3	

**position**

Declaration

```
public Vector3 position
```

Field Value

TYPE	DESCRIPTION
Vector3	

**rotation**

Declaration

```
public Quaternion rotation
```

Field Value

TYPE	DESCRIPTION
Quaternion	

**useWorld**

Declaration

```
public bool useWorld
```

Field Value

TYPE	DESCRIPTION
bool	

## Methods

## StartUp(Vector3Int, ITilemap, GameObject)

StartUp is called on the first frame of the running Scene.

### Declaration

```
public override bool StartUp(Vector3Int position, ITilemap tilemap, GameObject go)
```

### Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	position	Position of the Tile on the Tilemap.
ITilemap	tilemap	The Tilemap the tile is present on.
GameObject	go	The GameObject instantiated for the Tile.

### Returns

TYPE	DESCRIPTION
bool	Whether the call was successful.

### Overrides

UnityEngine.Tilemaps.TileBase.StartUp(UnityEngine.Vector3Int, UnityEngine.Tilemaps.ITilemap, UnityEngine.GameObject)

# Class TesseraTrianglePrismTile

GameObjects with this behaviour record adjacency information for use with a [TesseraGenerator](#).

Inheritance

[object](#)

[Object](#)

[Component](#)

[Behaviour](#)

[MonoBehaviour](#)

[TesseraTileBase](#)

[TesseraTrianglePrismTile](#)

Implements

[ISerializationCallbackReceiver](#)

Inherited Members

[TesseraTileBase.palette](#)

[TesseraTileBase.sylvesFaceDetails](#)

[TesseraTileBase.sylvesOffsets](#)

[TesseraTileBase.center](#)

[TesseraTileBase.cellSize](#)

[TesseraTileBase.rotatable](#)

[TesseraTileBase.reflectable](#)

[TesseraTileBase.rotationGroupType](#)

[TesseraTileBase.symmetric](#)

[TesseraTileBase.instantiateChildrenOnly](#)

[TesseraTileBase.Get\(Vector3Int, CellFaceDir\)](#)

[TesseraTileBase.Get\(Vector3Int, CellDir\)](#)

[TesseraTileBase.TryGet\(Vector3Int, CellFaceDir, out FaceDetails\)](#)

[TesseraTileBase.TryGet\(Vector3Int, CellDir, out FaceDetails\)](#)

[TesseraTileBase.GetSharedMesh\(\)](#)

[TesseraTileBase.AddOffset\(Vector3Int\)](#)

[TesseraTileBase.RemoveOffset\(Vector3Int\)](#)

[TesseraTileBase.OnBeforeSerialize\(\)](#)

[TesseraTileBase.OnAfterDeserialize\(\)](#)

[MonoBehaviour.IsInvoking\(\)](#)

[MonoBehaviour.CancelInvoke\(\)](#)

[MonoBehaviour.Invoke\(string, float\)](#)

[MonoBehaviour.InvokeRepeating\(string, float, float\)](#)

[MonoBehaviour.CancelInvoke\(string\)](#)

[MonoBehaviour.IsInvoking\(string\)](#)

[MonoBehaviour.StartCoroutine\(string\)](#)

[MonoBehaviour.StartCoroutine\(string, object\)](#)

[MonoBehaviour.StartCoroutine\(IEnumerator\)](#)

[MonoBehaviour.StartCoroutine\\_Auto\(IEnumerator\)](#)

[MonoBehaviour.StopCoroutine\(IEnumerator\)](#)

[MonoBehaviour.StopCoroutine\(Coroutine\)](#)

[MonoBehaviour.StopCoroutine\(string\)](#)

[MonoBehaviour.StopAllCoroutines\(\)](#)

[MonoBehaviour.print\(object\)](#)

[MonoBehaviour.useGUILayout](#)

[MonoBehaviour.RunWithEditMode](#)

Behaviour.enabled  
Behaviour.isActiveAndEnabled  
**Component.GetComponent(Type)**  
Component.GetComponent<T>()  
**Component.TryGetComponent(Type, out Component)**  
Component.TryGetComponent<T>(out T)  
Component.GetComponent(string)  
Component.GetComponentInChildren(Type, bool)  
Component.GetComponentInChildren(Type)  
**Component.GetComponentInChildren<T>(bool)**  
Component.GetComponentInChildren<T>()  
**Component.GetComponentsInChildren(Type, bool)**  
Component.GetComponentsInChildren(Type)  
Component.GetComponentsInChildren<T>(bool)  
Component.GetComponentsInChildren<T>(bool, List<T>)  
Component.GetComponentsInChildren<T>()  
**Component.GetComponentsInChildren<T>(List<T>)**  
**Component.GetComponentInParent(Type)**  
Component.GetComponentInParent<T>()  
**Component.GetComponentsInParent(Type, bool)**  
Component.GetComponentsInParent(Type)  
Component.GetComponentsInParent<T>(bool)  
Component.GetComponentsInParent<T>(bool, List<T>)  
Component.GetComponentsInParent<T>()  
**Component.GetComponents(Type)**  
**Component.GetComponents(Type, List<Component>)**  
**Component.GetComponents<T>(List<T>)**  
Component.GetComponents<T>()  
**Component.CompareTag(string)**  
Component.SendMessageUpwards(string, object, SendMessageOptions)  
Component.SendMessageUpwards(string, object)  
Component.SendMessageUpwards(string)  
Component.SendMessageUpwards(string, SendMessageOptions)  
Component.SendMessage(string, object)  
Component.SendMessage(string)  
Component.SendMessage(string, object, SendMessageOptions)  
Component.SendMessage(string, SendMessageOptions)  
Component.BroadcastMessage(string, object, SendMessageOptions)  
Component.BroadcastMessage(string, object)  
Component.BroadcastMessage(string)  
**Component.BroadcastMessage(string, SendMessageOptions)**  
Component.transform  
Component.gameObject  
Component.tag  
Object.GetInstanceID()  
Object.GetHashCode()  
**Object.Equals(object)**  
Object.Instantiate(Object, Vector3, Quaternion)  
Object.Instantiate(Object, Vector3, Quaternion, Transform)  
Object.Instantiate(Object)  
Object.Instantiate(Object, Transform)  
**Object.Instantiate(Object, Transform, bool)**

Object.Instantiate<T>(T)  
Object.Instantiate<T>(T, Vector3, Quaternion)  
Object.Instantiate<T>(T, Vector3, Quaternion, Transform)  
Object.Instantiate<T>(T, Transform)  
[Object.Instantiate<T>\(T, Transform, bool\)](#)  
[Object.Destroy\(Object, float\)](#)  
Object.Destroy(Object)  
[Object.DestroyImmediate\(Object, bool\)](#)  
Object.DestroyImmediate(Object)  
[Object.FindObjectsOfType\(Type\)](#)  
Object.DontDestroyOnLoad(Object)  
[Object.DestroyObject\(Object, float\)](#)  
Object.DestroyObject(Object)  
[Object.FindSceneObjectsOfType\(Type\)](#)  
[Object.FindObjectsOfTypeIncludingAssets\(Type\)](#)  
Object.FindObjectsOfType<T>()  
Object.FindObjectOfType<T>()  
[Object.FindObjectsOfTypeAll\(Type\)](#)  
[Object.FindObjectOfType\(Type\)](#)  
Object.ToString()  
Object.name  
Object.hideFlags

Namespace: [Tessera](#)

Assembly: Tessera.dll

## Syntax

```
[AddComponentMenu("Tessera/Tessera Triangle Prism Tile")]
public class TesseraTrianglePrismTile : TesseraTileBase, ISerializationCallbackReceiver
```

## Constructors

[TesseraTrianglePrismTile\(\)](#)

Declaration

```
public TesseraTrianglePrismTile()
```

## Properties

[SubFaceInfo](#)

Declaration

```
public override ISubFaceInfo SubFaceInfo { get; }
```

Property Value

TYPE	DESCRIPTION
<a href="#">ISubFaceInfo</a>	

Overrides

[TesseraTileBase.SubFaceInfo](#)

[SylvesCellGrid](#)

Declaration

```
public override IGrid SylvesCellGrid { get; }
```

Property Value

TYPE	DESCRIPTION
IGrid	

Overrides

[TesseraTileBase.SylvesCellGrid](#)

**SylvesCellType**

Declaration

```
public override ICellType SylvesCellType { get; }
```

Property Value

TYPE	DESCRIPTION
ICellType	

Overrides

[TesseraTileBase.SylvesCellType](#)

**Methods**

**GetBounds()**

Declaration

```
public BoundsInt GetBounds()
```

Returns

TYPE	DESCRIPTION
BoundsInt	

**Implements**

UnityEngine.ISerializationCallbackReceiver

# Class TesseraUncertainty

If this is set on the "uncertainty tile" used by TesseraGenerator/AnimatedGenerator, it will be populated with data about which tiles are actually possible.

Inheritance

[object](#)

[Object](#)

[Component](#)

[Behaviour](#)

[MonoBehaviour](#)

[TesseraUncertainty](#)

Inherited Members

[MonoBehaviour.IsInvoking\(\)](#)

[MonoBehaviour.CancelInvoke\(\)](#)

[MonoBehaviour.Invoke\(string, float\)](#)

[MonoBehaviour.InvokeRepeating\(string, float, float\)](#)

[MonoBehaviour.CancelInvoke\(string\)](#)

[MonoBehaviour.IsInvoking\(string\)](#)

[MonoBehaviour.StartCoroutine\(string\)](#)

[MonoBehaviour.StartCoroutine\(string, object\)](#)

[MonoBehaviour.StartCoroutine\(IEnumerator\)](#)

[MonoBehaviour.StartCoroutine\\_Auto\(IEnumerator\)](#)

[MonoBehaviour.StopCoroutine\(IEnumerator\)](#)

[MonoBehaviour.StopCoroutine\(Coroutine\)](#)

[MonoBehaviour.StopCoroutine\(string\)](#)

[MonoBehaviour.StopAllCoroutines\(\)](#)

[MonoBehaviour.print\(object\)](#)

[MonoBehaviour.useGUILayout](#)

[MonoBehaviour.runInEditMode](#)

[Behaviour.enabled](#)

[Behaviour.isActiveAndEnabled](#)

[Component.GetComponent\(Type\)](#)

[Component.GetComponent<T>\(\)](#)

[Component.TryGetComponent\(Type, out Component\)](#)

[Component.TryGetComponent<T>\(out T\)](#)

[Component.GetComponent\(string\)](#)

[Component.GetComponentInChildren\(Type, bool\)](#)

[Component.GetComponentInChildren\(Type\)](#)

[Component.GetComponentInChildren<T>\(bool\)](#)

[Component.GetComponentInChildren<T>\(\)](#)

[Component.GetComponentsInChildren\(Type, bool\)](#)

[Component.GetComponentsInChildren\(Type\)](#)

[Component.GetComponentsInChildren<T>\(bool\)](#)

[Component.GetComponentsInChildren<T>\(bool, List<T>\)](#)

[Component.GetComponentsInChildren<T>\(\)](#)

[Component.GetComponentsInChildren<T>\(List<T>\)](#)

[Component.GetComponentInParent\(Type\)](#)

[Component.GetComponentInParent<T>\(\)](#)

[Component.GetComponentsInParent\(Type, bool\)](#)

[Component.GetComponentsInParent\(Type\)](#)

[Component.GetComponentsInParent<T>\(bool\)](#)

```
Component.GetComponentsInParent<T>(bool, List<T>)
Component.GetComponentsInParent<T>()
Component.GetComponents(Type)
Component.GetComponents(Type, List<Component>)
Component.GetComponents<T>(List<T>)
Component.GetComponents<T>()
Component.CompareTag(string)
Component.SendMessageUpwards(string, object, SendMessageOptions)
Component.SendMessageUpwards(string, object)
Component.SendMessageUpwards(string)
Component.SendMessageUpwards(string, SendMessageOptions)
Component.SendMessage(string, object)
Component.SendMessage(string)
Component.SendMessage(string, object, SendMessageOptions)
Component.SendMessage(string, SendMessageOptions)
Component.BroadcastMessage(string, object, SendMessageOptions)
Component.BroadcastMessage(string, object)
Component.BroadcastMessage(string)
Component.BroadcastMessage(string, SendMessageOptions)
Component.transform
Component.gameObject
Component.tag
Object.GetInstanceID()
Object.GetHashCode()
Object.Equals(object)
Object.Instantiate(Object, Vector3, Quaternion)
Object.Instantiate(Object, Vector3, Quaternion, Transform)
Object.Instantiate(Object)
Object.Instantiate(Object, Transform)
Object.Instantiate(Object, Transform, bool)
Object.Instantiate<T>(T)
Object.Instantiate<T>(T, Vector3, Quaternion)
Object.Instantiate<T>(T, Vector3, Quaternion, Transform)
Object.Instantiate<T>(T, Transform)
Object.Instantiate<T>(T, Transform, bool)
Object.Destroy(Object, float)
Object.Destroy(Object)
Object.DestroyImmediate(Object, bool)
Object.DestroyImmediate(Object)
Object.FindObjectsOfType(Type)
Object.DontDestroyOnLoad(Object)
Object.DestroyObject(Object, float)
Object.DestroyObject(Object)
Object.FindSceneObjectsOfType(Type)
Object.FindObjectsOfTypeIncludingAssets(Type)
Object.FindObjectsOfType<T>()
ObjectFindObjectOfType<T>()
Object.FindObjectsOfTypeAll(Type)
Object.FindObjectOfType(Type)
Object.ToString()
Object.name
Object.hideFlags
```

Namespace: [Tessera](#)

Assembly: Tessera.dll

## Syntax

```
public class TesseraUncertainty : MonoBehaviour
```

## Properties

### modelTiles

Declaration

```
public ISet<ModelTile> modelTiles { get; }
```

Property Value

TYPE	DESCRIPTION
<a href="#">ISet&lt;ModelTile&gt;</a>	

# Class TesseraVolume

Inheritance

[object](#)

[Object](#)

[Component](#)

[Behaviour](#)

[MonoBehaviour](#)

[TesseraBaseSceneConstraintComponent](#)

[TesseraVolume](#)

Inherited Members

[MonoBehaviour.IsInvoking\(\)](#)

[MonoBehaviour.CancelInvoke\(\)](#)

[MonoBehaviour.Invoke\(string, float\)](#)

[MonoBehaviour.InvokeRepeating\(string, float, float\)](#)

[MonoBehaviour.CancelInvoke\(string\)](#)

[MonoBehaviour.IsInvoking\(string\)](#)

[MonoBehaviour.StartCoroutine\(string\)](#)

[MonoBehaviour.StartCoroutine\(string, object\)](#)

[MonoBehaviour.StartCoroutine\(IEnumerator\)](#)

[MonoBehaviour.StartCoroutine\\_Auto\(IEnumerator\)](#)

[MonoBehaviour.StopCoroutine\(IEnumerator\)](#)

[MonoBehaviour.StopCoroutine\(Coroutine\)](#)

[MonoBehaviour.StopCoroutine\(string\)](#)

[MonoBehaviour.StopAllCoroutines\(\)](#)

[MonoBehaviour.print\(object\)](#)

[MonoBehaviour.useGUILayout](#)

[MonoBehaviour.runInEditMode](#)

[Behaviour.enabled](#)

[Behaviour.isActiveAndEnabled](#)

[Component.GetComponent\(Type\)](#)

[Component.GetComponent<T>\(\)](#)

[Component.TryGetComponent\(Type, out Component\)](#)

[Component.TryGetComponent<T>\(out T\)](#)

[Component.GetComponent\(string\)](#)

[Component.GetComponentInChildren\(Type, bool\)](#)

[Component.GetComponentInChildren\(Type\)](#)

[Component.GetComponentInChildren<T>\(bool\)](#)

[Component.GetComponentInChildren<T>\(\)](#)

[Component.GetComponentsInChildren\(Type, bool\)](#)

[Component.GetComponentsInChildren\(Type\)](#)

[Component.GetComponentsInChildren<T>\(bool\)](#)

[Component.GetComponentsInChildren<T>\(bool, List<T>\)](#)

[Component.GetComponentsInChildren<T>\(\)](#)

[Component.GetComponentsInChildren<T>\(List<T>\)](#)

[Component.GetComponentInParent\(Type\)](#)

[Component.GetComponentInParent<T>\(\)](#)

[Component.GetComponentsInParent\(Type, bool\)](#)

[Component.GetComponentsInParent\(Type\)](#)

[Component.GetComponentsInParent<T>\(bool\)](#)

[Component.GetComponentsInParent<T>\(bool, List<T>\)](#)

Component.GetComponentsInParent<T>()  
Component.GetComponents(Type)  
Component.GetComponents(Type, List<Component>)  
Component.GetComponents<T>(List<T>)  
Component.GetComponents<T>()  
Component.CompareTag(string)  
Component.SendMessageUpwards(string, object, SendMessageOptions)  
Component.SendMessageUpwards(string, object)  
Component.SendMessageUpwards(string)  
Component.SendMessageUpwards(string, SendMessageOptions)  
Component.SendMessage(string, object)  
Component.SendMessage(string)  
Component.SendMessage(string, object, SendMessageOptions)  
Component.SendMessage(string, SendMessageOptions)  
Component.BroadcastMessage(string, object, SendMessageOptions)  
Component.BroadcastMessage(string, object)  
Component.BroadcastMessage(string)  
Component.BroadcastMessage(string, SendMessageOptions)  
Component.transform  
Component.gameObject  
Component.tag  
Object.GetInstanceID()  
Object.GetHashCode()  
**Object.Equals(object)**  
Object.Instantiate(Object, Vector3, Quaternion)  
Object.Instantiate(Object, Vector3, Quaternion, Transform)  
Object.Instantiate(Object)  
Object.Instantiate(Object, Transform)  
**Object.Instantiate(Object, Transform, bool)**  
Object.Instantiate<T>(T)  
Object.Instantiate<T>(T, Vector3, Quaternion)  
Object.Instantiate<T>(T, Vector3, Quaternion, Transform)  
Object.Instantiate<T>(T, Transform)  
**Object.Instantiate<T>(T, Transform, bool)**  
Object.Destroy(Object, float)  
Object.Destroy(Object)  
Object.DestroyImmediate(Object, bool)  
Object.DestroyImmediate(Object)  
**Object.FindObjectsOfType(Type)**  
Object.DontDestroyOnLoad(Object)  
Object.DestroyObject(Object, float)  
Object.DestroyObject(Object)  
**Object.FindSceneObjectsOfType(Type)**  
**Object.FindObjectsOfTypeIncludingAssets(Type)**  
Object.FindObjectsOfType<T>()  
Object.FindObjectOfType<T>()  
**Object.FindObjectsOfTypeAll(Type)**  
**Object.FindObjectOfType(Type)**  
Object.ToString()  
Object.name  
Object.hideFlags

## Syntax

```
[AddComponentMenu("Tessera/Tessera Volume", 21)]
public class TesseraVolume : TesseraBaseSceneConstraintComponent
```

## Fields

**generator**

No effect on behaviour, setting this improves the UI in the Unity inspector.

## Declaration

```
public TesseraGenerator generator
```

## Field Value

TYPE	DESCRIPTION
TesseraGenerator	

**invertArea**

If false, affect all cells inside the volume's colliders. If true, affect all cells outside.

## Declaration

```
public bool invertArea
```

## Field Value

TYPE	DESCRIPTION
bool	

**tiles**

The list of tiles to filter on.

## Declaration

```
public List<TesseraTileBase> tiles
```

## Field Value

TYPE	DESCRIPTION
List<TesseraTileBase>	

**volumeType**

Controls the behaviour of this volume

## Declaration

```
public VolumeType volumeType
```

## Field Value

TYPE	DESCRIPTION
VolumeType	

# Class TesseraVolumeFilter

Inheritance

[object](#)

TesseraVolumeFilter

Implements

[ITesseraInitialConstraint](#)

Namespace: [Tessera](#)

Assembly: Tessera.dll

Syntax

```
public class TesseraVolumeFilter : ITesseraInitialConstraint
```

Fields

volumeType

Declaration

```
public VolumeType volumeType
```

Field Value

TYPE	DESCRIPTION
<a href="#">VolumeType</a>	

Properties

Name

Declaration

```
public string Name { get; }
```

Property Value

TYPE	DESCRIPTION
<a href="#">string</a>	

Implements

[ITesseraInitialConstraint](#)

# Enum TesseraWfcAlgorithm

Namespace: [Tessera](#)

Assembly: Tessera.dll

## Syntax

```
public enum TesseraWfcAlgorithm
```

## Fields

NAME	DESCRIPTION
Ac3	
Ac4	
Default	
OneStep	

# Class TileEntry

Specifies a tile to be used by [TesseraGenerator](#)

Inheritance

[object](#)

[TileEntry](#)

Namespace: [Tessera](#)

Assembly: Tessera.dll

Syntax

```
[Serializable]
public class TileEntry
```

Fields

**tile**

The tile to use

Declaration

```
public TesseraTileBase tile
```

Field Value

TYPE	DESCRIPTION
<a href="#">TesseraTileBase</a>	

**weight**

The weight controls the relative probability of this tile being selected. I.e. tile with weight of 2.0 is twice common in the generation than a tile with weight 1.0.

Declaration

```
public float weight
```

Field Value

TYPE	DESCRIPTION
<a href="#">float</a>	

# Class TileList

Inheritance

[object](#)

[TileList](#)

Namespace: [Tessera](#)

Assembly: Tessera.dll

Syntax

```
[Serializable]
public class TileList
```

Fields

[tiles](#)

Declaration

```
public List<TesseraTileBase> tiles
```

Field Value

TYPE	DESCRIPTION
<a href="#">List&lt;TesseraTileBase&gt;</a>	

# Enum TileSelectionType

Namespace: [Tessera](#)

Assembly: Tessera.dll

## Syntax

```
public enum TileSelectionType
```

## Fields

NAME	DESCRIPTION
Ordered	
Random	

# Class TrianglePrismDirExtensions

## NOTE

This class is available only in Tessera Pro

Inheritance

[object](#)

[TrianglePrismDirExtensions](#)

Namespace: [Tessera](#)

Assembly: Tessera.dll

Syntax

```
public static class TrianglePrismDirExtensions
```

## Methods

[Forward\(SylvesTrianglePrismDir\)](#)

Declaration

```
public static Vector3 Forward(this SylvesTrianglePrismDir dir)
```

Parameters

TYPE	NAME	DESCRIPTION
<a href="#">SylvesTrianglePrismDir</a>	dir	

Returns

TYPE	DESCRIPTION
<a href="#">Vector3</a>	

[GetSide\(SylvesTrianglePrismDir\)](#)

Declaration

```
public static int GetSide(this SylvesTrianglePrismDir dir)
```

Parameters

TYPE	NAME	DESCRIPTION
<a href="#">SylvesTrianglePrismDir</a>	dir	

Returns

TYPE	DESCRIPTION
<a href="#">int</a>	

[IsUpDown\(SylvesTrianglePrismDir\)](#)

Declaration

```
public static bool IsUpDown(this SylvesTrianglePrismDir dir)
```

## Parameters

TYPE	NAME	DESCRIPTION
SylvesTrianglePrismDir	dir	

## Returns

TYPE	DESCRIPTION
bool	

**IsValid(SylvesTrianglePrismDir, bool)**

## Declaration

```
public static bool IsValid(this SylvesTrianglePrismDir dir, bool pointsUp)
```

## Parameters

TYPE	NAME	DESCRIPTION
SylvesTrianglePrismDir	dir	
bool	pointsUp	

## Returns

TYPE	DESCRIPTION
bool	

**IsValid(SylvesTrianglePrismDir, Vector3Int)**

## Declaration

```
public static bool IsValid(this SylvesTrianglePrismDir dir, Vector3Int offset)
```

## Parameters

TYPE	NAME	DESCRIPTION
SylvesTrianglePrismDir	dir	
Vector3Int	offset	

## Returns

TYPE	DESCRIPTION
bool	

**SylvesOffsetDelta(SylvesTrianglePrismDir)**

## Declaration

```
public static Vector3Int SylvesOffsetDelta(this SylvesTrianglePrismDir dir)
```

## Parameters

<b>TYPE</b>	<b>NAME</b>	<b>DESCRIPTION</b>
SylvesTrianglePrismDir	dir	

Returns

<b>TYPE</b>	<b>DESCRIPTION</b>
Vector3Int	

## Up(SylvesTrianglePrismDir)

Declaration

```
public static Vector3 Up(this SylvesTrianglePrismDir dir)
```

Parameters

<b>TYPE</b>	<b>NAME</b>	<b>DESCRIPTION</b>
SylvesTrianglePrismDir	dir	

Returns

<b>TYPE</b>	<b>DESCRIPTION</b>
Vector3	

# Enum TrianglePrismFaceDir

## NOTE

This class is available only in Tessera Pro

Namespace: [Tessera](#)

Assembly: Tessera.dll

## Syntax

```
public enum TrianglePrismFaceDir
```

## Fields

NAME	DESCRIPTION
Back	
BackLeft	
BackRight	
Down	
Forward	
ForwardLeft	
ForwardRight	
Up	

## Extension Methods

[TrianglePrismFaceDirExtensions.Forward\(TrianglePrismFaceDir\)](#)  
[TrianglePrismFaceDirExtensions.GetSide\(TrianglePrismFaceDir\)](#)  
[TrianglePrismFaceDirExtensions.IsUpDown\(TrianglePrismFaceDir\)](#)  
[TrianglePrismFaceDirExtensions.IsValid\(TrianglePrismFaceDir, bool\)](#)  
[TrianglePrismFaceDirExtensions.IsValid\(TrianglePrismFaceDir, Vector3Int\)](#)  
[TrianglePrismFaceDirExtensions.OffsetDelta\(TrianglePrismFaceDir\)](#)  
[TrianglePrismFaceDirExtensions.Up\(TrianglePrismFaceDir\)](#)

# Class TrianglePrismFaceDirExtensions

## NOTE

This class is available only in Tessera Pro

## Inheritance

[object](#)

[TrianglePrismFaceDirExtensions](#)

Namespace: [Tessera](#)

Assembly: Tessera.dll

## Syntax

```
public static class TrianglePrismFaceDirExtensions
```

## Methods

[Forward\(TrianglePrismFaceDir\)](#)

### Declaration

```
public static Vector3 Forward(this TrianglePrismFaceDir dir)
```

### Parameters

TYPE	NAME	DESCRIPTION
<a href="#">TrianglePrismFaceDir</a>	dir	

### Returns

TYPE	DESCRIPTION
<a href="#">Vector3</a>	

[GetSide\(TrianglePrismFaceDir\)](#)

### Declaration

```
public static int GetSide(this TrianglePrismFaceDir dir)
```

### Parameters

TYPE	NAME	DESCRIPTION
<a href="#">TrianglePrismFaceDir</a>	dir	

### Returns

TYPE	DESCRIPTION
<a href="#">int</a>	

[IsUpDown\(TrianglePrismFaceDir\)](#)

### Declaration

```
public static bool IsUpDown(this TrianglePrismFaceDir dir)
```

## Parameters

TYPE	NAME	DESCRIPTION
TrianglePrismFaceDir	dir	

## Returns

TYPE	DESCRIPTION
bool	

IsValid(TrianglePrismFaceDir, bool)

## Declaration

```
public static bool IsValid(this TrianglePrismFaceDir dir, bool pointsUp)
```

## Parameters

TYPE	NAME	DESCRIPTION
TrianglePrismFaceDir	dir	
bool	pointsUp	

## Returns

TYPE	DESCRIPTION
bool	

IsValid(TrianglePrismFaceDir, Vector3Int)

## Declaration

```
public static bool IsValid(this TrianglePrismFaceDir dir, Vector3Int offset)
```

## Parameters

TYPE	NAME	DESCRIPTION
TrianglePrismFaceDir	dir	
Vector3Int	offset	

## Returns

TYPE	DESCRIPTION
bool	

OffsetDelta(TrianglePrismFaceDir)

## Declaration

```
public static Vector3Int OffsetDelta(this TrianglePrismFaceDir dir)
```

## Parameters

TYPE	NAME	DESCRIPTION
TrianglePrismFaceDir	dir	

Returns

TYPE	DESCRIPTION
Vector3Int	

## Up(TrianglePrismFaceDir)

Declaration

```
public static Vector3 Up(this TrianglePrismFaceDir dir)
```

Parameters

TYPE	NAME	DESCRIPTION
TrianglePrismFaceDir	dir	

Returns

TYPE	DESCRIPTION
Vector3	

# Class TrianglePrismGeometryUtils

## NOTE

This class is available only in Tessera Pro

## Inheritance

[object](#)

TrianglePrismGeometryUtils

Namespace: [Tessera](#)

Assembly: Tessera.dll

## Syntax

```
public static class TrianglePrismGeometryUtils
```

## Methods

[CoordRotate\(TriangleRotation, Vector3Int\)](#)

### Declaration

```
public static Vector3Int CoordRotate(TriangleRotation rotation, Vector3Int coords)
```

### Parameters

TYPE	NAME	DESCRIPTION
<a href="#">TriangleRotation</a>	rotation	
<a href="#">Vector3Int</a>	coords	

### Returns

TYPE	DESCRIPTION
<a href="#">Vector3Int</a>	

[FindCell\(Vector3, Vector3, Vector3, out Vector3Int\)](#)

### Declaration

```
public static bool FindCell(Vector3 origin, Vector3 cellSize, Vector3 position, out Vector3Int cell)
```

### Parameters

TYPE	NAME	DESCRIPTION
<a href="#">Vector3</a>	origin	
<a href="#">Vector3</a>	cellSize	
<a href="#">Vector3</a>	position	
<a href="#">Vector3Int</a>	cell	

### Returns

TYPE	DESCRIPTION
bool	

### FromSide(int)

Declaration

```
public static TrianglePrismFaceDir FromSide(int side)
```

Parameters

TYPE	NAME	DESCRIPTION
int	side	

Returns

TYPE	DESCRIPTION
TrianglePrismFaceDir	

### FromTriCoords(Vector3Int, int)

Declaration

```
public static Vector3Int? FromTriCoords(Vector3Int coords, int y = 0)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	coords	
int	y	

Returns

TYPE	DESCRIPTION
Vector3Int?	

### GetCellCenter(Vector3Int, Vector3, Vector3)

Declaration

```
public static Vector3 GetCellCenter(Vector3Int cell, Vector3 origin, Vector3 cellSize)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	cell	
Vector3	origin	
Vector3	cellSize	

Returns

TYPE	DESCRIPTION
Vector3	

## Pack(Vector2Int, bool, int)

### Declaration

```
public static Vector3Int Pack(Vector2Int tri, bool pointsUp, int y)
```

### Parameters

TYPE	NAME	DESCRIPTION
Vector2Int	tri	
bool	pointsUp	
int	y	

### Returns

TYPE	DESCRIPTION
Vector3Int	

## PointsUp(Vector3Int)

### Declaration

```
public static bool PointsUp(Vector3Int cell)
```

### Parameters

TYPE	NAME	DESCRIPTION
Vector3Int	cell	

### Returns

TYPE	DESCRIPTION
bool	

## Standardize(Vector2)

### Declaration

```
public static Vector2 Standardize(Vector2 p)
```

### Parameters

TYPE	NAME	DESCRIPTION
Vector2	p	

### Returns

<b>TYPE</b>	<b>DESCRIPTION</b>
Vector2	

### ToTriCoords(Vector3Int)

Declaration

```
public static Vector3Int ToTriCoords(Vector3Int cell)
```

Parameters

<b>TYPE</b>	<b>NAME</b>	<b>DESCRIPTION</b>
Vector3Int	cell	

Returns

<b>TYPE</b>	<b>DESCRIPTION</b>
Vector3Int	

### Unpack(Vector3Int)

Declaration

```
public static (Vector2Int, bool, int) Unpack(Vector3Int cell)
```

Parameters

<b>TYPE</b>	<b>NAME</b>	<b>DESCRIPTION</b>
Vector3Int	cell	

Returns

<b>TYPE</b>	<b>DESCRIPTION</b>
(Vector2Int, bool, int)	

### Unstandardize(Vector2)

Declaration

```
public static Vector2 Unstandardize(Vector2 p)
```

Parameters

<b>TYPE</b>	<b>NAME</b>	<b>DESCRIPTION</b>
Vector2	p	

Returns

<b>TYPE</b>	<b>DESCRIPTION</b>
Vector2	

# Struct TriangleRotation

Represents rotations / reflections of a hexagon

## NOTE

This class is available only in Tessera Pro

Inherited Members

### ValueType.ToString()

Namespace: [Tessera](#)

Assembly: Tessera.dll

Syntax

```
public struct TriangleRotation
```

## Properties

### All

Declaration

```
public static TriangleRotation[] All { get; }
```

Property Value

TYPE	DESCRIPTION
<a href="#">TriangleRotation[]</a>	

### Identity

Declaration

```
public static TriangleRotation Identity { get; }
```

Property Value

TYPE	DESCRIPTION
<a href="#">TriangleRotation</a>	

### IsReflection

Declaration

```
public bool IsReflection { get; }
```

Property Value

TYPE	DESCRIPTION
<a href="#">bool</a>	

### ReflectX

Declaration

```
public static TriangleRotation ReflectX { get; }
```

## Property Value

TYPE	DESCRIPTION
TriangleRotation	

## ReflectY

### Declaration

```
public static TriangleRotation ReflectY { get; }
```

## Property Value

TYPE	DESCRIPTION
TriangleRotation	

## RotateCCW

### Declaration

```
public static TriangleRotation RotateCCW { get; }
```

## Property Value

TYPE	DESCRIPTION
TriangleRotation	

## RotateCW

### Declaration

```
public static TriangleRotation RotateCW { get; }
```

## Property Value

TYPE	DESCRIPTION
TriangleRotation	

## Rotation

### Declaration

```
public int Rotation { get; }
```

## Property Value

TYPE	DESCRIPTION
int	

## Methods

### Equals(object)

#### Declaration

```
public override bool Equals(object obj)
```

## Parameters

TYPE	NAME	DESCRIPTION
object	obj	

## Returns

TYPE	DESCRIPTION
bool	

## Overrides

[ValueType.Equals\(object\)](#)

**GetHashCode()**

## Declaration

```
public override int GetHashCode()
```

## Returns

TYPE	DESCRIPTION
int	

## Overrides

[ValueType.GetHashCode\(\)](#)

**Invert()**

## Declaration

```
public TriangleRotation Invert()
```

## Returns

TYPE	DESCRIPTION
TriangleRotation	

**RotateCCW60(int)**

## Declaration

```
public static TriangleRotation RotateCCW60(int i)
```

## Parameters

TYPE	NAME	DESCRIPTION
int	i	

## Returns

TYPE	DESCRIPTION
TriangleRotation	

## Operators

**operator ==(TriangleRotation, TriangleRotation)**

Declaration

```
public static bool operator ==(TriangleRotation a, TriangleRotation b)
```

Parameters

TYPE	NAME	DESCRIPTION
TriangleRotation	a	
TriangleRotation	b	

Returns

TYPE	DESCRIPTION
bool	

**implicit operator TriangleRotation(CellRotation)**

Declaration

```
public static implicit operator TriangleRotation(CellRotation r)
```

Parameters

TYPE	NAME	DESCRIPTION
CellRotation	r	

Returns

TYPE	DESCRIPTION
TriangleRotation	

**implicit operator CellRotation(TriangleRotation)**

Declaration

```
public static implicit operator CellRotation(TriangleRotation r)
```

Parameters

TYPE	NAME	DESCRIPTION
TriangleRotation	r	

Returns

TYPE	DESCRIPTION
CellRotation	

**operator !=(TriangleRotation, TriangleRotation)**

Declaration

```
public static bool operator !=(TriangleRotation a, TriangleRotation b)
```

## Parameters

TYPE	NAME	DESCRIPTION
TriangleRotation	a	
TriangleRotation	b	

## Returns

TYPE	DESCRIPTION
bool	

**operator \*(TriangleRotation, int)**

## Declaration

```
public static int operator *(TriangleRotation a, int side)
```

## Parameters

TYPE	NAME	DESCRIPTION
TriangleRotation	a	
int	side	

## Returns

TYPE	DESCRIPTION
int	

**operator \*(TriangleRotation, TrianglePrismFaceDir)**

## Declaration

```
public static TrianglePrismFaceDir operator *(TriangleRotation rotation, TrianglePrismFaceDir faceDir)
```

## Parameters

TYPE	NAME	DESCRIPTION
TriangleRotation	rotation	
TrianglePrismFaceDir	faceDir	

## Returns

TYPE	DESCRIPTION
TrianglePrismFaceDir	

**operator \*(TriangleRotation, TriangleRotation)**

## Declaration

```
public static TriangleRotation operator *(TriangleRotation a, TriangleRotation b)
```

## Parameters

TYPE	NAME	DESCRIPTION
TriangleRotation	a	
TriangleRotation	b	

## Returns

TYPE	DESCRIPTION
TriangleRotation	

# Class UnityEngineInterface

Inheritance

[object](#)

[UnityEngineInterface](#)

Implements

[IEngineInterface](#)

Namespace: [Tessera](#)

Assembly: Tessera.dll

Syntax

```
public class UnityEngineInterface : IEngineInterface
```

Properties

Instance

Declaration

```
public static UnityEngineInterface Instance { get; }
```

Property Value

TYPE	DESCRIPTION
<a href="#">UnityEngineInterface</a>	

Methods

[Destroy\(Object\)](#)

Declaration

```
public void Destroy(Object o)
```

Parameters

TYPE	NAME	DESCRIPTION
Object	o	

[Instantiate\(GameObject, Vector3, Quaternion, Transform\)](#)

Declaration

```
public GameObject Instantiate(GameObject gameObject, Vector3 position, Quaternion rotation, Transform parent)
```

Parameters

TYPE	NAME	DESCRIPTION
GameObject	gameObject	
Vector3	position	
Quaternion	rotation	
Transform	parent	

Returns

TYPE	DESCRIPTION
GameObject	

## RegisterCompleteObjectUndo(Object)

Declaration

```
public void RegisterCompleteObjectUndo(Object objectToUndo)
```

Parameters

TYPE	NAME	DESCRIPTION
Object	objectToUndo	

## RegisterCreatedObjectUndo(Object)

Declaration

```
public void RegisterCreatedObjectUndo(Object objectToUndo)
```

Parameters

TYPE	NAME	DESCRIPTION
Object	objectToUndo	

Implements

[IEngineInterface](#)

# Enum VolumeType

Namespace: [Tessera](#)

Assembly: Tessera.dll

## Syntax

```
public enum VolumeType
```

## Fields

NAME	DESCRIPTION
MaskOut	Removes the cells inside the volume from generation
TilesetExclude	Restricts the set of tiles inside the volume to not these tiles
TilesetFilter	Restricts the set of tiles inside the volume to just these tiles