

Documentation v1.1.0.1f

Detailed documentation is available online at <https://terratiler2d.tiemenkamp.com/>.

If you do not have access to a stable internet connection, this local documentation can guide you through the basics of TerraTiler2D. Most aspects of TerraTiler2D will supply you with tooltips to make learning easier, and most scripts are well commented to make them easier to understand.

If you have any questions that were left unanswered, feel free to ask for help in our [community](#), or contact me by mail at TerraTiler2D@gmail.com.

Table of contents

Terminology	-	P. 2
How to create a graph	-	P. 3/4
How to add nodes to a graph	-	P. 5-8
Opening the node creation menu	-	P. 5
Searching for nodes	-	P. 6
Using properties	-	P. 7/8
How to create Tiles	-	P. 9
How to run a graph	-	P. 10-12
Running a graph in the editor	-	P. 10
Saving the result of a graph	-	P. 11
Running a graph during runtime	-	P. 12
How to access the output of a graph	-	P. 13
How to create custom nodes	-	P. 14/15
Saving and loading of custom node fields	-	P. 15
Important classes	-	P. 16-19
GraphData	-	P. 16/17
GraphOutput	-	P. 18
GraphRunner	-	P. 19
WorldBuilder	-	P. 20

Terminology:

Graph: A network of connected elements that are processed in a specific order. Graphs are saved as, and loaded from [GraphData](#) objects.

Node: The main type of element in a graph. In TerraTiler2D, a node defines a function with input parameters and return values.

Some nodes have very intricate functions, while others only create a simple variable.

Port: A node can contain several ports, indicated by the colored circles on the node. These ports define the input parameters and return values of the node. Each port has a color that indicates the type of variable of the port.

A port can either be an input port, or an output port. Ports on the left side of a node are input ports, which pass variables into the node. Ports on the right side of a node are output ports, which return a variable.






Edge: An input port and an output port of the same type can be connected using an edge, by dragging and dropping from one port to the other. Connected output ports will pass their return value on to the input ports they are connected to.

Flow: The word 'flow' is used to describe the main order in which nodes are processed. In

TerraTiler2D the flow is indicated by ports with the  icon.

Each TerraTiler2D graph contains a Start node with a flow port. This is the starting point of the graph, and will always be the first node to be processed. After processing a node with a flow port, the graph continues on to the next node along the flow.

Special ports: Some ports have an icon around their colored circle. These icons indicate special behaviour.

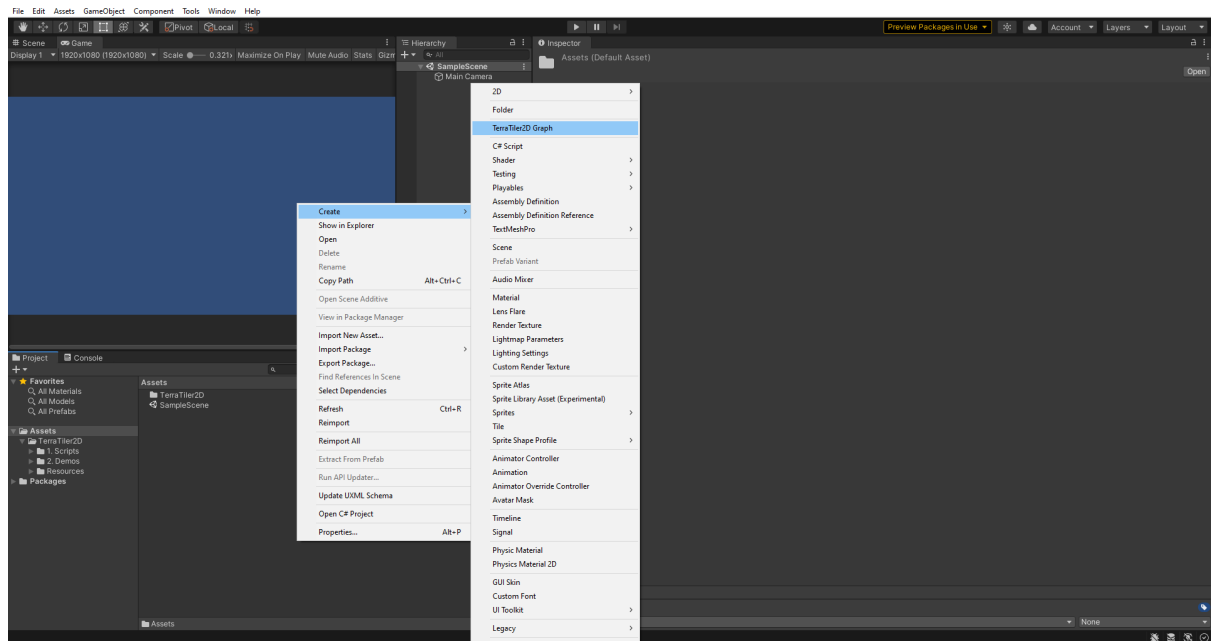
-  indicates the flow of the graph.
-  indicates that the port requires a connection for the node to work.
-  indicates that the port can be connected to more than one other port.
-  indicates that the port is a type of array.
-  indicates that the port is a type of mask.

How to create a graph

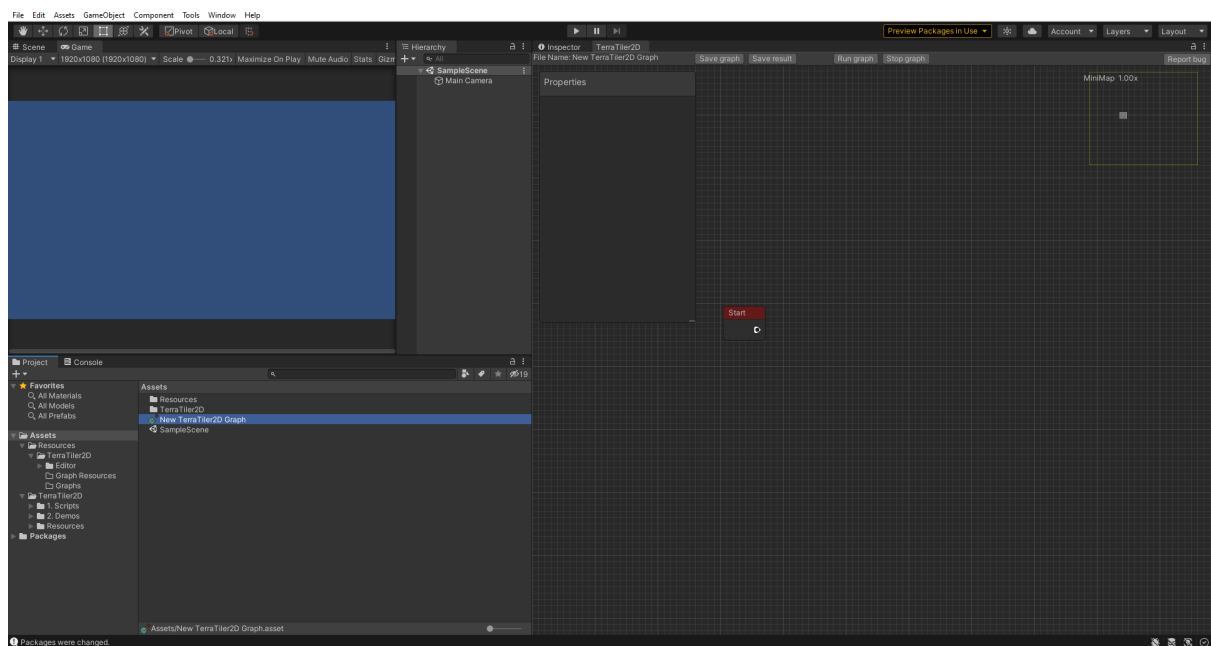
Creating a new graph is very easy.

You first need to access the Assets menu in Unity, either by right clicking in your Project window, or by clicking the 'Assets' button in the top left of Unity.

In the Assets menu, select 'Create -> TerraTiler2D Graph'. A [GraphData](#) object should appear in the current folder of your Project window.

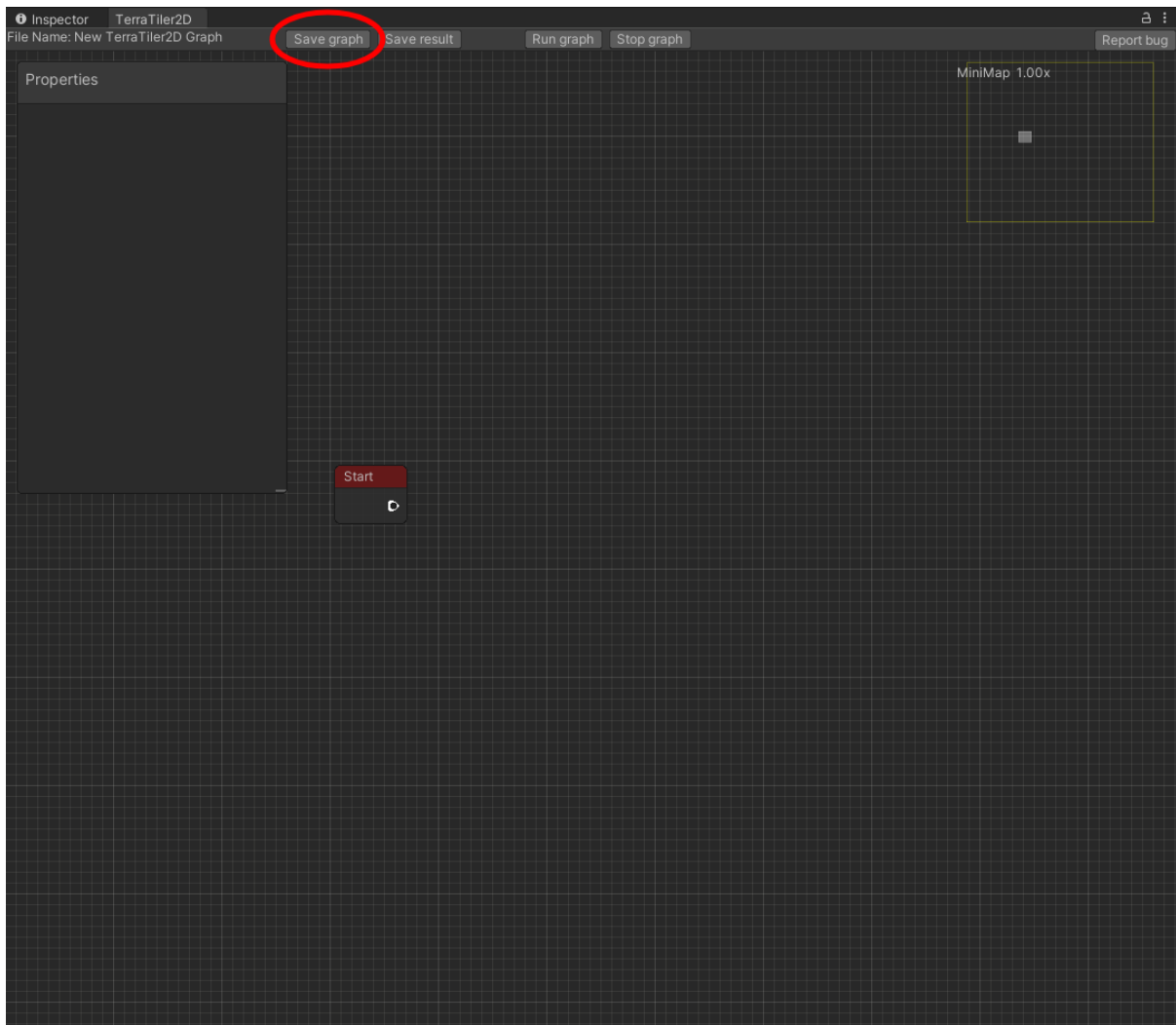


Double-click this GraphData object, and the TerraTiler2D window should open up and show the content of your newly created graph.



After making changes to your graph, make sure to click the 'Save graph' button at the top of the TerraTiler2D window. This will save the changes in your GraphData object.

To load a graph, simply double-click the GraphData object like you did before.




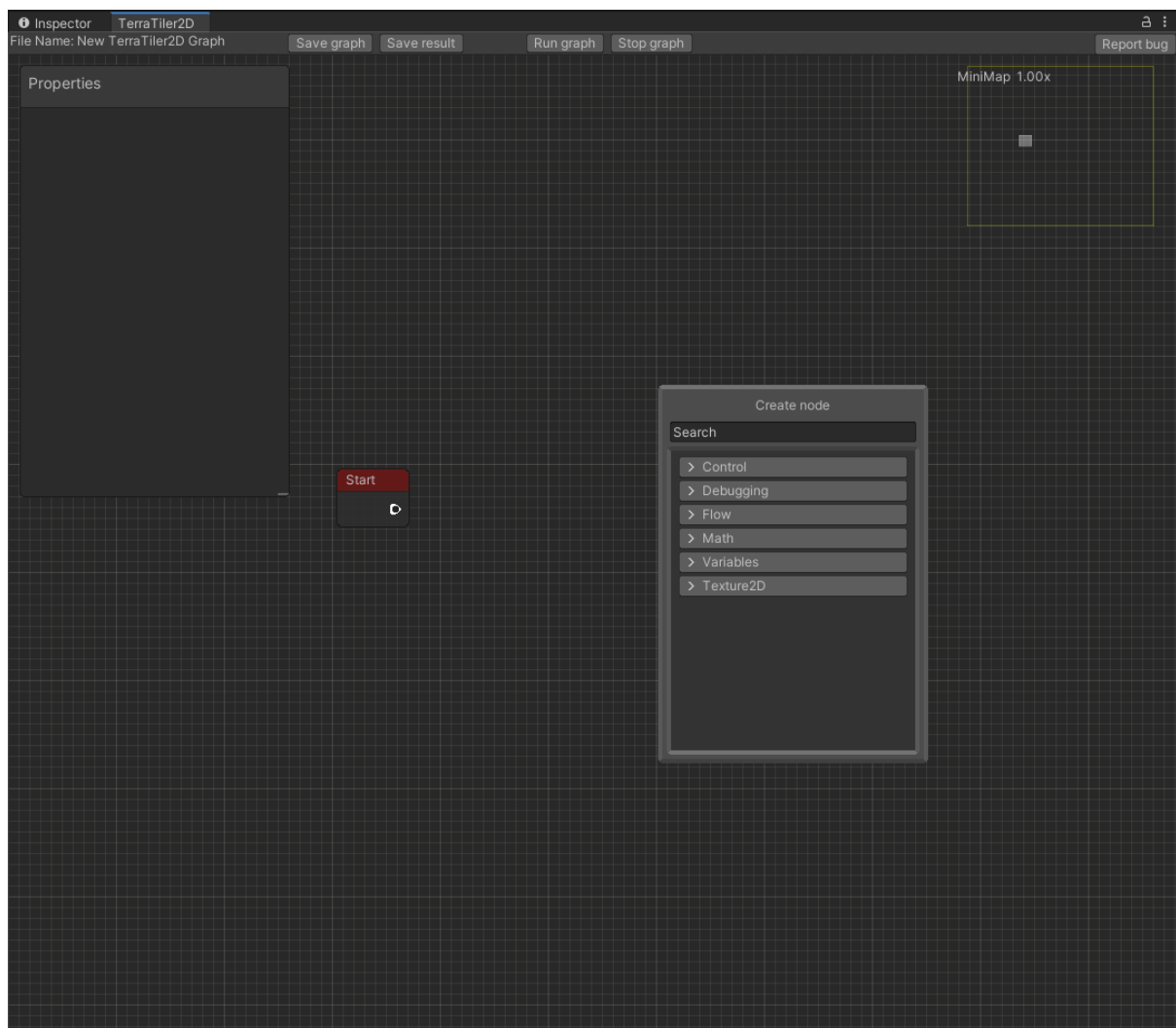
How to add nodes to a graph

Opening the node creation menu

In the TerraTiler2D window, right click anywhere on the grid background, and select 'Create Node'. A menu should open up. This is the node creation menu.

This menu contains all the nodes you are able to create, ordered by category. Simply press the node's menu entry to make it appear in your graph.

To get information on a node, hover over the  icon to see that node's tooltip. This tooltip consists of 3 components; the node's description, the node's input tooltip, and the node's output tooltip.



Searching for nodes

If you want to find a specific node, you can use the search bar. All the nodes with a name that contains your search will be listed in the menu.

Another way to find the node you are looking for is by dragging an edge from a port, and dropping it anywhere in the graph. This will open up the node creation menu, and automatically apply a filter to only list nodes that can be connected to that edge.

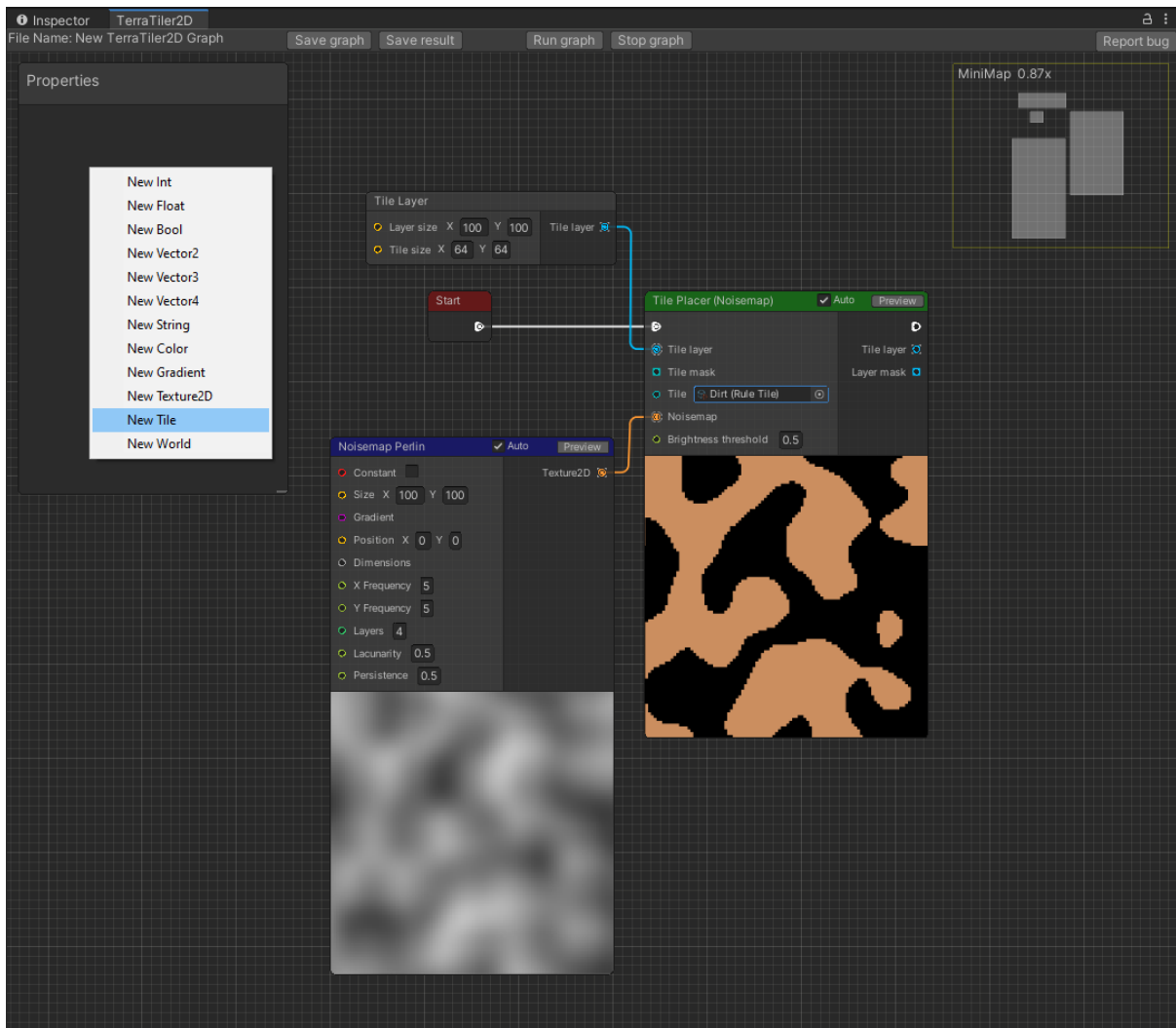
You can also manually apply filters to your search:

Filter	Functionality	Example
:name: or :title:	Only return nodes with a name that contains the search criteria. This is the default behaviour of the search bar, but if you wish to combine this default behaviour with other filters, you need to manually apply it using these tags.	":name: noisemap" Returns all nodes whose name contains "noisemap".
:description: or :des:	Only return nodes with a description that contains the search criteria.	":description: places tiles" Returns all nodes whose description contains "places" and "tiles".
:input: or :in:	Only return nodes with an input tooltip that contains the search criteria.	":input: texture2d bool vect" Returns all nodes whose input tooltip contains "texture2d", "bool", and "vect"
:output: or :out:	Only return nodes with an output tooltip that contains the search criteria.	":output: world :input: textu" Returns all nodes whose output tooltip contains "world", and input tooltip contains "textu".

Using properties

Besides creating nodes, you can also create properties. To do so, right click anywhere in the Properties window found in the top left corner of the TerraTiler2D window.

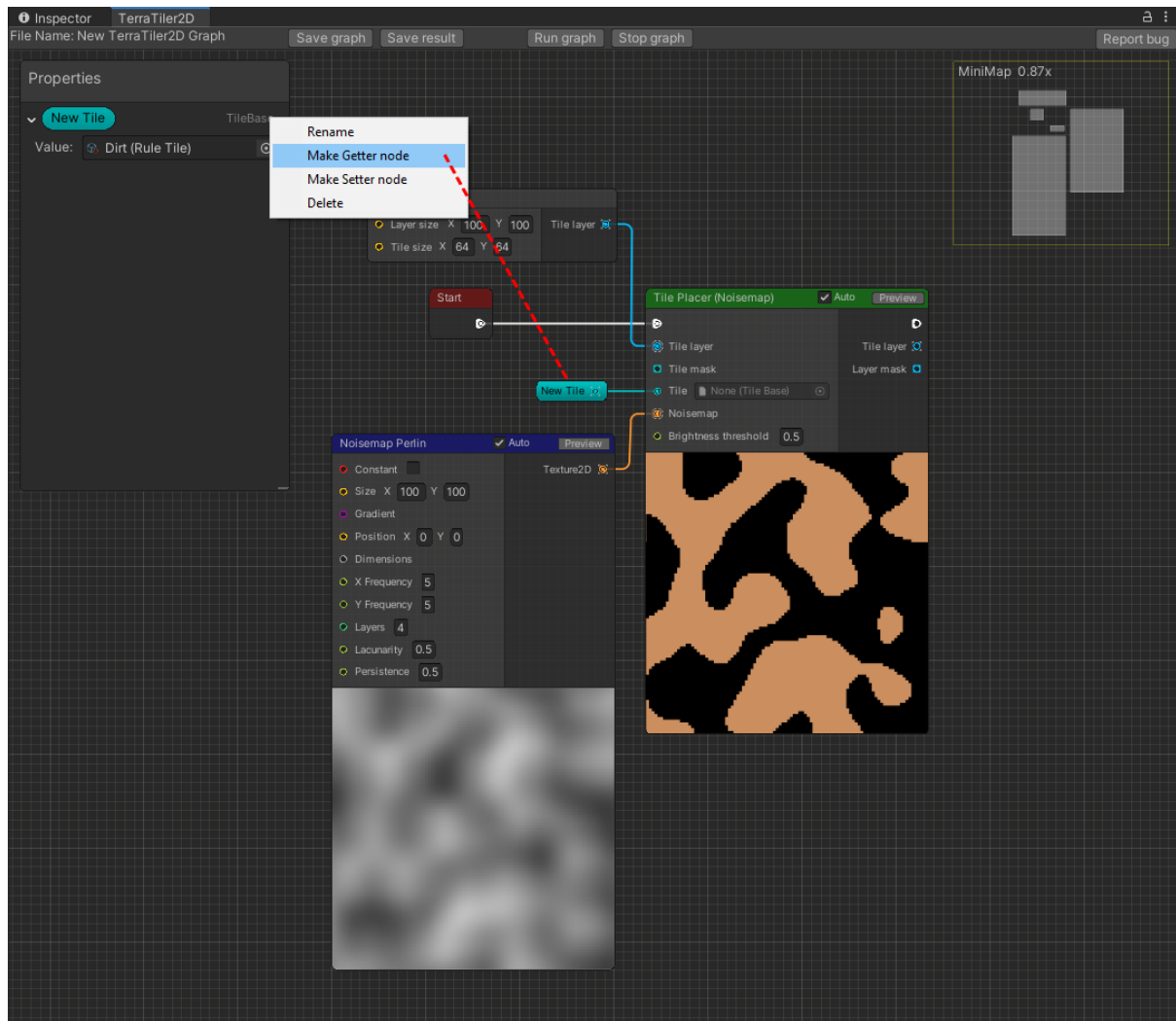
Properties act as the input and output of your graph. Both before and after running a graph, you can access the values of these properties. You can change property values manually using the variable fields in the Properties window, or you can do so through scripts (which will be explained in a later chapter).



Once you have created a property, you can right click that property to open up its contextual menu.

If you press 'Make Getter node', a small colored node should appear in your graph. This node will return the value of the associated property.

If you press 'Make Setter node', a bigger colored node should appear in your graph. This node can be used to change the value of a property during graph execution.



How to create Tiles

Creating a Tile is just as easy as creating a new Graph.

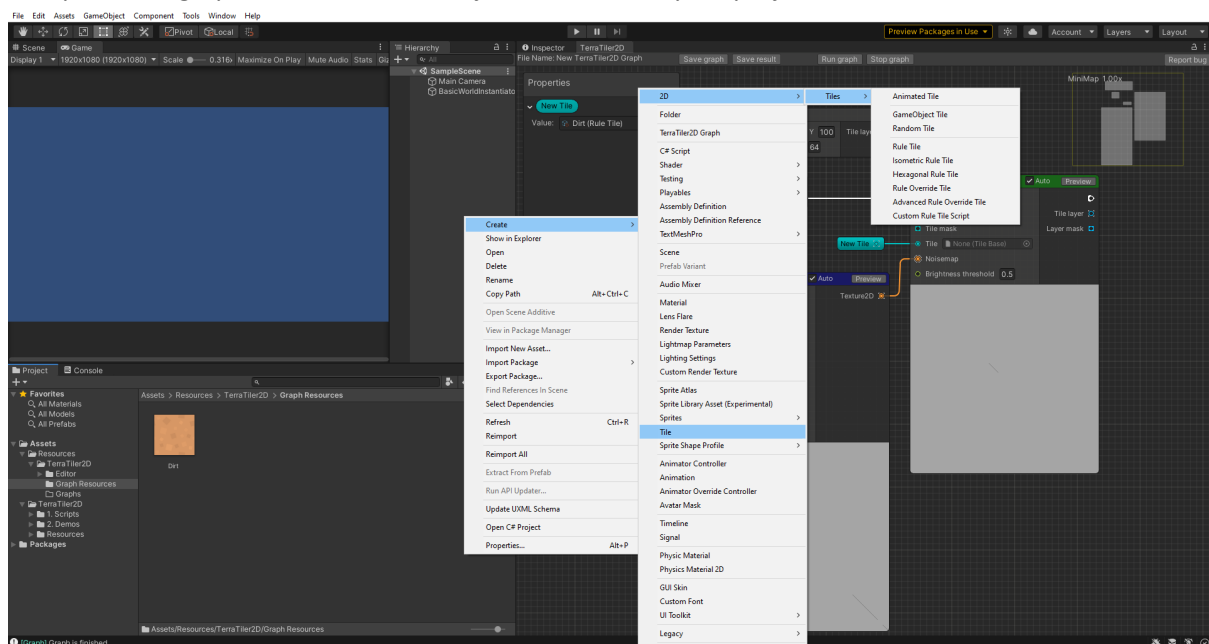
Just like you did when you created a Graph, access the Assets menu in Unity, either by right clicking in your Project window, or by clicking the 'Assets' button in the top left of Unity.

In the Assets menu, select 'Create -> Tile' to create a basic Tile, or select 'Create -> 2D -> Tiles' to see a list of Tiles with special behaviours. After selecting which type of Tile you want, a Tile object should appear in the current folder of your Project window.

These Tiles are part of the Unity 2D Tilemap Extras package. Below is a brief description of each type of Tile, but for more information you should check out their [documentation](#) or [GitHub repository](#).

- **Tile:** Base tile type that shows a Texture and has the option to detect collisions.
- **GameObjectTile:** Spawns a GameObject prefab, and then disables itself.
- **RandomTile:** Random Tiles are tiles which pseudo-randomly pick a sprite from a given list of sprites.
- **AnimatedTile:** Animated Tiles are tiles which run through and display a list of sprites in sequence.
- **RuleTile:** Generic visual tile for creating different tilesets like terrain, pipeline, random or animated tiles. Change their look based on neighbouring tiles.
- **Isometric Rule Tile:** A Rule Tile for use with Isometric Grids.
- **Hexagonal Rule Tile:** A Rule Tile for use with Hexagonal Grids. Enable Flat Top for Flat Top Hexagonal Grids and disable for Pointed Top Hexagonal Grids.
- **Rule Override Tile:** Rule Override Tiles are Tiles which can override a subset of Rules for a given Rule Tile to provide specialised behaviour while keeping most of the Rules originally set in the Rule Tile.

If you reference a Tile in one of your graphs, whether it is as a Property or in a Port field, you should place your Tile object in any Resources folder at 'Resources -> TerraTiler2D -> Graph Resources'. This is required for graphs to find the Tile object in a build of your project.

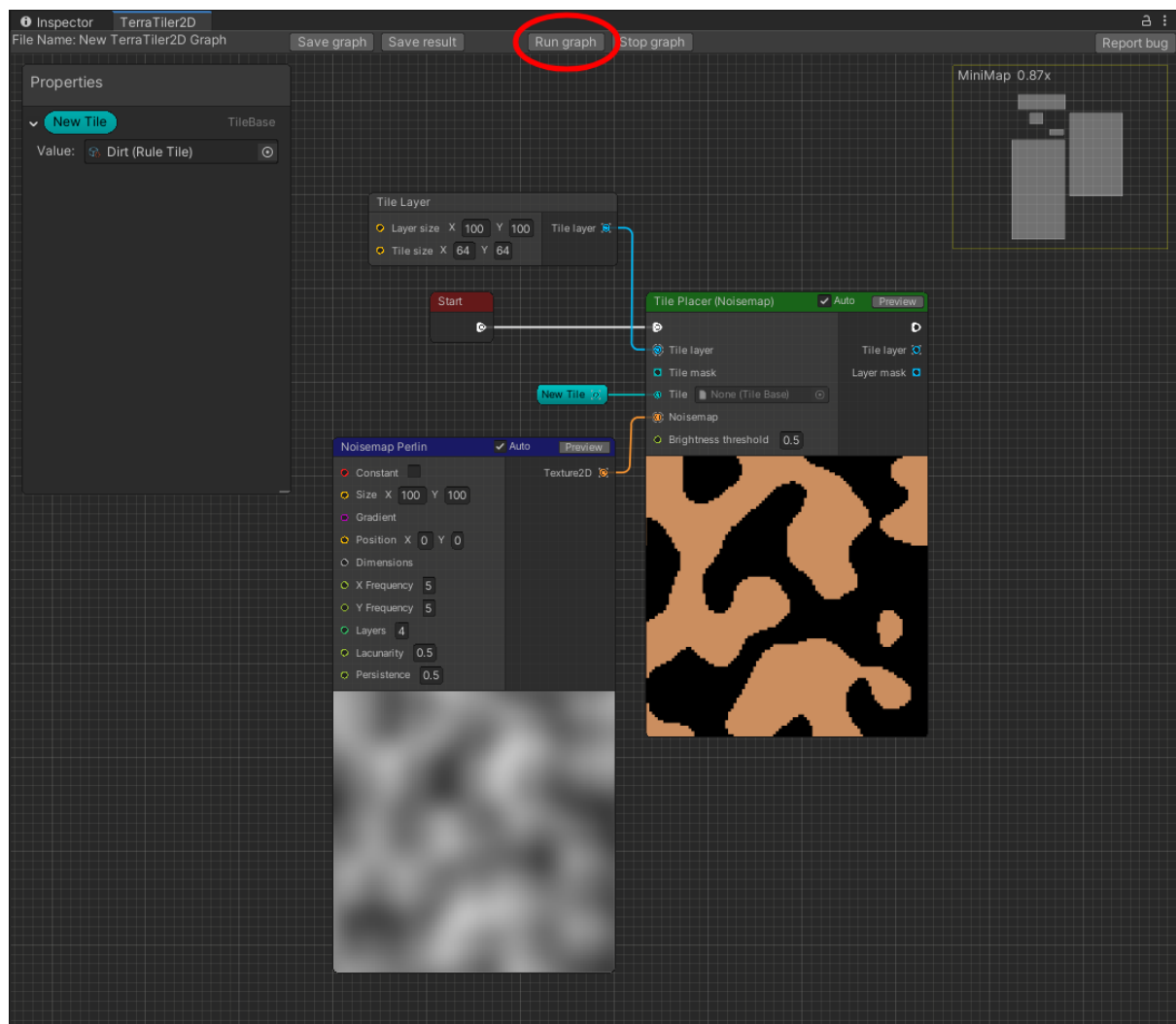


How to run a graph

Running a graph in the editor

Once you have connected some nodes to your flow, you may want to check what kind of results your graph will produce. To do this, you can press the 'Run graph' button in the toolbar at the top of the TerraTiler2D window. This will tell your graph to process all the nodes connected along the flow, starting at the Start node.

Nodes with a NodePreview extension, like Clamp Texture2D or Tile Placer (Noisemap), will update their previews to the newly generated results.



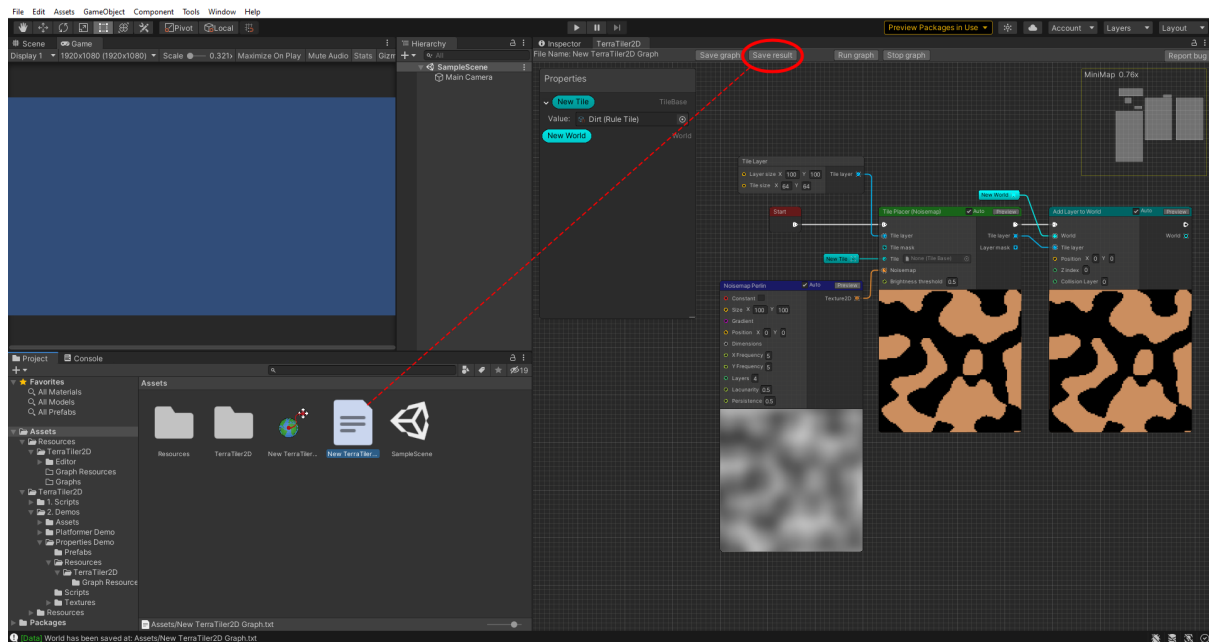
If your graph does not seem to run, check the Console window for any error messages. If this does not help, check if the forums can provide you with an answer, or contact me about your issue.

Saving the result of a graph

If your graph produced a result that you like, you can save that specific result by pressing the 'Save result' button in the toolbar at the top of the TerraTiler2D window.

This will try to get a World property with the exact name 'New World', and save the TileLayers added to that world as a .txt file with comma separated values. World properties with a different name will not work.

A .txt file should appear in the Project window, in the same folder as where you saved your graph (Note: The file will appear after some time, or after refreshing Unity by pressing CTRL+R.)



Running a graph during runtime

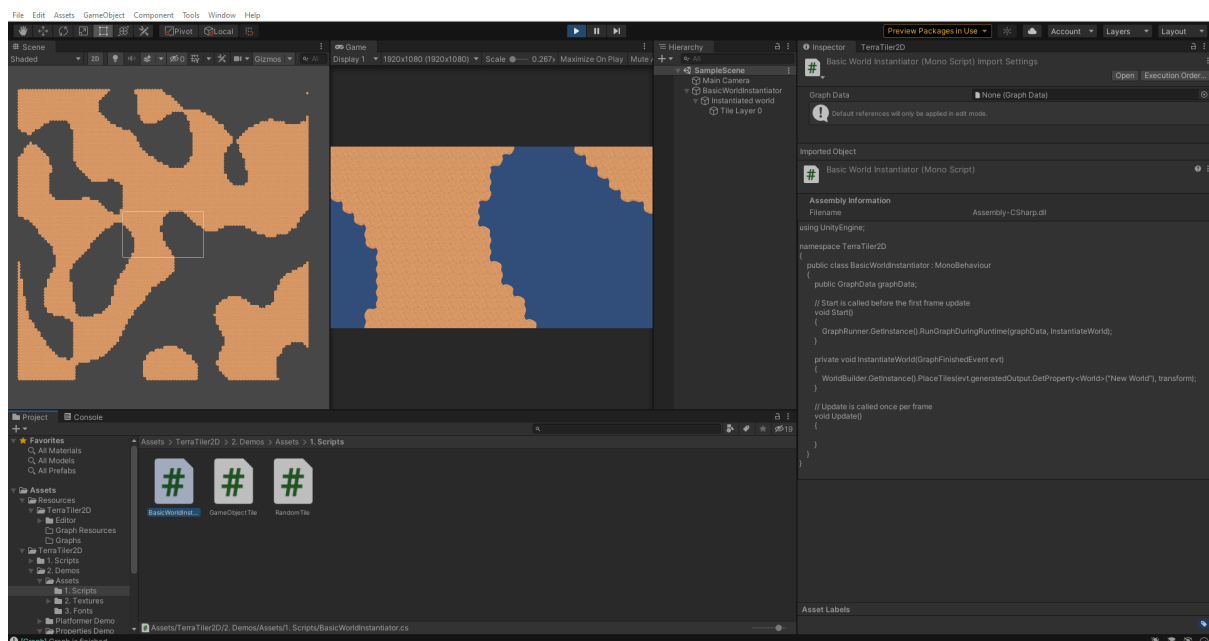
To run a graph during runtime, the [GraphRunner](#) singleton can be used. This singleton contains two methods, one to run the [GraphData](#) that is currently loaded in the TerraTiler2D window, and one to run a GraphData object without loading it in the TerraTiler2D window. These methods will return a [GraphOutput](#) object, containing all of the output properties of your graph, which you can then use for whatever you wish.

If your desired game world is a standard 2D tile-based world, you can use the BasicWorldInstantiator script included with the TerraTiler2D demos. Alternatively, you can use the 'PlaceTiles()' method of the [WorldBuilder](#) singleton to build the world in your scene during runtime.

First call 'GraphRunner.GetInstance().RunGraphDuringRuntime()' to get the GraphOutput of your graph. Retrieve the World object you want to instantiate, and pass it into 'WorldBuilder.GetInstance().PlaceTiles()' to place actual Tile objects in your scene.

The WorldBuilder singleton provides a simple way to create a world for people with little knowledge of tile-based instantiation. If you know how to work with .csv files, 2D arrays, and tile-based worlds, you may want to create your own world instantiation method.

Both of these singletons are part of the TerraTiler2D namespace. To access them from your scripts you either have to put 'using TerraTiler2D;' at the top of your script, or write 'TerraTiler2D.GraphRunner' instead of 'GraphRunner'.



How to access the output of a graph

Accessing the output of a graph is pretty straight forward. As explained earlier, properties act as the input and output of your graph.

The properties in your [GraphData](#) object act as the input. These can be accessed and changed in the properties window inside the TerraTiler2D window, or in a script using the `GetPropertyValue()` and `SetPropertyValue()` methods on your `GraphData` object. For more information on the Get and Set methods, take a look at the `GraphData` page near the bottom of this documentation file.

Upon running a graph using your `GraphData` object, a [GraphOutput](#) object is created. This `GraphOutput` object contains all of the same properties as your `GraphData` object, but these include any changes applied by your graph. If your graph added a `TileLayer` to World property "myWorld", the `GraphOutput` property "myWorld" will include that tile layer. Your `GraphData` object on the other hand, will not have that added tile layer.

The `GraphData` and `GraphOutput` classes are explained in more detail at the end of this documentation file.

It is advised to read `GraphData` carefully, as incorrect usage of this class may result in corrupted data. This is because you are setting values on a persistent data type, which may permanently overwrite data you did not want to lose. The `GraphData` page explains how to prevent this.

Check out the Properties demo included with TerraTiler2D for examples on getting and setting property values, and for examples on persistently saving and loading graph data instances.

How to create custom nodes

Creating custom nodes is possible, but the process has not been polished yet, and can therefore be somewhat confusing. To create a custom node, follow these steps:

Glob script
<p>In the Glob script, add a new entry to the NodeTypes enum for your node type, and give it a unique index.</p> <p>NOTE: Do not change any of the existing indexes, or you will corrupt all saved Graph Data objects that use that existing index, due to changed NodeType indices!</p>
<p>Give your node a name, by adding an entry to the DefaultNodeNames dictionary with your new NodeType.</p>
New script
<p>Create a new script with a class for your new node type, and make it inherit from Node, or any other class that inherits from Node.</p>
<p>Implement a constructor that uses ': base()' to pass all the required variables to the Node constructor.</p>
<p>Make sure to manually set the 'nodeType' variable of your new node in its constructor, or the wrong node will be created.</p> <p>You can also define your node's categorization in the node creation menu by assigning a value to 'searchMenuEntry' in the constructor. This requires you to define an array of strings, where each string defines a sub-group your node will be part of. Defining no searchMenuEntry will prevent your node from showing up in the node creation menu.</p>
<p>Define how your new node should look in the abstract Initialization methods. Be careful not to reference any of the 'GraphView.Node' variables/methods, as they are part of the UnityEditor namespace, which will not work in builds. Look at scripts of existing nodes to see how this can be done (W.I.P.).</p> <p>If you accidentally referenced a UnityEditor method or variable, it is not the end of the world. You will not notice any issues in the Unity Editor, but as soon as you try to build your project, a bunch of "null reference" errors will pop-up, as the UnityEditor namespace is unavailable outside of the Unity editor.</p> <p>To solve these errors, you may be able to make use of '#if (UNITY_EDITOR)' and '#endif'. Any code in between these two tags is excluded from builds, meaning the UnityEditor references are not included in your build. Do note that this code will now no longer throw an error, but also not run in your build.</p> <p>If you still need the code in your build, you need to find a work around. Feel free to ask for help on the forums, in our discord server, or contact me directly.</p>
GraphView script
<p>Lastly, you will need to update the CreateNode method in the GraphView script to include your new node type.</p>

Saving and loading of custom node fields

Most common types are supported by PortWithField, which will handle saving and loading of that port. If you added custom fields to your node, you may have to save and load the values for these fields yourself. If your custom nodes has such fields, follow these steps:

New NodeData class

Add a NodeData class for your node type at the top of your Node script. Make this NodeData class inherit from NodeData (or any other class that inherits from NodeData).

Add all the additional variables that should be saved for your node to the new NodeData class. Make these variables public, or otherwise accessible.

Add a [Serializable] tag to the new NodeData class.

Your Node class

In your Node class, override the virtual methods GetNodeData and LoadNodeData. In these methods, handle the saving and loading of data for your Node. Look at scripts of existing nodes, like Gradient_Node.cs, to see how this can be done.

GraphData script

Create a new NodeData list for your node type in the GraphData script.

Update the GetAllNodeData method to include your new NodeData list.

Update the AddNodeData method to include your new node type. Make sure to add the nodeData to your new list.

Important classes

GraphData

Inherits from ScriptableObject

Description

ScriptableObject that holds all the information for TerraTiler2D graphs. This is the object type used for saving and loading TerraTiler2D graphs from the Project window.

Use this class with the [GraphRunner](#) singleton to run graphs during runtime.

If you are having trouble with getting and setting properties, or saving and loading graph data, feel free to ask for help on the forums or in our Discord server, or check out the Properties Demo included with TerraTiler2D.

Methods

Name	Return type	Description
GetProperty<T>(string name)	T	<p>Get the value of an unprocessed graph property of type 'T' with name 'name'.</p> <p>Value types that are not serializable by default require you to use a serializable alternative. Some serializable alternatives can be found in the static 'Serializable' class.</p> <p>Currently supported unserializable types: Vector2, Vector3, Vector4, Color, Gradient, Texture2D, Tile, and World.</p> <p>Example: Use 'Serializable.Vector2' as type T instead of 'Vector2'.</p>
SetProperty<T>(string name, T value)	void	<p>Set the value of an unprocessed graph property of type 'T' with name 'name'. This is useful for changing the behaviour of a graph before running it.</p> <p>Changing properties in this way is not persistent throughout different play sessions. Use SaveSerializedData and LoadSerializedPropertyData to persistently save and load property values.</p> <p>Value types that are not serializable by default require you to use a serializable alternative. Some serializable alternatives can be found in the static 'Serializable' class.</p> <p>Currently supported unserializable types: Vector2, Vector3, Vector4, Color, Gradient, Texture2D, Tile, and World.</p> <p>Example: Use 'Serializable.Vector2' as type T instead of 'Vector2'.</p>

SaveSerializedData (string saveName = "")	void	<p>Save the current state of the GraphData onto the local machine with file name 'saveName.dat', including any changes done to property values. This is useful for creating persistent save states for your game.</p> <p>Example: Your player has unlocked an achievement, and by doing so has unlocked a new kind of biome that can spawn in your world. Set the value of boolean property 'UnlockedBiome' to true, and save this. The next time the player loads up the game, load the save file, and the property should be set to true automatically.</p> <p>An example of saving and loading can be seen in the Properties Demo.</p>
LoadSerializedPropertyData(string saveName = "")	Graph Data	<p>Load data from a save file from the local machine with file name 'saveName.dat' into this GraphData object. This method only overwrites the values of properties, and is able to merge most save files with different versions without issues.</p> <p>When loading data, it is good practice to not load the data directly into your GraphData ScriptableObject. Instead, use 'Instantiate(yourGraphData);' to create an instance of your GraphData, load the data into the instance, and use the instance in your scripts. This ensures that you don't accidentally overwrite existing data that you wanted to keep.</p>
LoadSerializedData (string saveName = "")	Graph Data	<p>Load data from a save file from the local machine with file name 'saveName.dat' into this GraphData object. This method overwrites all of the data, including properties, nodes, edges, and port values. It is advised to use LoadSerializedPropertyData instead of this method if you only need to load property values, as this method may cause unwanted behaviour when loading GraphData with a different graph Flow.</p> <p>When loading data, it is good practice to not load the data directly into your GraphData ScriptableObject. Instead, use 'Instantiate(yourGraphData);' to create an instance of your GraphData, load the data into the instance, and use the instance in your scripts. This ensures that you don't accidentally overwrite existing data that you wanted to keep.</p>
Other methods (AddNodeData, AddPropertyData, GetAllPortData, etc.)	/	<p>Most of the other methods on this class are not needed for normal use of TerraTiler2D. These other methods are used by singletons like GraphSaveManager for saving and loading graph data, and have little other uses.</p>

Graph Output

No inheritance

Description

The type of class you receive from running a graph. This class contains methods to retrieve the resulting output properties from graphs.

Methods

Name	Return type	Description
GetProperty<T>(string name)	T	Get the value of a processed graph property of type 'T' with name 'name'.
GetPropertyAsObject (string name)	object	Get the value of a processed graph property with name 'name' as an object.
GetAllPropertiesAsObject()	Dictionary <string, object>	Get the values of all processed graph properties.
GetAllPropertyNames()	List<string>	Get the names of all processed graph properties.

Graph Runner

Inherits from Singleton

Description

To run a graph during runtime, this GraphRunner singleton can be used. This singleton contains two methods, one to run the graph that is currently loaded in the TerraTiler2D editor window, and one to run a Graph object without showing it in the TerraTiler2D editor window. These methods will return a [Graph Output](#) object containing all of the processed properties of your graph, which you can then use for whatever you wish.

Methods

Name	Return type	Description
GetInstance()	GraphRunner	Get the singleton instance.
RunGraph(GraphView graphView, Action<GraphFinishedEvent> listener = null, GraphSeed graphSeed = GraphSeed.Random)	void	Run the graph that is currently loaded in the TerraTiler2D editor window (also known as a GraphView). After the graph is finished, the 'listener' will be invoked, and the GraphOutput that was produced will be passed into the listener method.
RunGraphDuringRuntime(GraphData graphData, Action<GraphFinishedEvent> listener, GraphSeed graphSeed = GraphSeed.Random)	void	Run a graph based on a GraphData object, which contains serialized data on properties, nodes, edges, and port values. This version of the RunGraph method will not visually show the graph, and is usable during runtime.
StopGraph(GraphView graphView = null)	void	Tries to stop the graph that is currently running in the target GraphView.
GetPreviousSeed()	int	Gets the seed that was used for the previous graph execution.

World Builder

Inherits from Singleton

Description

If your desired world is a standard 2D tile-based world, you can use the 'PlaceTiles()' method of this WorldBuilder singleton to build the world in your scene during runtime.

First pass your [GraphData](#) object into '[GraphRunner](#).GetInstance().RunGraphDuringRuntime()' to get a [GraphOutput](#) object. Get the World property you want to instantiate from the GraphOutput using `GetProperty<T>()`, and pass it into '`WorldBuilder.GetInstance().PlaceTiles()`' to place actual tile objects in your scene.

This WorldBuilder singleton provides a simple way to create a world for people with little knowledge of tile-based instantiation. If you know how to work with .csv files, 2D arrays, and tile-based worlds, you may want to create your own world instantiation method.

Methods

Name	Return type	Description
GetInstance()	GraphRunner	Get the singleton instance.
PlaceTiles(World world, Transform parent, bool destroyExistingWorlds = true)	GameObject	<p>Places a TileMap in the scene and fills it with Tile objects based on the data in the World object. Uses a basic tile placing algorithm, which should work for standard 2D worlds generated with TerraTiler2D. If your project requires a more advanced tile placing algorithm, you will have to write it yourself.</p> <p>If 'destroyExistingWorlds' is true, the World Builder will destroy all existing tiles before building a new world.</p>