WORLD POLITICAL MAP
GLOBE EDITION

+ Map Editor!

RELEASE NOTES

Index

Introduction

Thank you for purchasing!

World Political Map – Globe Edition is a commercial asset for Unity 5.1.1 and above that allows to:

- ✓ Visualize the frontiers of 241 countries, +4000 provinces and states and the location of +7000 most important cities in the world without connecting to the Internet (integrated cartography).
- ✓ Ability to integrate with online map tile systems.
- ✓ Colorize, texture and also highlight the regions of countries and provinces/states as mouse hovers them.
- ✓ Automatically draw country labels.
- ✓ Add markers and line animations to the globe.
- ✓ Define custom mount points and customize its location and tags with the editor.
- ✓ Fly to a chosen country, city, province or location. It will make the globe rotate until the destination is reached.
- ✓ Ease choose beteween different catalogs included based on quality/size for frontiers and cities. Filter number of cities by population and/or size.
- ✓ Lots of customization options: colors, labels, frontiers, provinces, cities, Earth (7 styles including scenic with clouds, shadows, day/light and nice glow effects compatible with mobile and another even more advanced including physically based atmosphere scattering)…
- ✓ Works on Android and iOS (all styles except for the atmosphere scattering style).
- ✓ And much more…

You can use this asset to represent or allow the user choose a location in your game/application, in mission briefings, reports, statistical or educational software, etc.

QuickStart and Demo Scene

1. Import the asset into your project or create an empty project.
2. Open any of the demo scenes included in Demo folder.
3. Run and experiment with the demonstrations.

The Demo scenes contain a WorldMapGlobe instance (the prefab) and a Demo gameobject which has a Demo script attached which you can browse to understand how to use some of the properties of the asset from code (C#).

Support & Contact info

We hope you find the asset easy and fun to use.
Feel free to contact us for any enquiry.

Kronnect Games
Email: contact@kronnect.com
Support Forum: https://www.kronnect.com/support
Unity Forum Thread: http://forum.unity3d.com/threads/released-world-political-map-globe-edition.343245/

How to use the asset in your project

Open your project in Unity Editor and select from the top menu GameObject / 3D Object / World Political Map Globe Edition option. You can also drag the prefab “WorldMapGlobe” from “Resources/Prefabs” folder to your scene. A WorldMapGlobe Game Object will appear in the hierarcy of your scene. Select it to show custom properties:

Universe Settings

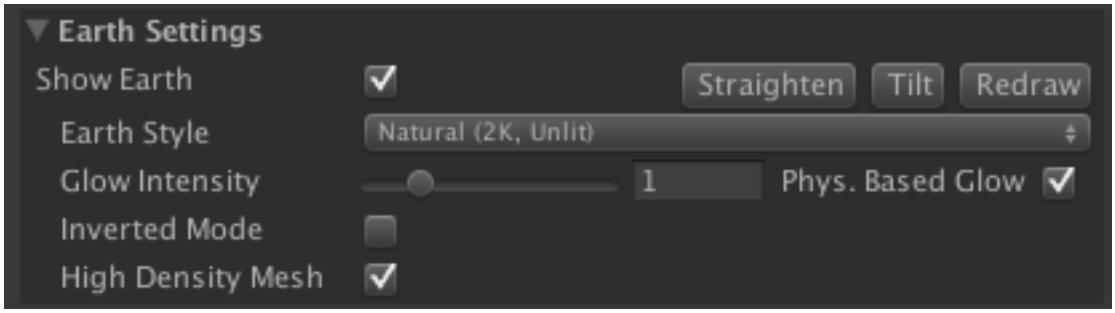
- **Skybox**: “User Defined” will respect your current scene setting. Any other setting will load the skybox provided with the asset. Basic skybox uses a 1024x1024 texture while the Tycho skybox uses 6 textures of 2K resolution of the Milky Way.

- **Sun GameObject:** allows you to specify which directional light will be used as the Sun.

- **Sun Position:** if no Sun game object is set, you can enter the position of the Sun in the scene here. Earth lighting will take into account the light direction.

- **Show Moon**: shows/hide the Moon. Optionally set “Auto Scale” to true, so both the distance and scale of the moon game object will be computed according to current Earth size.

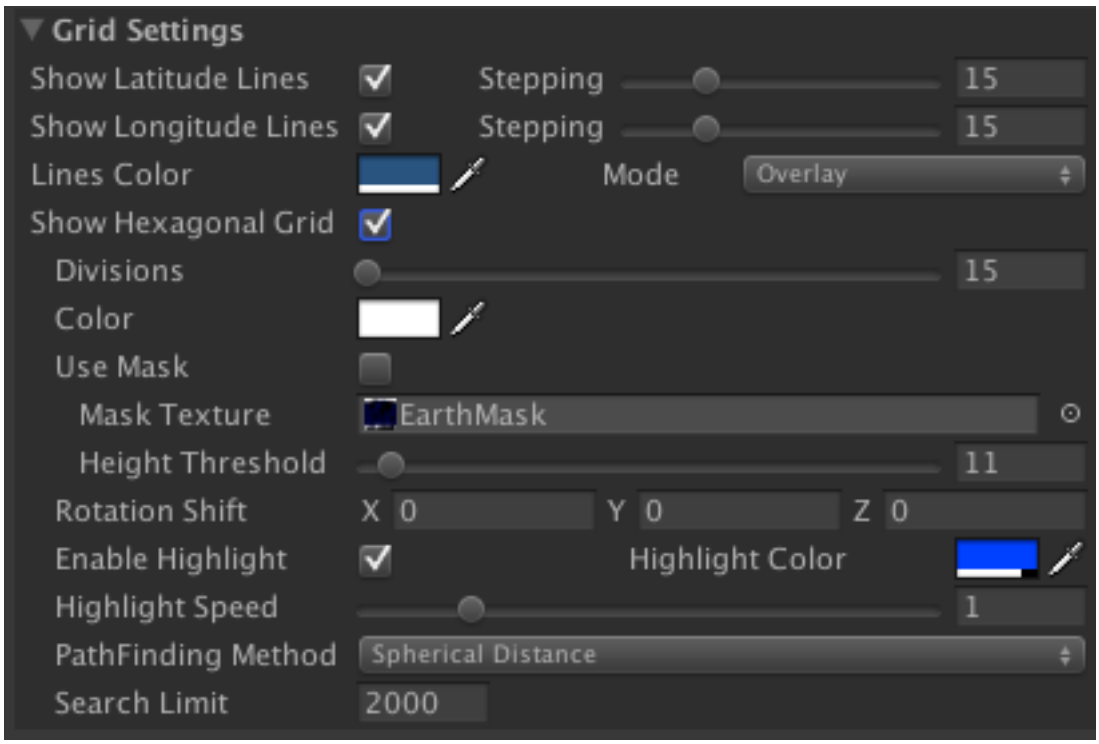
Earth Settings



- **Show Earth**: shows/hide the Earth. You can for example hide the Earth and show only frontiers giving a look of futuristic UI.

- **Earth Style:** changes current texture applied on the Sphere of the prefab.

Grid Settings



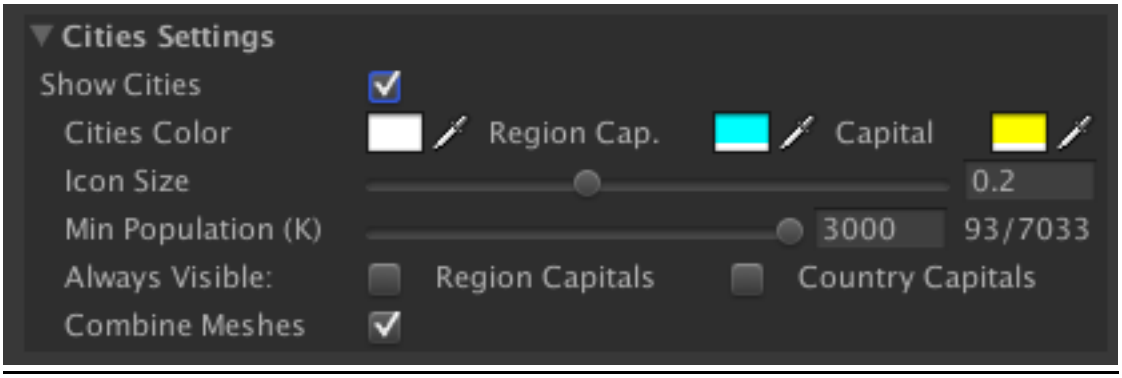
- **Show Latitude/Longitude Lines:** will activate/deactivate the layers of the grid. The stepping options allow you to specify the separation in degrees between lines (for longitude is the number of lines).

- **Lines Color**: modifies the color of the material of the grid (latitude and longitude lines).

- **Mode**: overlay or masked, which will draw the grid only over oceans (note that masking the grid will incur in a performance hit – test it).

(See Hexagonal Grid Settings in the appropriate section below)

Cities Settings



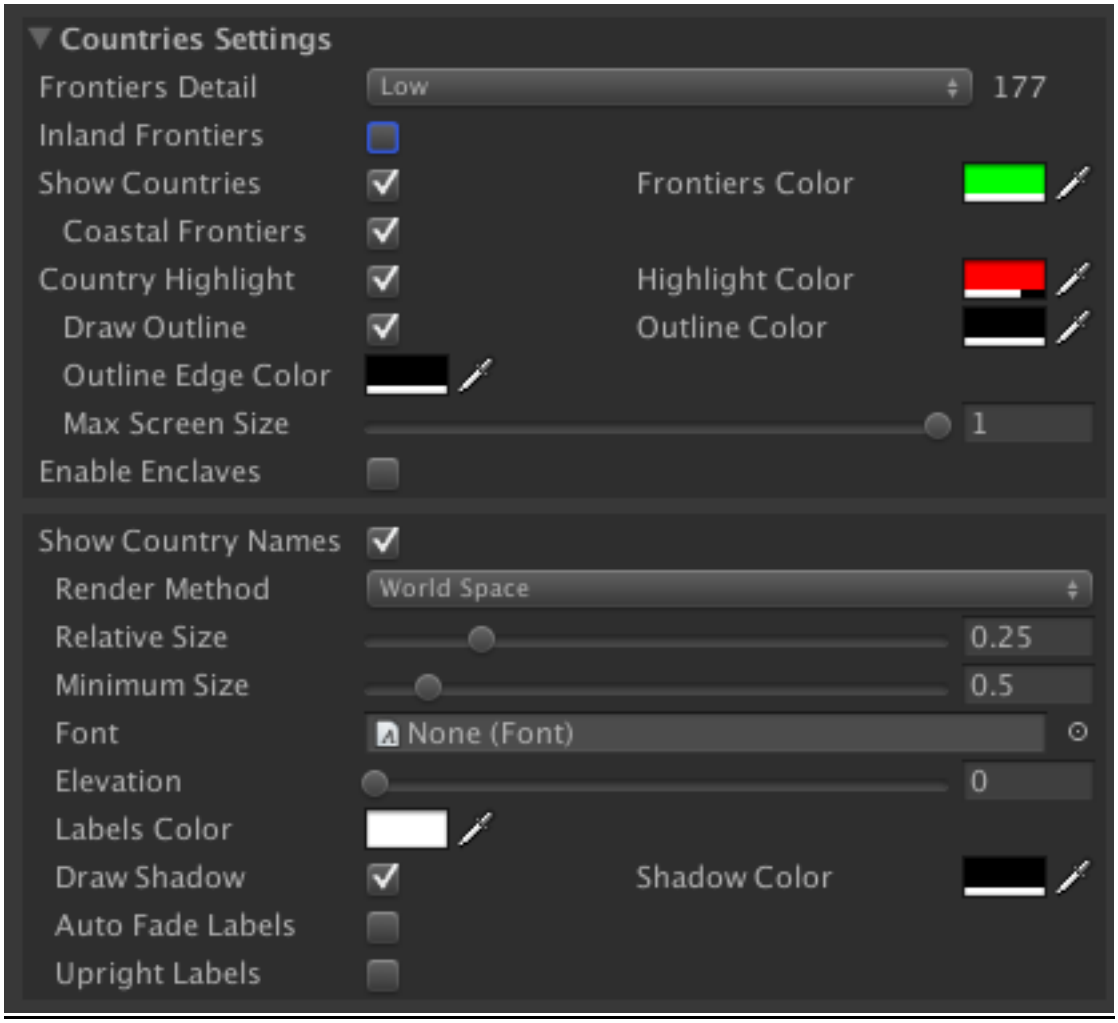
- **Show Cities**: activate/deactivate the layer of cities.

- **Min Population and other city filters**: allows you to filter the cities to be drawn by either minimum population (metropolitan population) or its class (can force to always show region or country capitals).

- **Min Population (K)**: allows to filter cities from current catalog based on population (K = in thousands). When you move the slider to the right/left you will see the number of cities drawn below. Setting this to 0 (zero) will make all cities in the catalog visible.

- **Combine Meshes**: combines all cities meshes into a single mesh improving performance. Note that once combined the dynamic city scaler no longer work, the city icons will retain its current scale.

Country Settings



- **Frontiers Detail:** specify the frontiers data bank in use. Low detail is the default and it's suitable for most cases (it contains definitions for frontiers at 110.000.000:1 scale). If you want to allow zoom to small regions, you may want to change to High setting (30:000:000:1 scale). Note that choosing high detail can impact performance on low-end devices.

- **Inland Frontiers**: show/hide continent borders. This option computes which frontiers segments are shared by two or more countries showing only unique segments (take this into account if you modify frontiers).

- **Show Countries:** show/hide all country frontiers. It applies to all countries, however you can colorize individual countries using the API. If you enable this option, make sure "Inland Frontiers" is disabled to avoid redrawing continent borders.

- **Frontiers Color:** will change the color of the material used for all frontiers.

- **Country Highlight Enabled**: when activated, the countries will be highlighted when mouse hovers them. Current active country can be determined using *countryHighlighted* property.

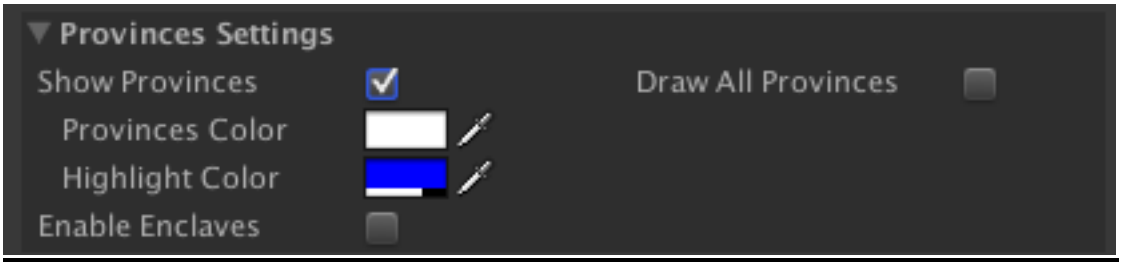
- **Country Highlight Color**: fill color for the highlighted country. Color of the country will revert back to the colorized color if used.

- **Draw Outline** and **Outline Color**: draws a colored border around the colorized or highlighted country.

- **Show Country Names**: when enabled, country labels will be drawn and blended with the Earth map. This feature uses RenderTexture and has the following options:

    - **Render Method**: you can choose either render the labels on a texture that wraps around the globe (Blended) or individual text mesh objects in world space (World Space).
    - **Texture Resolution**: controls the size of the RenderTexture used when Render Method is set to Blended, thus affecting to the resolution of the labels shown in the map. Low quality uses a texture of 2048x1024, Medium 4096x2048 and High 8192x4096.
    - **Relative Size**: controls the amount of "fitness" for the labels. A high value will make labels grow to fill the country area.
    - **Minimum Size**: specifies the minimum size for all labels. This value should be let low, so smaller areas with many countries don't overlap.
    - **Labels and shadow color**: they affect the Font material color and alpha value used for both labels and shadows. If you need to change individual label, you can get a reference to the TextMesh component of each label with Country.labelGameObject field.

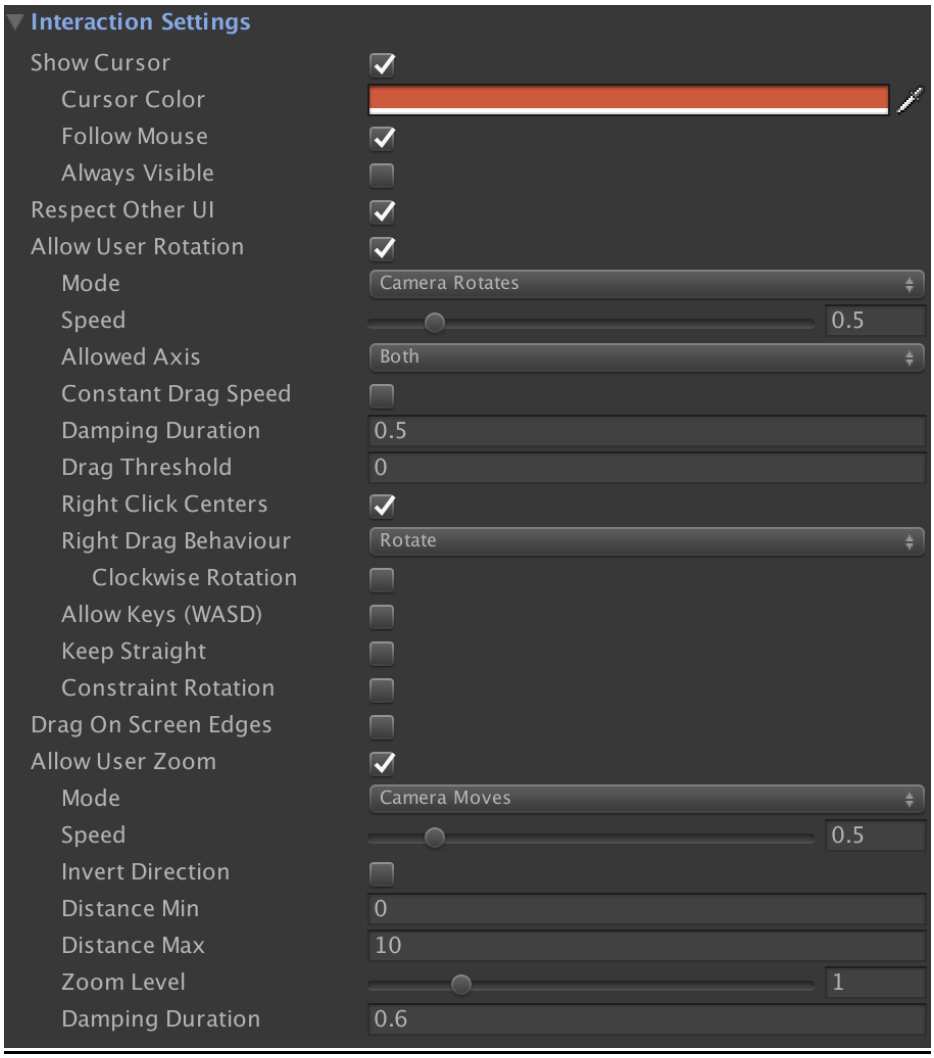
Province Settings



- **Show Provinces**: when enabled, individual provinces/states will be highlighted when mouse hovers them. Current active province can be determined using *provinceHighlighted* property.

- **Enable Enclaves**: when enabled, provinces inside other provinces will be allowed.

Interaction Settings



- **Show Cursor:** will display a cross centered on mouse cursor. Current location of cursor can be obtained with *cursorLocation* property when *mouseOver* property is true.

- **Always Visible:** will not hide the cursor cross when pointer is outside the globe.

- **Respect Other UI:** will prevent any interaction with the globe if pointer is over other UI element.

- **Allow User Rotation:** whether the user can rotate the Earth with the mouse. You can implement your own interactions setting this to false and modifying the rotation / position fields of the gameObject transform.

    - **Right Click Centers:** it will center the selection on the globe when pressing right mouse button.
    - **Constant Drag Speed:** prevents acceleration when dragging/rotating/zooming the globe.
    - **Keep Straight:** will ensure the globe is always vertically oriented at any moment.
    - **Constraint Position:** limits rotation around a sphere position

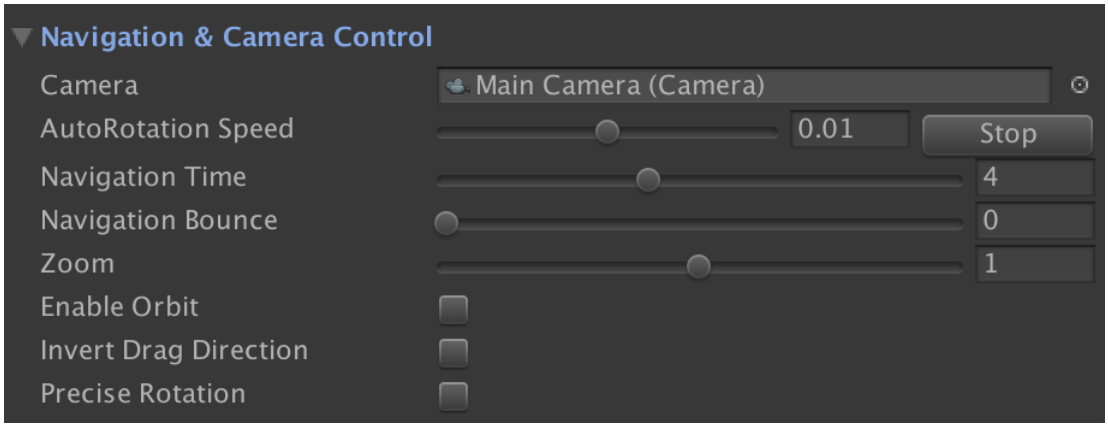
- **Drag On Screen Edges**: performs an automatic drag when mouse is on the edges of screen.

- **Allow User Zoom:** wether the user can zoom in/out the Earth with the mouse wheel.

- **Zoom Speed:** multiplying factor to the zoom in/out caused by the mouse Wheel (Allow User Zoom must be set to true for this setting to have any effect).

- **Distance Min / Max**: minimum and maximum camera distance in World units.

Navigation & Camera Control



- **Camera**: assign your main camera. The system will automatically pick any camera tagged as "MainCamera".

- **Autorotation Speed**: makes the Earth rotate around its vertical axis automatically.

- **Navigation Time**: default duration when navigating to a target country/province/city or location.

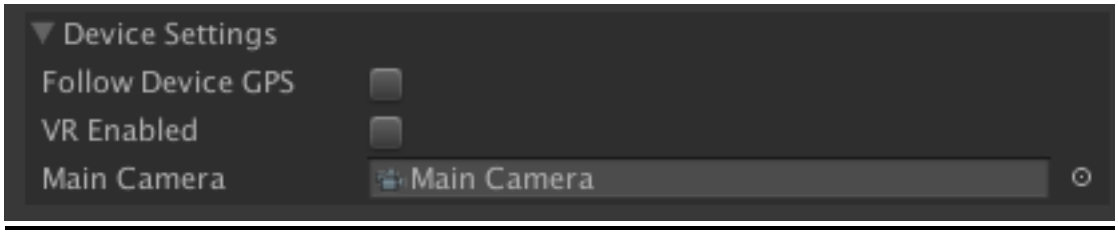
- **Navigation Mode:** this option is very important. You must decide if you want the Earth to be rotated when navigating to a target location or make the Camera rotate around the Earth instead. If you plan to use the asset as part as the UI of your application/game, then the default behaviour (Earth rotates) may suit better since it won't affect the main camera. Otherwise, choose "Camera rotates" which will make the camera fly around the Earth.

- **Zoom**: let you set the zoom (distance from camera to Earth) using a value between 0 (ground or minimum distance) to 1 (distance where the Earth can be seen entirely in the screen). Optionally you can push this value further.

- **Enable Orbit:** activates orbiting controls in which you can specify custom pitch/yaw angles.

- **Precise Rotation:** this option is designed to be used with tile system mode. It will switch Rotate Mode to Camera Rotates when zoom is beyond certain zoom level and switches it back to Earth Rotates when zoom level is reduced. This change prevents floating point issues by keeping the camera near the origin and only allow it to drag/rotate when it's very near to the surface of the Earth, minimizing the distance travelled in world space.

Device Settings

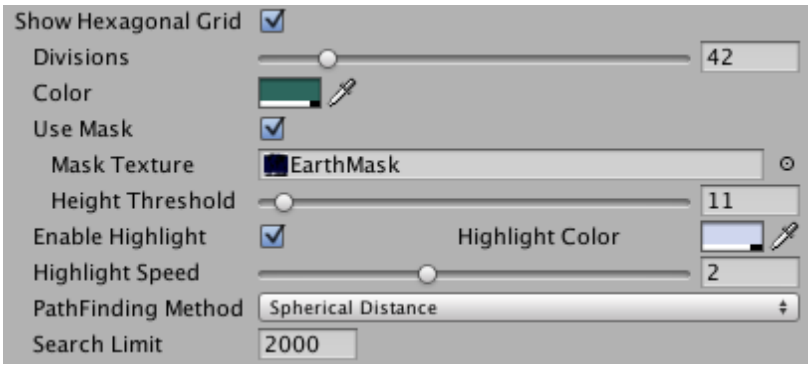


- **Follow Device GPS**: if set to true, the map will always be centered on the coordinates returned by the device GPS.
- **VR Enabled**: forces VR compatibility in normal mode (when inverted mode is enabled, VR compatibility is always on).
- **Main Camera:** assign your main camera here (by default it uses Camera.main). Useful if you need to change cameras at runtime.

Choose Reset option from the gear icon to revert values to factory defaults.

Hexagonal Grid (demo scene 20)

In v9.1, WPM Globe Edition includes the ability to render hexagonal grids over the globe. The section below controls the appearance and behaviour of the grid:



Divisions: the greater the value, the more cells. Note that as you increase the number of cells, the performance will be reduced.

Use Mask: when enabled, it will use the mask texture provided to limit hexagonal cells over land areas exclusing water. The value of the blue channel of the texture (0-255) is used to determine the height for any pixel. The **Height Threshold** specifies the cut value for the blue component value above which cells will be displayed.

Enable Highlight: allows the user to highlight a cell as it moves the pointer over the grid. **Highlight Color** and **Speed** controls the behaviour of the highlight effect.

PathFinding Method: determines which heuristic is used to estimate the shorted path to the destination when calling the FindPath method.

Search Limit: this is the maximum length for any path obtained when calling FindPath.

Check out demo scene 20 for usage example including sample code. Also refer to the API section of this manual to learn about available functions and properties.

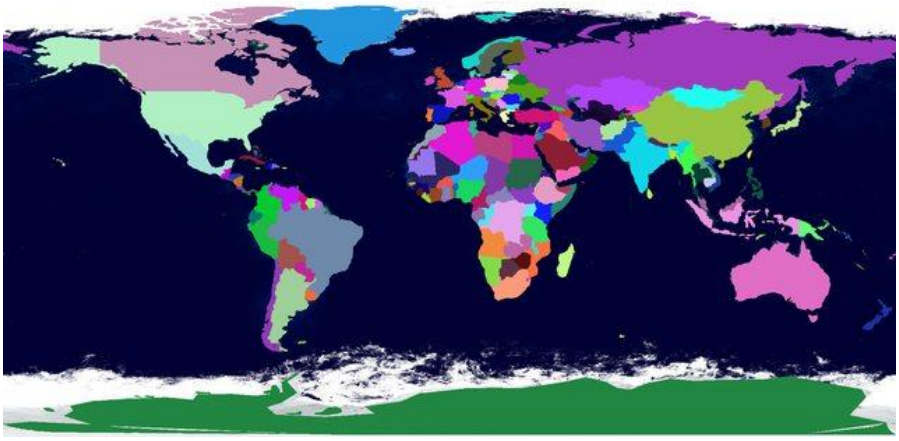
Using the Bake Texture command

The Bake Texture command allows to visualize lots (or all) countries or provinces in colors without incurring in a performance hit (the colorized areas are baked into the current texture so for those cases where you need to colorize lots of countries this will be the way to go).

This command is available from the gear in the inspector title bar:

The generated texture will be saved into Resources/Textures folder in the file EarthCustom. From this moment, you can select the new baked texture from the Style combo and choosing “Custom”.

An example of a generated texture is shown below. To colorize all countries, instead of the Decorator component, a simple loop for all countries was used assigning a random color using the API ToggleCountrySurface.

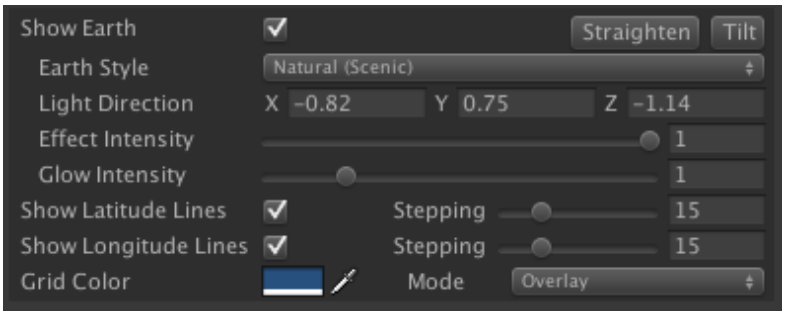


Using the Scenic styles

There're two "scenic" styles available from the Earth style combo (one which uses 2K textures and another high-res one which uses 8K textures).

This scenic style use custom planetary and glow shaders to provide an oustanding effect to the Earth map.

When you select a scenic style, some new global properties are shown in the inspector:

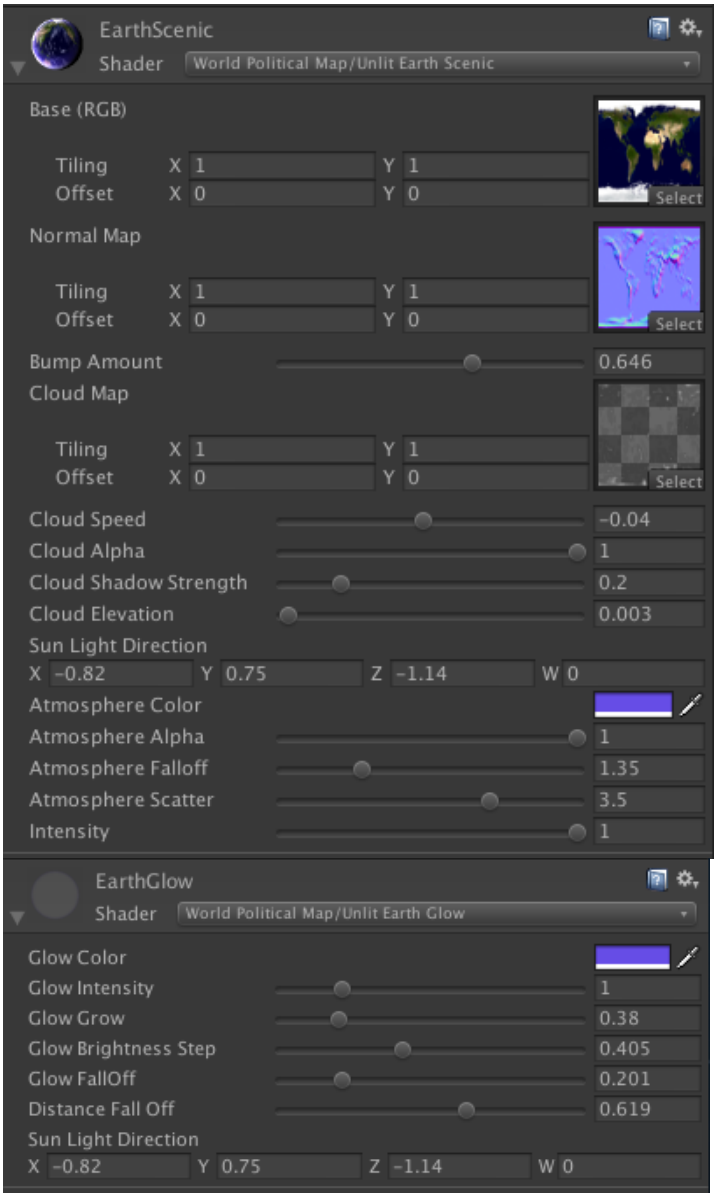


Light Direction: specify the Sun light direction. You can achieve day/light transitions changing this value.

Effect Intensity: controls how much of scenic effect is visible. This includes atmosphere, clouds and relief.

Glow Intensity: controls the brightness of the glow.

But there's more! If you want to fine control the look of the globe, you can go to the Scenic material and change the default values. To do so, scroll down the inspector and expand the Scenic material. You can do the same for the glow material which is attached to a game object called "WorldMapGlobeAtmosphere" which can be found under the WorldMapGlobe in the hierarchy:



Above image: properties of the Earth Glow shader showin in the inspector.

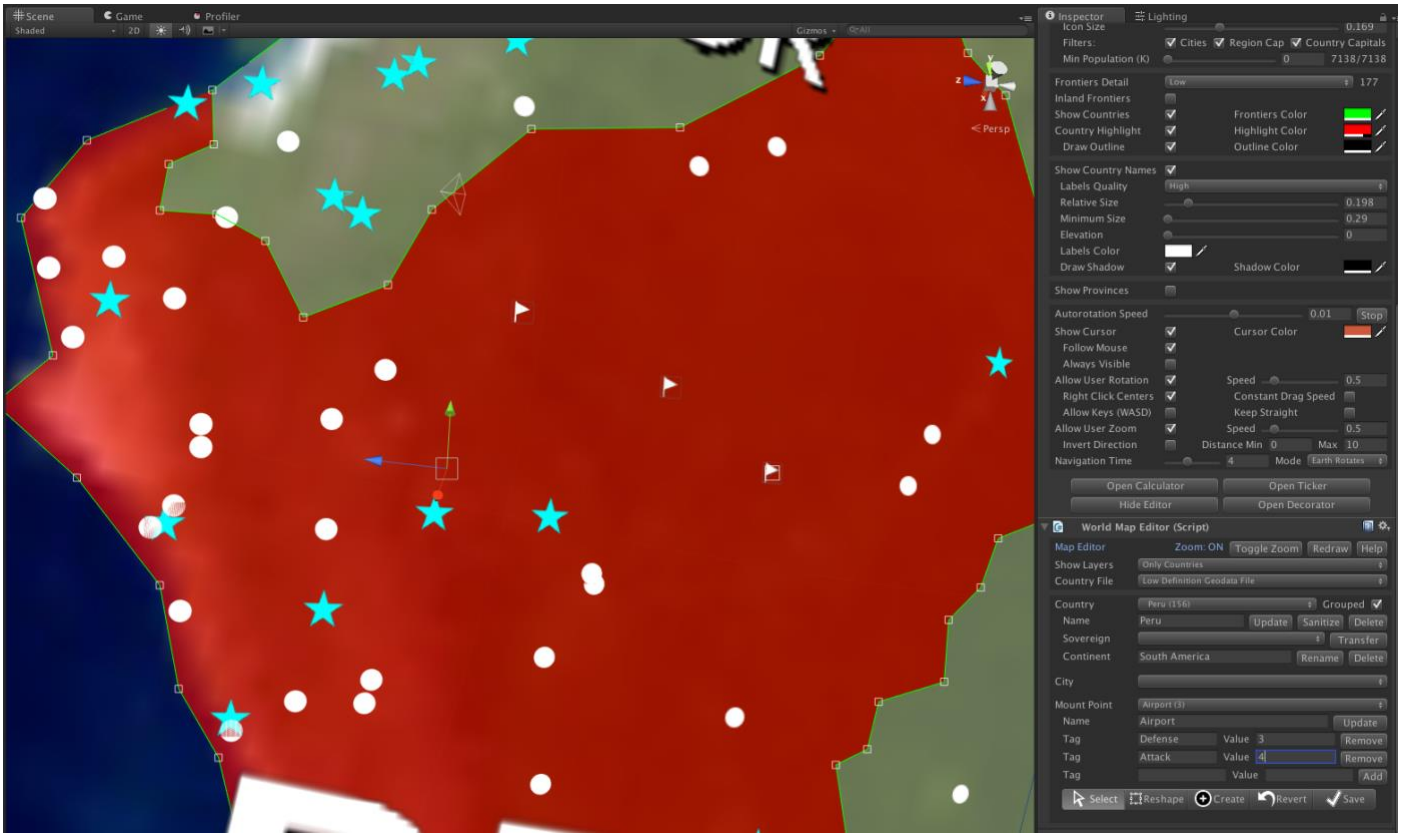
Mount Points

Mount Points are user-defined markers on the map created in the Map Editor. Basically, a mount point is a special location that includes a name, a class identifier (an user-defined number) and a collection of tags.

Mount Points are useful to add user-defined strategic locations, like airports, military units, resources and other landmarks useful for your application or game. To better describe your mount points, WPM allows you to define any number of tags (or attributes) per mount point. The list of tags is implemented as a dictionary of strings pairs, so you can assign each mount point information like ("Defense", "3") and ("Attack", "2"), or ("Capacity", "10"), ("Mineral", "Uranium") and so on.

Note that Mount Points are invisible during play mode since they are only placeholder for your game objects. The list of mount points is accesible through the mountPoints property of the map API.

Mount Points appear during design time (not in playmode) as a flag:

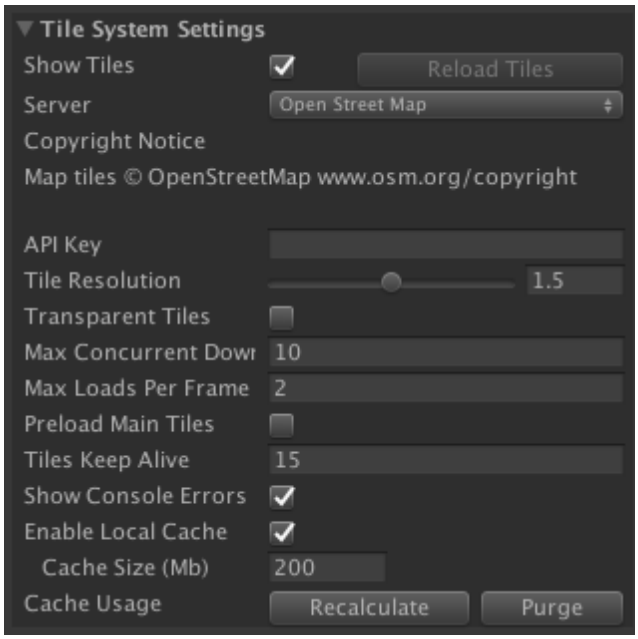


Integrating with Online Tile Map Systems

Since V9, World Political Map Globe Edition can integrate with different tile servers to enable new Earth styles that provides progressive zoom detail. To enable this feature just select “Tiled” as the Earth style in the inspector.

Once you enable “Show Tiles” under Tile System section, the asset will automatically begin showing tiles based on current view and distance to Earth.

Options displayed in this section are:



Server: choose one of the provided tile servers. You can extend the list by editing the file WorldMapGlobeTileServers.cs inside the Scripts folder.

Copyright notice: this is the copyright notice you should show in your applications according to the license you may have purchased from online map systems (see API Key below).

API Key: each server provides their tiles subject to terms of use. Most servers allows you to use them for free as long as you put their copyright notices visible in your application UI. Please refer to each server documentation online as their terms of use can change. Some servers may require you to sign up and obtain an API key or even purchase a special license if you exceed usage limitation for free tiers.

Currently the tile severs included are:

Tile Services	Terms of use / Copyright page
OpenStreeMap	http://www.osm.org/copyright
Stamen	http://maps.stamen.com
Carto	https://carto.com/location-data-services/basemaps/
Wikimedia Atlas	https://wikimediafoundation.org/wiki/Maps_Terms_of_Use
Thunderforest	http://thunderforest.com/terms/
OpenTopoMap	https://opentopomap.org/credits
MapBox	https://www.mapbox.com/pricing/
Sputnik	http://corp.sputnik.ru/maps
AerisWeather	https://aerisweather.com

Transparent Tiles: when enabled, all tiles will use a transparent material which enables seeing through them the globe texture. This option is useful when adding certain types of layers, like clouds.

Tile Resolution: determines the maximum scaling when zooming in. A higher value will provide the most sharp resolution but also will lead to more tile downloads.

Max Concurrent Downloads: determines the maximum number of tile downloads at any given time. This value can be increased depending of the quality and bandwidth of your Internet connection.

Max Loads Per Frame: this is the maximum number of tiles showing up per frame. This option just refers to how many tiles will be activated per frame once they have been downloaded. When one tile is downloaded it's activated and it appears on the globe with a fade animation. This parameter controls the maximum number of animations started per frame.

Show Console Errors: if enabled, any tile request or download error will be printed out to the console if running inside Unity Editor or to the player.log file if running in a build).

Enable Local Cache: stores downloaded tiles locally into your device. The local cache size defaults to 50 Mb. The cache will automatically remove older tiles. You may increase this value to allow offline tile browsing.

Cache Usage: press Recalculate to display current usage of the local cache space. Press Purge to to remove those files.

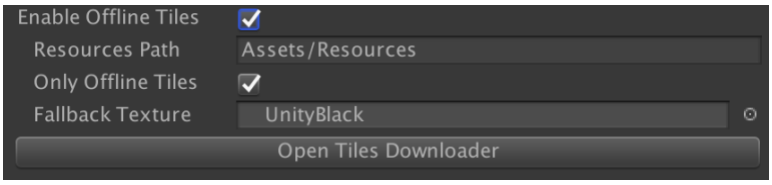
Note that you have also access to a tile API set of functions and properties which you can use to control and customize the tile system feature with scripting.

Downloading and embedding tiles with your application

Since 11.1 version, it's possible to download and load map tiles from your application bundle directly. WPM Globe Edition includes a Tile Downloader assistant that enables you to select a zoom level range and world area and fetch the tiles to a custom Resources folder inside your Unity application.

Each tile is a small PNG image file with name z_x_y.png referring to the zoom level and x/y tile coordinates. Tiles are stored in a subfolder with a number corresponding to the tile server enum. Please note that number of tiles grow exponentially with the zoom level. For a table of number of tiles per zoom level refer to https://wiki.openstreetmap.org/wiki/Tile_disk_usage

The new options are located in the Tile System section of Globe's inspector:

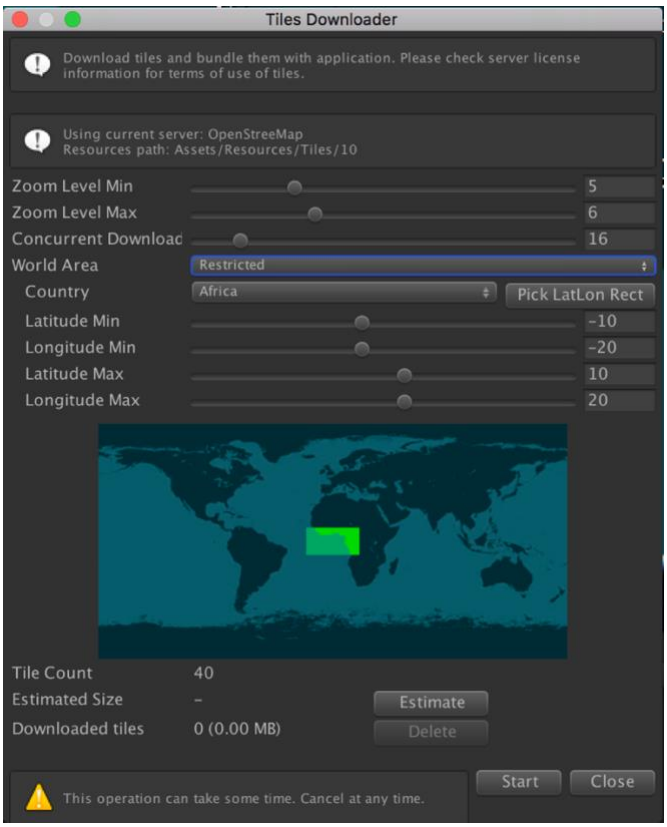


Resources Path: specify the root in the application project where the downloaded tiles will be stored. This path must be contained within the Assets folder at any level but must end with the "Resources" word.

Only Offline Tiles: if enabled only tiles available in the resource path will be used. If the user zooms or navigate to an area where no tiles are available, the **Fallback texture** will be used for those tiles.

Important: it's recommended to set the Max Zoom Level setting to the higher zoom level of the downloaded tiles.

Click "Open Tiles Downloader" to open the downloader assistant:



The downloader assistant shows the following options:

Zoom Level Min / Max: the range of zoom levels to download.

Concurrent Downloads: maximum number of simultaneous downloads. If your Internet connection and tile server can afford it, you can increase this number to download more tiles per second.

World Area: choose between “Full World” or “Restricted”. The restricted mode will download tiles within a given rectangle defined by latitude/longitude. For convenience a list of countries are shown which you can select to quickly select their rect area in the world.

Tile Count: this is the number of tiles that will be downloaded according to the range of zoom levels and selected world area.

Estimated Size: click “Estimate” to download a sample of tiles and produce an estimation of storage size for all the tiles. This is a very rough number and the actual size may vary.

Downloaded Tiles: total number and size of currently downloaded tiles.

Remarks:

- Use a limited zoom level to download tiles (ie. 5-6). As number of tiles grow exponentially with the zoom level, it’s not practical to include higher zoom levels unless you restrict the area.
- The downloader can be stopped at any time. When you click “Start” it will not download/replace any previously downloaded tile, so work can continue.
- If “Only Offline Tiles” is not checked, then the Tile System will try to load the tile from the Resources path. If not found, it will search in the Local Cache (if cache is enabled). Finally it will try to download it from the remote tile server.

Usage tips:

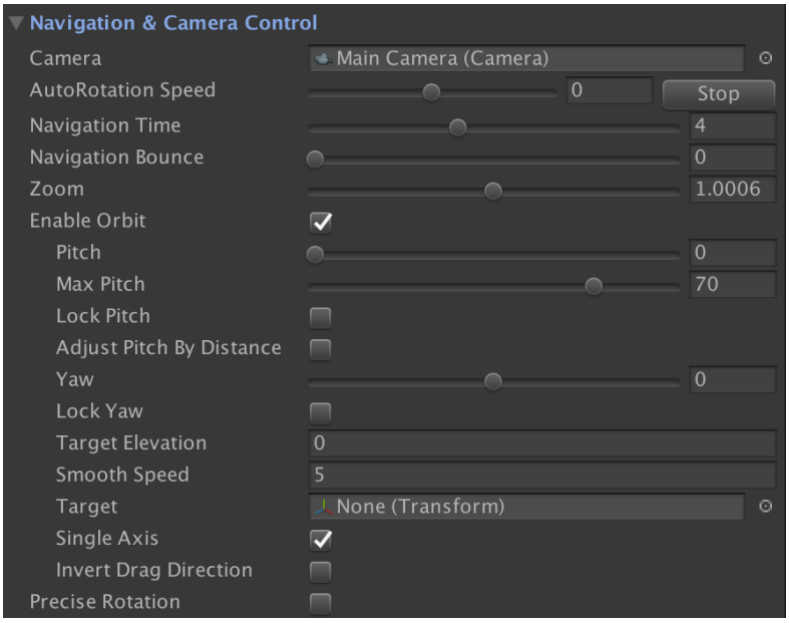
- If you just want to accelerate the load of the tiles in your application, you can download the tiles for only zoom level 5. Do NOT enable the “Only Offline Tiles” so the application will automatically download tiles for higher zoom levels.
- If you want to limit your application to a restricted set of tiles, then enable the “Only Offline Tiles” and make sure you download all the tiles using the Tiles Downloader. Remember that the Tiles Downloader assistant won’t remove any downloaded tile unless you click the “Delete” button. This behaviour allows you to combine different rectangle areas per different zoom levels.

Navigation

The asset supports many navigation options distributed in the Interaction and “Navigation & Camera Control” sections.

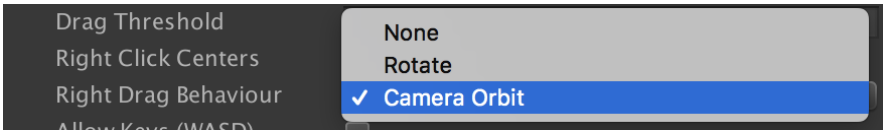
Within Interaction section, you can customize the drag (rotation) and zoom behaviours, decide if it’s the Earth or the camera which moves/rotates, and much more.

In Navigation & Camera Control section you can customize the orbit mode:



When “Enable Orbit” is enabled, you can control camera pitch/yaw as well as use the OrbitTo() and other orbit APIs.

Also, you can set the right drag behaviour to “Orbit” under the Interaction section:



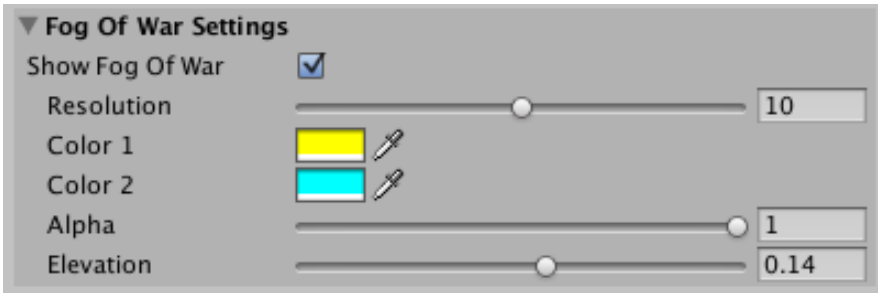
Now, you can use the right mouse button to modify the pitch/yaw of the camera.

Check the Navigation methods in the API section.

Using Fog of War feature

The fog of war feature adds a foggy layer on top of the Globe. It allows you to occlude areas that you won't to be visible to the user or player. When this feature is enabled, the entire globe is covered by fog.

The fog appearance can be customized using the provided section in the inspector:



Resolution: defines the size of the internal texture used to mask contents behind the fog. The texture size is 2^this value – for instance, a value of 10 means 2^10 = 1024 pixels. The greater the resolution the finer the granularity of the fog but it takes more memory and the performance is worse.

Color 1 and Color 2: they blend along an internal noise texture to create the turbulence effect.

Alpha: overall transparency of the fog.

Elevation: thickness of the fog around the globe.

WPM Globe Edition includes a demo scene (#11) with example code of how to clear/reset the globe with fog including methods for clearing custom positions, countries, provinces or cells.

Please refer to the Fog of War API section in this manual for a complete list of available functions.

Virtual Reality (VR)

Basic VR Usage

World Political Map Globe Edition works great in VR. When using it with a VR headset, two different modes can be used:

- "Inverted Mode" which sits you at the center of the globe and it rotates around you. This option is located under the Earth style in the inspector and automatically enables VR compatibility (you don't need to configure anything else). Note that this option is not available for all Earth styles.
- "Normal Mode". This is the mode where you see the entire Earth in front of you. In this case you need to verify that the "VR Enabled" parameter is checked (located at bottom of inspector).

To enable VR raycasting support, check the VR Enable toggle under Devices section of WPM Globe inspector.
This is enough to support basic virtual reality platforms such us Google VR compatible devices, Gear VR, and others.

Custom Gazes & Pointers

Unity's VR Samples Gaze & VREyeRayCast

WPM Globe Edition also supports gaze selections through VREyeRayCast, included in Unity VR Samples. With VREyeRayCast, WPM Globe will properly hide current selection if the gaze is over a interactible element (like a menú). To support VREyeRayCast, you need to import these VR scripts from Unity and also edit WPMInternal.cs and uncomment the line that reads:

```
//#define VR_EYE_RAY_CAST_SUPPORT
```

Google VR Pointer & Controller Touch

To enable country selection and globe interaction using controller touch edit WPMInternal.cs and uncomment the line that reads:

```
//#define VR_GOOGLE
```

Samsung Gear VR Pointer (old API)

To enable country selection and globe interaction using Samsung Gear VR pointer (laser) uncomment the line that reads:

```
//#define VR_SAMSUNG_GEAR_CONTROLLER
```

Oculus Pluging Integration (Go / Quest) Controllers

To enable globe interaction using Go/Quest controllers, uncomment the line in WPMInternal.cs script that reads:

//#define VR_OCULUS

Important! Make sure the Oculus plugin is present in your project (install it from Asset Store or from the Oculus developer site).

This enables integration with the controllers:

- The Earth cursor will move according to the controller direction (Note: no laser is drawn).
- Dragging over the touchpad (Go) or with Thumbstick (Quest) will rotate the Earth.
- Clicking the touchpad (Go) will center the Earth on the current country/province.
- Pulling the controller trigger will raise a normal click (as with mouse).
- Zoom in/out can be performed on the Quest controller by holding the primary hand trigger while moving up/down the thumbstick

Trouble shooting

If you have rendering issues using Oculus Plugin:

- v18 of Oculus Plugin has some issues with Unity 2018.4 and later. Try downloading v17 from Oculus website.
- Select OVRCameraRig and disable “Use Recommended MSAA Level” in the OVRManager script.

Test/Walk-through with Oculus Quest

1. Create an empty project. Switch to Android.

2. Configure project for VR. Go to Project Settings and under XR, install the XR Management Plugin. Also install the Oculus support in the Android tab under the XR Management section.

3. Import World Political Map Globe Edition asset.

4. Import Oculus Plugin from the Asset Store.

5. If you receive any message to upgrade, click “Yes” (upgrade).

6. Open demo scene 1 of the globe asset.
   - Remove the camera and drag & drop the OVRCameraRig from the Oculus/VR/Prefabs folder.
   - Unselect “Use Recommended MSAA level” in the OVR Manager script.

7. Select the WorldMapGlobe gameobject and move it a bit forward to position (0,0,10) so it appears full in front of the camera. Scroll down and select:
   - Allow User Rotation / Mode = Earth Rotates
   - Allow User Zoom / Mode = Earth Moves
   - Distance Max = 20

8. As explained above, edit WPMInternal.cs and uncomment the #define VR_OCULUS macro.

9. Build the app. Go to Player settings and switch Android minimum API version to 7 at least. You may want to remove many of the 8K/16K textures included in the globe / Resources / Textures folder to speed up the build process.

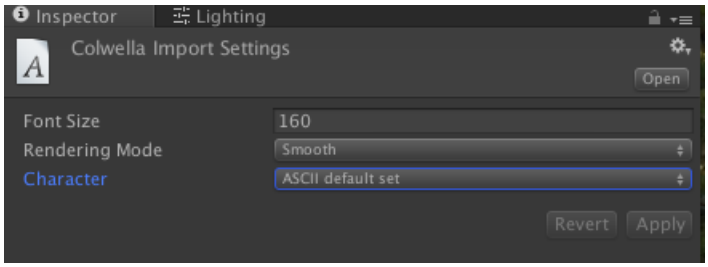
10. Run the app on the Oculus Quest.

On the Oculus Quest using the right hand controller you will be able to control the pointer. Use the thumbstick to rotate the Earth. Hold the lateral button and move the thumbstick up/down to control zoom.

Using custom fonts

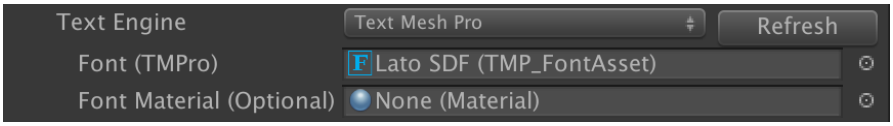
WPM Globe Edition comes with 2 default fonts: Lato and Colwella.

To use another font, make sure it's not marked as "Dynamic" in Import Settings. Select your new font and use the following settings (you can also use another character set if you need to include special characters):



Using Text Mesh Pro

You can also switch the text rendering engine used by the asset to Text Mesh Pro. To do so, just select Text Mesh Pro in the Text Rendering property of the World Map Globe inspector:



If TextMesh Pro Essentials package is not present, the system will show a window where you can import that required package with a button click.

The asset let you specify a custom SDF font and a font material. Please refer to the TextMesh Pro documentation to learn how to create custom SDF fonts from standard fonts. These resources explain the font material presets in depth:
https://learn.unity.com/tutorial/textmesh-pro-working-with-material-presets#
https://www.youtube.com/watch?v=d2MARbDNeaA

Changing default Input system

WPM Globe Edition was designed to use the classic Unity input system (like Input.GetMouseButton methods, etc.). In order to support future and new input systems, the asset has decoupled this usage so all methods are included in a single proxy class, called DefaultUnitySystem.cs. This class implements the interface IInputProxy.

You can therefore implement a new class which implements such interface or create a new class that derive and override the methods of DefaultUnitySystem class, so you provide your own input logic for common tasks like getting the key or button pressed.

The DefaultUnitySystem.cs class can be found in the folder WorldPoliticalMapGlobe/Scripts/Core/Input.

Once you have created your own class, you can just assign it to World Map Globe Edition component at runtime like this:

```
WorldMapGlobe map = WorldMapGlobe.instance;
MyCustomInputSystem newInput = new MyCustomInputSystem();
map.input = newInput;
```

Enabling globe clipping on objects inside the world

The globe sphere is compound of thousands of triangles which approximates well to the sphere curve for most situations. However, when you zoom in or use greater scales, using spheres based on triangles is not a good solution since clipping and z-fighting artifacts will occur on objects near the surface.

Globe Edition rendering system is designed to avoid clipping artifacts on surface due to the imperfection of triangle-based geometry. It works by rendering the different map layers from back to front without writing to z-buffer. As a result, objects around the world will not be clipped.

However, there's a downside – if you need objects inside the globe itself, they will not be clipped! Fortunately, Globe Edition provides you with 2 custom materials/shaders for any objects that you need to go inside the globe and still be clipped correctly. They implement clipping logic in the fragment shader to provide precise clipping around the globe curve without the triangle approximation issue!

The materials are SurfaceGlobeClip (for Standard Shader materials) and UnlitGlobeClip (for unlit materials). You can test a demo of these materials in action opening demo scene #13.

Reducing application build size & memory usage

You can greatly reduce the size of your build removing unused textures and geodata files:

- 17 Earth styles plus Moon and Skybox maps are included with associated textures that ranges from 2K to 8K image files. **Determine which styles you don't use and remove the textures you don't use** from Resources/Textures folder. You could also remove custom shaders that you don't use located in Resources/Shaders folders.

Earth style	Textures used (*.PNG files located in Resources/Textures folder)
Natural (2K, Unlit)	Earth
Natural (2K, Standard Shader)	Earth
Natural (2K, Scenic)	Earth, EarthElevationMap, EarthClouds
Natural (2K, Scenic + City Lights)	Earth, EarthElevationMap, EarthClouds, EarthCityLights
Alternate Style 1 (2K)	Earth2
Alternate Style 2 (2K)	Earth4
Alternate Style 3 (2K)	Earth5
Natural (8K, Unlit)	EarthHighRes8k
Natural (8K, Standard Shader)	EarthHighRes8k
Natural (8K, Scenic)	EarthHighRes8k, EarthElevationMap8k, EarthClouds8k
Natural (8K, Scenic + City Lights)	EarthHighRes8k, EarthElevationMap8k, EarthClouds8k, EarthCityLights8K2
Natural (8K, Scenic Scatter)	EarthHighRes8k, EarthElevationMap8k, EarthClouds8k
Natural (8K, Scenic Scatter + City Lights)	EarthHighRes8k, EarthElevationMap8k, EarthClouds8k, EarthCityLights8K2
Natural (8K)	EarthHighRes8k
Natural (16K)	Earth16K_BL, Earth16K_BR, Earth16K_TL, Earth16K_TR
Natural (16K, Scenic)	Earth16KScenic_BL, Earth16KScenic_BR, Earth16KScenic_TL, Earth16KScenic_TR, , EarthElevationMap8k, EarthClouds8k
Natural (16K, Scenic + City Lights)	Earth16KScenic_BL, Earth16KScenic_BR, Earth16KScenic_TL, Earth16KScenic_TR, , EarthElevationMap8k, EarthClouds8k, EarthCityLights8K2
Natural (16K, Scenic Scatter)	Earth16KScenic_BL, Earth16KScenic_BR, Earth16KScenic_TL, Earth16KScenic_TR, , EarthElevationMap8k, EarthClouds8k
Natural (16K, Scenic Scatter+ City Lights)	Earth16KScenic_BL, Earth16KScenic_BR, Earth16KScenic_TL, Earth16KScenic_TR, , EarthElevationMap8k, EarthClouds8k, EarthCityLights8K2
Custom	EarthCustom

- By default, all textures are provided with TrueColor settings in Import Settings (uncompressed). Consider enabling the compression for the texstures (ie. use Automatic Compression setting). **Compressing the textures will greatly reduce the build size of your application but can also reduce the image quality a bit (mostly unnoticieable)**.
- You may also remove some geodata files located in Resources/Geodata folder. For example, if you don't use provinces, you can remove the provinces10 file. You can also remove the frontiers geodata file corresponding to a resolution you don't use (countries110 is low definition vs countries10 which is high definition frontiers).
- Scenic and Scenic Scatter textures (Earth, EarthHighRes8K and Earth16K*) have water mask baked into alpha channel. This is used for the specular shining effect. If you don't use this option, you can switch the Alpha Import setting to None.
- If labels are rendered in world space and tickers or markers are not used, you can switch the "Overlay Resolution" setting to "Not Used" to save an internal render texture.

Please contact us for any question you may have.

Shaders used by the asset

The table below lists all shaders used by WPM Globe Edition. They can be found in WorldPoliticalMapGlobeEdition / Resources / Shaders folder.

A **required** shader must be kept for basic operation of the asset. All other shaders are required by the related feature. For example, if you don't use the hexagonal grid, you can remove the Hexa* family of shaders.

Shader Filename	Used for…	Required
BackFacesForZBuffer	Used internally to clip geometry behind the globe.	Yes
UnlitOverlay / UnlitOverlayInverted	Overlay layer	Yes
FogOfWar (.shader/.cginc), FogOfWar Painter	Fog of war effect	Optional
HexaGridNoExtrusion, HexaGridNoExtrusionAlpha, HexaTile, HexaTileAlpha, HexaTileHighlighted HexaTileTextured, HexaTileTexturedAlpha	Hexagonal grid	Optional
Moon	Moon texturing	Optional
SurfaceGlobeClip, UnlitGlobeClip	Render objects that clips with globe surface (Standard Shader and Unlit versions)	Optional
UnlitCities	Cities	Optional
UnlitCountryHighlight	Country highlighting	Optional
UnlitCountrySurfaces*	Country coloring/texturing	Optional
UnlitEarth16K*	Earth textures for 16K styles	Optional
UnlitEarthGlow	Atmosphere effect (non-physically based)	Optional
UnlitEarthGlow2	Atmosphere effect (physically based)	Optional
UnlitEarthScenic	Earth scenic style	Optional
UnlitEarthScenicCityLights	Earth scenic style + city lights at night	Optional
UnlitEarthSingleColor	Earth style with single flat color	Optional
UnlitEarthStandardShader	Earth style that uses Standard Shader	Optional
UnlitGUIText	Country labels	Optional
UnlitMarker	Polygon markers on globe (eg. circles)	Optional
UnlitMarkerLine	Lines or trajectories	Optional
UnlitOutline	Country outline when highlighted	Optional
UnlitProvinceHighlight	Province highlighting	Optional
UnlitProvinceSurfaces*	Province coloring/texturing	Optional
UnlitSingleColorCursor	Cursor	Optional
UnlitSingleColorFrontiers*	Country frontiers	Optional
UnlitSingleColorMasked	Latitude / longitude lines (masked with oceans option)	Optional
UnlitSingleColorOrder 1*	Inland frontiers	Optional
UnlitSingleColorOrder 2*	Latitude / Longitude lines (without masking)	Optional
UnlitSingleTexture	Earth texture & Earth alternate styles	Optional
UnlitTickersBackground	Tickers	Optional
UnlitTile*	Tiles (slippy map)	Optional

Programming Guide (API)

You can instantiate the prefab “WorldMapGlobe“ or add it to the scene from the Editor. Once the prefab is in the scene, you can access the functionality from code through the static instance property:

```
using WPM; // imports the namespace

WorldMapGlobe map;

void Start () {
map = WorldMapGlobe.instance;
        ...
}
```

(Note that you can have more WorldMapGlobe instances in the same scene. In this case, the instance property will returns the same object. To use the API on a specific instance, you can get the WorldMapGlobe component of the GameObject).

Structure of the API

To make it easier to find properties and methods in the source code, it has been subdivided in several files with same prefix WorldMapGlobe*:

WorldMapGlobe.cs: contains basic functionality to access the API, like WorldMapGlobe.instance above.
WorldMapGlobeCities: all related to cities. Similar for Countries, Provinces, Mount Points, Earth And Continents, as well as Markers and Lines.
Finally, all properties and methods related with user interaction is available in WorldMapGlobeInteraction.

Countries API

Country object fields

country.name: name of the country.

country.hidden: makes the country invisible on the map.

country.continent: name of the continent to which the country belongs to.

country.fips10_4, iso_a2, iso_a3, iso_n3: standardized codes of the country.

country.regions: list of physical regions. Each region contains a list of frontier points.

country.mainRegionIndex: the index of the biggest region in the regions list.

country.allowShowProvinces: if set to false, province borders won't be shown when pointer is over the country (requires ShowProvinces = true).

country.localPosition: center of the country on the sphere in local space.

country.latLonCenter: latitude/longitude of the center of the country.

country.customLabel: overrides the country name with a custom string (just for the label).

country.labelColorOverride: set to true to override the country color just for the label.

country.labelColor: custom color of the country label (needs labelColorOverride = true).

country.labelFontOverride: overrides the default font setting for this country.

country.labelVisible: if country name must be drawn.

country.labelRotation and labelOffset: custom rotation and displacement for this country label.

Country methods and properties

map.countries: return a List<Country> of all countries records.

map.countryHighlighted: returns the Country object for the country under the mouse cursor (or null).

map.countryHighlightedIndex: returns the index of country under the mouse cursor (or null if none).

map.countryRegionHIghlightedIndex: returns the index of the region of currently highlighted country.

map.countryLastClicked: returns the index of last clicked country.

map.countryRegionLastClicked: returns the index of last clicked country region.

map.enableCountryHighlight: set it to true to allow countries to be highlighted when mouse pass over them.

map.fillColor: color for the highlight of countries.

map.showCountryNames: enables/disables country labeling on the map.

map.showOutline: draws a border around countries highlightes or colored.

map.outlineColor: color of the outline.

map.showFrontiers: show/hide country frontiers. Same than inspector property.

map.frontiersDetail: detail level for frontiers. Specify the frontiers catalog to be used.

map.frontiersColor: color for all frontiers.

map.GetCountry(name): given a country name returns the Country object.

map.GetCountry(index): given a country index returns the Country object. Same than map.countries[index].

map.GetCountryIndex(name): returns the index of the country in the collection.

map.GetCountryIndexByFIPS10_4(fips): returns the index of the country in the collection by FIPS code.

map.GetCountryIndexByISO_A2(iso_a2): returns the index of the country in the collection by ISO 2-character code.

map.GetCountryIndexByISO_A3(iso_a3): returns the index of the country in the collection by ISO 3-character code.

map.GetCountryIndexByISO_N3(iso_n3): returns the index of the country in the collection by ISO 3-digit code.

map.GetCountryIndex(localPosition): returns the country that contains a map coordinate.

map.GetCountryNearToPoint(localPosition): returns the country that contains a map coordinate or the country whose center is nearest to the map coordinate.

map.GetCountryUnderSpherePosition(spherePoint, out countryIndex, out countryRegionIndex): returns the index of the country and region located in sphere point provided.

map.ToggleCountrySurface(name, visible, color): colorize one country with color provided or hide its surface (if visible = false).

map.ToggleCountrySurface(index, visible, color): same but passing the index of the country instead of the name.

map.ToggleCountryMainRegionSurface(index, visible, color, Texture2D texture): colorize and apply an optional texture to the main region of a country.

map.ToggleCountryRegionSurface(countryIndex, regionIndex, visible, color): same but only affects one single region of the country (not province/state but geographic region).

map.HideCountrySurface(countryIndex): un-colorize / hide specified country.

map.HideCountryRegionSurface(countryIndex): un-colorize / hide specified region of a country (not province/state but geographic region).

map.HideCountrySurfaces: un-colorize / hide all colorized countries (cancels ToggleCountrySurface).

map.DrawCountryOutline: adds a colored outline to a country.

map.ToggleCountryOutline: toggles on/off country outline or creates the outline if it didn't exist.

map.CountryNeighbours (int countryIndex): returns a List of Countries which are neighbours of specified country index. Note that a Country can have one or more land regions (separated) which can have different neighbours each. This method returns all neighbours for all land regions of the country.

map.CountryNeighboursOfMainRegion (int countryIndex): returns a list of neighbours of the specified country having into account only the main land region of the country.

map.CountryNeighboursOfCurrentRegion (): returns a list of countries neighbours of currently selected country's region.

map.GetCountryUnderSpherePosition(spherePoint, out countryIndex, out countryRegionIndex): returns true/false and the index of the country/region of a country under specified sphere position (you can use the calculator component to convert lat/lon to spherePoint).

map.GetCountryRegionSurfaceGameObject(countryIndex, regionIndex): returns the game object corresponding to the colored surface of a given country and region, or null if that country has not been colored yet.

map.GetVisibleCountries: returns a list of visible countries on the screen.

map.CountryTransferCountryRegion: merges one country into another country.

map.CountryTransferProvinceRegion: merges one province into another country.

map.CountryTransferCell: merges one cell into a country.

map.CountrySetProvinces(countryIndex, provinces, mergeRegions, updateCities): updates country frontiers by the result of merging the land regions covered by a list of provinces. The parameter mergeRegions specifies if the resulting region will be one (merge) or many (one per province) and by default is true (merge). The parameter updateCities specifies if the cities should also be linked to the given country (by default is true).

map.CountryAddProvinces: similar to the previous methods but preserves existing country provinces and allows you to pass a list of provinces.

Map.CountryRemoveProvinces: similar to CountrySetProvinces. This method updates the country frontiers by replacing the current provinces of that country by the result of removing a list of given provinces.

map.GetCountryMainRegionZoomExtents: returns the zoom level which makes this country's main region occupy the entire screen

map.GetCountryRegionZoomExtents: returns the zoom level which makes this country's region occupy the entire screen.

Provinces API

Province object fields

province.name: name of the province.

province.countryIndex: index of the country in the countries list.

province.hidden: shows/hide province borders.

province.regions: list of physical regions. Each region contains a list of frontier points.

province.mainRegionIndex: the index of the biggest region in the regions list.

province.localPosition: center of the province on the sphere in local space.

province.latLonCenter: latitude/longitude of the center of the province.

province.mainRegionArea: approximate surface area (width x height) of the main region.

Provinces methods and properties

map.provinces: return a List<Province> of all provinces/states records.

map.provinceHighlighted: returns the province/state object in the provinces list under the mouse cursor (or null if none).

map.provinceHighlightedIndex: returns the province/state index in the provinces list under the mouse cursor (or null if none).

map.GetProvinceIndex(countryIndex, provinceName): returns the province index matching the given name and belonging to specified country.

map.GetProvinceIndex(localPosition): returns the province that contains a map coordinate.

map.GetProvinceNearToPoint(localPosition): returns the province that contains a map coordinate or the province whose center is nearest to the map coordinate.

map.GetProvinceUnderSpherePosition(countryIndex, spherePoint, out provinceIndex, out provinceRegionIndex): returns the index of the province and region located in sphere point provided. Must provide the country index to which the province belongs to.

map.provinceLastClicked: returns the index of last clicked province.

map.provinceRegionLastClicked: returns the index of last province region clicked.

map.showProvinces: show/hide provinces when mouse enters a country. Same than inspector property.

map.provincesFillColor: color for the highlight of provinces.

map.provincesColor: color for provinces/states color.

map.ToggleProvinceSurface(name, visible, color): colorize one province with color provided or hide its surface (if visible = false).

map.ToggleProvinceSurface(index, visible, color): same but passing the index of the country instead of the name.

map.ToggleProvinceSurface(index, visible, color, texture, …): same can use a texture with scale, offset and rotation options.

map.HideProvinceSurface(countryIndex): un-colorize / hide specified province.

map.HideProvinceSurfaces: un-colorize / hide all colorized provinces (cancels ToggleProvinceSurface).

map.ProvinceNeighbours (int provinceIndex): returns a List of Provinces which are neighbours of specified province index. Note that a province can have one or more land regions (separated) which can have different neighbours each. This method returns all neighbours for all land regions of the province.

map.ProvinceNeighboursOfMainRegion (int provinceIndex): returns a list of neighbours of the specified province having into account only the main land region of the province.

map.ProvinceNeighboursOfCurrentRegion (): returns a list of province neighbours of currently selected province's region.

map.GetProvinceRegionSurfaceGameObject(countryIndex, regionIndex): returns the game object corresponding to the colored surface of a given province and region, or null if that province has not been colored yet.

map.GetVisibleProvinces: returns a list of visible provinces on the screen.

map.ProvinceTransferProvinceRegion: merges one province into another province.

map.ProvincesMerge(List<Province>…): merges a group of provinces.

map.ProvinceTransferCell: merges one cell into a province.

map.GetProvinceMainRegionZoomExtents: returns the zoom level which makes this province's main region occupy the entire screen

map.GetProvinceRegionZoomExtents: returns the zoom level which makes this province's region occupy the entire screen

Cities API

City object fields

city.name: name of the city.

city.fullName: returns name of city + province + country names.

city.countryIndex: index of the country in the countries list.

city.province: name of the province in the provinces list.

city.localPosition: position of the city on the sphere in local space.

city.population: approximate metropolitan population.

city.cityClass: type of city (country capital, region capital, normal city).

city.latitude / longitude: latitude and longitude of the city.

Cities methods and properties

map.cities: return a List<City> of all cities records.

map.cityHighlighted: returns the city under the mouse cursor (or null if none).

map.cityLastClicked: returns the index of last clicked city.

map.showCities: show/hide all cities. Same than inspector property.

map.minPopulation: the mínimum population amount for a city to appear on the map (in thousands). Set to zero to show all cities in the current catalog. Range: 0 .. 17000.

map.cityClassFilter: bitwise filter which specifies the class of cities to be drawn (Normal/Region Capitals/Country Capitals). See CITY_CLASS_FILTER enum for bit flags.

map.numCitiesDrawn: number of cities actually drawn.

map.citiesColor: color for the normal cities.

map.citiesRegionCapitalColor: color for the region capital cities.

map.citiesCountryCapitalColor: color for the country capital cities.

map.cityIconSize: custom scaling for city icons.

map.cityClassAlwaysShow: bitwise mask for the type of cities to be drawn (2 = region capitals, 4 = country capitals).

map.combineCityMeshes: whether the cities meshes should be combined into one, improving performance when lot of cities are drawn. If set to true, automatic city scaling based on distance to camera will be disabled.

map.GetVisibleCities: returns a list of visible cities on the screen.

map.GetCityNames: returns a list of city names.

map.GetCityIndex: returns the index of a given City based on different criteria or a random city if no argument is passed.

map.GetCityIndexInProvince: returns the index of a given City belonging to a province.

map.GetCityIndexInCountry: returns the index of a given City belonging to a country.

map.GetCountryCapital: returns the city object which is the capital of the given country.

map.GetCountryCapitalIndex: returns the city index of the country capital.

map.GetCityRandom: returns a random city from a country, province or map.

map.GetCityIndexRandom: returns a random city index from a country, province or map.

Mount Points API

map.mountPoints: return a List<MountPoint> of all mount points records.

map.GetMountPointNearPoint: returns the nearest mount point to a location on the sphere.

map.GetMountPoints: returns a list of mount points, optionally filtered by country, province or region.

map.GetVisibleMountPoints: returns a list of mount points that are “visible” on the screen.

A MountPoint object can be created using the Map Editor or scripting. If you use scripting, you can create a MountPoint and add it to the mountPoints list as follows:

```
MountPoint myMountPoint = new MountPoint();
map.mountPoints.Add(myMountPoint);
```

The MountPoint class contains the following data:

- Name (string): user defined mount point name (optional)
- Type (int): user defined integer that can be used to distinguish mount point types (optional)
- CountryIndex: the country to which the mount point belongs to.
- ProvinceIndex: same but with province.
- localPosition: a Vector3 with the local position of the mount point on the sphere.
- latlon: a Vector2 that contains the latitude/longitude of the mount point.
- customTags a dictionary<string,string> used to store any kind of additional data.

Earth/Globe API

map.showEarth: show/hide the planet Earth. Same than inspector property.

map.earthStyle: the currently texture used in the Earth.

map.autoRotationSpeed: the speed of the automatic/continuous rotation of the Earth.

map.showLatitudeLines: draw latitude lines.

map.latitudeStepping: separation in degrees between each latitude line.

map.showLongitudeLines: draw longitude lines.

map.longitudeStepping: number of longitude lines.

map.gridColor: color of latitude and longitude lines.

map.earthScenicLightDirection: Sun light direction for the scenic styles.

map.earthScenicAtmosphereIntensity: intensity of the scenic effect (0-1).

map.earthScenicGlowIntensity: brightness of the glow (0-5).

map.earthTexture: returns an instanted texture of the Earth. Useful for texture drawing. See demo scene 7.

Navigation API

map.navigationTime: time in seconds to fly to the destination (see FlyTo methods).

map.navigationBounceIntensity: amount (0..1) of bouncing between navigation points.

map.followDeviceGPS: if set to true, the map will be centered on the latitude and longitude coordinates returned by the device GPS.

map.FlyToCountry(name): start navigation at *navigationTime* speed to specified country. The list of country names can be obtained through the cities property.

map.FlyToCountry(index): same but specifying the country index in the countries list.

map.FlyToProvince(name): start navigation at *navigationTime* speed to specified province. The list of provinces names can be obtained through the provinces property.

map.FlyToProvince (index): same but specifying the province index in the provinces list.

map.FlyToCity(name): start navigation at *navigationTime* speed to specified city. The list of city names can be obtained through the cities property.

map.FlyToCity(index): same but specifying the city index in the cities list.

map.FlyToLocation (): same but specifying the location in local Unity spherical coordinates. Optionally pass duration in seconds and destination zoom level to zoom in/out smoothly from current position.

map.GetCurrrentMapLocation: returns the sphere coordinates of the center of the visible map from camera. Similar to cursorLocation but restricted to the center of the globe. Note that cursorLocation requires the pointer to hover the globe while GetCurrentMapLocation will always return the center of the map from the camera perspective.

map.GetZoomLevel(): returns the normalized zoom level based on current distance to Earth (0..1).

map.GetZoomLevel(altitudeInKM): returns the zoom level for an altitude given in km.

map.SetZoomLevel(): sets the normalized zoom level (0..1) based on current distance to Earth.

map.OrbitRotateTo(…): performs a pitch/yaw rotation (only if Enable Orbit is on).

map.DragTowards(Vector2 direction): performs a drag move in the given screen-space direction.

map.ZoomTo(…): performs a zoom transition to specified zoom level (0..1).

map.Stopxxx: several methods to immediately stop any navigation.

Navigation methods with duration provides a callback pattern to chain actions. Example:
map.FlyToCity(..).Then(() => action;);

User Interaction API

map.mouseIsOver: returns true if mouse has entered the Earth's sphere collider.

map.allowUserRotation/map.allowUserZoom: enables/disables user interaction with the map.

map.rotationAllowedAxis: specifies the allowed rotation axis (both, X or Y).

map.mouseWheelSensibility: multiplying factor for the zoom in/out functionality.

map.invertZoomDirection: switch direction of zoom when using the mouse wheel.

map.showCursor: enables the cursor over the map.

map.cursorFollowMouse: makes the cursor follow the map.

map.cursorLocation: current location of cursor in local coordinates (by default the sphere is size (1,1,1) so x/y/z can be in (-0.5,0.5) interval. Can be set and the cursor will move to that coordinate.

map.constraintPosition, map.constraintAngle, map.constraingPositionEnabled: restricts user rotation so a specified center (contraintPosition) is always less than constraintAngle degrees from the center of the screen.

map.centerOnRightClick: enables auto-centering on country/province under mouse.

map.rightClickRotates: enables globe rotation when holding right mouse button.

map.rightClickRotatingClockwise: changes direction of rotation of globe with holding right mouse button.

map.setSimulatedMouseButtonClick: simulates a mouse click (press + release) (0=left mouse button, 1=right mouse button)

map.setSimulatedMouseButtonPressed: simulates a mouse press (0=left mouse button, 1=right mouse button)

map.setSimulatedMouseButtonRelease: simulates a mouse release (0=left mouse button, 1=right mouse button)

Labels API

map.countryLabelsSize: this is the relative size for labels. Controls how much the label can grow to fit the country area.

map.countryLabelsAbsoluteMinimumSize: minimum absolute size for all labels.

map.labelsQuality: specify the quality of the label rendering (Low, Medium, High).

map.showLabelsShadow: toggles label shadowing on/off.

map.countryLabelsColor: color for the country labels. Supports alpha.

map.countryLabelsShadowColor: color for the shadow of country labels. Also supports alpha.

map.labelsFaceToCamera: if set to true, labels will rotate automatically to ensure they can be easily read.

map.countryLabelsEnableAutomaticFade: if set to true, labels will fade in/out depending on screen size.

map.countryLabelsAutoFadeMaxHeight: max height of a label relative to screen height (0..1).

map.countryLabelsAutoFadeMaxHeightFallOff: gradient for the fade out of label.

map.countryLabelsAutoFadeMinHeight: max height of a label relative to screen height (0..1).

map.countryLabelsAutoFadeMinHeightFallOff: gradient for the fade in of label.

Tile System API

map.tileServer: the tile server to use in current session. Can be changed at any time and tiles will be refreshed automatically.

map.tileServerClientId: the account id provided by the tile server (optional, currently used by AerisWeather service for example).

map.tileServerAPIKey: the string to be appended to any tile server URL request. Usually has the form of "apikey=xxxxx" (without quotes).

map.tileServerCopyrightNotice: the required copyright to show in your application according to the tile server terms of use.

map.tileServerLayerTypes: list of layer names separated by commas (used by AerisWeather service).

map.tileTransparentLayer: when enabled, tiles will be rendered using a transparent material.

map.tileMaxAlpha: when tileTransparentLayer is true, this property determines the maximum opacity for the layers. Can be reduced to force transparency on certain opaque layers.

map.tileMaxConcurrentDownloads: maximum number of concurrent tile downloads. Defaults to 10.

map.tileMaxLoadsPerFrame: maximum number of tile animations started per frame. Defaults to 2.

map.tileEnableLocalCache: enables local caching of tiles. Defaults to true.

map.tileMaxLocalCacheSize: size of the local cache in Mb. Defaults to 50.

map.tileQueueLength: current length of the download queue. 0 means all pending tiles have been downloaded. Some downloads may be cancelled if their tiles are no longer visible.

map.tileConcurrentDownloads: current number of concurrent downloads.

map.tileCurrentZoomLevel: current zoom level according to tile system standards (in WPM from 5 to 19).

map.tileWebDownloads: total number of tiles downloads from the web.

map.tileWebDownloadsTotalSize: total size in bytes of tiles downloads from the web.

map.tileCacheLoads: total number of tiles loaded from the local cache.

map.tileCacheLoadsTotalSize: total size in bytes of tiles loaded from the local cache.

map.tileCurrentCacheUsage: total size in bytes of tiles stored in the local cache.

map.tileLastError: last error occurred when downloading tiles (if any). Null is no error so far.

map.tileLastErrorDate: date of the last error when downloading tiles (if any).

map.PurgeTileCache(): removes any downloaded tile in the local cache.

Haxagonal Grid and Path Finding API

map.GenerateGrid(): generated the grid cells. You can call this method is you're not showing the grid.

map.showHexagonalGrid: shows or hides the hexagonal grid over the globe. It will automatically generate the grid cells.

map.hexaGridUseMask: activates the texture mask when rendering the grid. The mask helps define where a cell can be rendered. It uses the value for the blue channel of the texture as a "height" value.

map.hexaGridMask: the mask texture. This texture must be marked as readable.

map.hexaGridMaskThreshold: the minimum height to render a cell.

map.hexaGridDivisions: a value that determines how many divisions are applied to the initial icosahedron.

map.hexaGridColor: the wireframe color of the grid.

map.hexaGridHighlightEnabled: if the user can select a cell with the pointer.

map.hexaGridHighlightColor: the color for the highlight effect.

map.hexaGridHighlightSpeed: the blinking speed of the highlight effect.

map.lastHighlightedCellIndex: the index of the highlighted cell.

map.lastHighlightedCell: the cell object currently highlighted.

map.cells: the array containing all generated cells.

map.GetCellIndex(cell): gets the cell index of a Cell object.

map.GetCellIndex(localPosition): gets the cell index containing a position on the sphere (position given in local space).

map.GetCellAtWorldPos(worldSpacePosition): gets the cell under a world space position.

map.GetCellNeighbours(cellIndex): gets the neighbours of a given cell.

map.GetCellNeighboursIndices(cellIndex): gets the neighbours indices of a given cell.

map.GetCellSharedEdge(cell1, cell2): gets the 2 points of the grid edge shared by cell 1 and 2.

map.SetCellMaterial(cellIndex, material, temporary): assigns a material to a cell. Can be assigned temporarily which will be erased when other color effects occurs on the cell.

map.SetCellColor(cellIndex, color, temporary): assigns a color to a cell. Can be assigned temporarily which will be erased when other color effects occurs on the cell.

map.SetCellColor(list of cells, color, temporary): assigns a color to a list of cells. Can be assigned temporarily which will be erased when other color effects occurs on the cell.

map.SetCellTexture(cellIndex, texture, temporary): assigns a texture to a cell. Can be assigned temporarily which will be erased when other color effects occurs on the cell.

map.pathFindingHeuristicFormula: the heuristic used to estimate the shorted path to a destination cell.

map.pathFindingSearchLimit: the maximum path for any path obtained with FindPath method.

map.FindPath(cell1, cell2): returns the shorted path beween cell1 and cell2.

map.SetCellCanCross / GetCellCanCross(cellIndex): determines if the cell blocks any path when using FindPath.

map.SetCellNeighbourCost / GetCellNeighbourCost (cellIndex, cost): sets or gets the cost for crossing from cell 1 to cell 2 (being cell 2 a neighbour of cell 1).

map.ClearCell / ClearCells: removes any color or texture from a cell.

map.DestroyCell(cellIndex): removes any color or texture from a cell and also removes the cached geometry.

map.ToggleCell(cellIndex, visible): makes a cell visible or invisible.

map.HideCell / ShowCell(cellIndex): hides/shows any cell.

map.GetWorldSpaceCenter(cellIndex): gets the position in world space coordinates of the cell center.

map.FlyToCell(cellIndex): navigates towards the cell.

map.GetCells(region): returns a list of cells contained by a region.

Fog of War API

map.showFogOfWar: shows or hides the fog of war layer.

map.fogOfWarResolution: a value that translates into the internal texture for storing the transparency of the fog of war around the globe. The texture size is 2^this value, so a value of 10 means a texture size of 1024x512 (equirectangular textures have a 2x1 ratio).

map.fogOfWarColor1/2: colors for the fog of war.

map.fogOfWarColor1/2: colors for the fog of war.

map.fogOfWarElevation: thickness for the fog of war layer.

map.fogOfWarClear(alpha): fills fog of war with alpha value (0 = no fog, 1 = full opaque fog)

map.SetFogOfWarAlpha(spherePos | region | country | province | cell, alpha, radius, strength): erases or paints with fog a custom position with given radius and strength of pen.

Markers API

map.AddText: adds a label at any position on the globe with custom font, color, size and style.

map.AddMarker: adds a marker on the globe. It can be a gameobject, a circle, projected circle or quad. Returns a reference to the marker.

map.AddLine: draws a line on the globe between two points with custom width, color, drawing duration and fade out effects. Returns a reference to the LineMarkerAnimator component.

map.AddPolygon3D: draws a polygon over the globe with custom color and optional fill color. Receives a list of latitude/longitude coordinates.

map.ClearMarker: destroy a marker created with AddMarker.

map.ClearMarkers: destroy all markers.

map.ClearLineMarker: destroy a line added to the globe and returned when calling AddLine function.

map.ClearLineMarkers: destroy all lines.

Loading/Saving Data

Often you will need to modify data in runtime and need to store those modifications somewhere.
The API provides you the following methods to get / set geodata information on the fly.
Check demo scene “15 Custom Attributes” for an example about storing/reading custom country attributes.
When using Set**GeoData methods, call **Redraw**() to update the representation.

Countries

- **GetCountryGeoData**(): returns all country geodata information in a packed string with same format as the geodata files.
- **SetCountryGeoData**(string s): sets the country geodata from a packed string.
- **GetCountriesAttributes**(): returns all custom attributes of all countries.
- **SetCountriesAttributes**(string s): sets all countries attributes.

Provinces

- **GetProvinceGeoData**() / **SetProvincesGeoData**(): similar to above methods for countries, but for provinces: gets / sets provines geodata.
- **GetProvincesAttributes**() / **SetProvincesAttributes**(string s): gets / sets all custom attributes of all provinces.

Cities

- **GetCitiesGeoData**() / **SetCitiesGeoData**(string s): same for cities.
- **GetCitiesAttributes**() / **SetCitiesAttributes**(string s): gets / sets cities attributes.

Mount Points

- **GetMountPointsGeoData**() / **SetMountPointsGeoData**(): same for mount points.
- **GetMountPointsAttributes**() / **SetMountPointsAttributes**(string s): gets / sets mount points attributes.

Other Methods

map.ToggleContinentSurface(name, visible, color): colorize all countries belonging to speficied continent with color provided or hide its surface (if visible = false).

map.HideContinentSurface(name): un-colorize / hide specified continent.

map.Show / map.Hide: faster alternative to toggling gameobject for showing/hiding globe.

Events

In addition to above methods you can listen to the following events (check out the Demo.cs script for sample code):

OnClick(Vector3 spherePosition, int mouseButtonIndex);
Occurs when user clicks on the globewith left button. You can use the API Conversion.GetLatLonFromSpherePoint(spherePosition) to retrieve the latitude/longitude coordinates of the clicked position (see demo scene 1 code example).

OnDrag(Vector3 spherePosition);
Occurs when user moves the mouse over the map while holding left button.

OnCityEnter(int cityIndex);
Occurs when cursor hits a city.

OnCityExit(int cityIndex);
Occurs when cursor leaves a city.

OnCityPointerDown(int cityIndex);
Occurs when user starts pressing on a city with the mouse/pointer.

OnCityPointerUp(int cityIndex);
Occurs when user release button on a city with the mouse/pointer.

OnCityClick(int cityIndex);
Occurs when user clicks over a city. If user drags the globe or it doesn't click fast, this event won't trigger.

OnCountryBeforeEnter(int countryIndex, int regionIndex, ref bool ignoreCountry);
Occurs when cursor is about to enter a country. You can set ignoreCountry to boolean to ignore this country from highlighthing.

OnCountryEnter(int countryIndex, int regionIndex);
Occurs when cursor is about to enter a country.

OnCountryExit(int countryIndex, int regionIndex);
Occurs when cursor leaves a country.

OnCountryPointerDown(int countryIndex, int regionIndex);
Occurs when user starts pressing on a country with the mouse/pointer.

OnCountryPointerUp(int countryIndex, int regionIndex);
Occurs when user release utton on a country with the mouse/pointer.

OnCountryClick(int countryIndex, int regionIndex);
Occurs when user clicks over a country. If user drags the globe or it doesn't click fast, this event won't trigger.

OnProvinceBeforeEnter(int provinceIndex, int regionIndex, ref bool ignoreProvince);
Occurs when cursor is about to enter a province. You can set ignoreProvince to true so this province won't be highlight.

OnProvinceEnter(int provinceIndex, int regionIndex);
Occurs when cursor enters a province.

OnProvinceExit(int provinceIndex, int regionIndex);
Occurs when cursor leaves a province.

OnProvincePointerDown(int provinceIndex, int regionIndex);
Occurs when user starts pressing on a province with the mouse/pointer.

OnProvincePointerUp(int provinceIndex, int regionIndex);
Occurs when user releases button on a province with the mouse/pointer.

OnProvinceClick(int countryIndex, int regionIndex);
Occurs when user clicks over a province. If user drags the globe or it doesn't click fast, this event won't trigger.

OnCellEnter(int cellIndex)
Occurs when pointer enters a cell of the hexagonal grid.

OnCellClick(int cellIndex)
Occurs when user clicks a cell of the hexagonal grid.

OnCellExit(int cellIndex)
Occurs when pointer abandon a cell of the hexagonal grid.

OnCellCross(int cellIndex)
Occurs whenthe path finding evaluates a cell. It must return the cost for crossing the cell. If not used, it's assumed all cells cost 1 point.

OnFlyStart(Vector3 spherePosition)
Fired when a FlyTo operation has started.

OnFlyEnd(Vector3 spherePosition)
Fired when a FlyTo operation has finished and it has reached destination.

bool **OnTileRequest**(int zoomLevel, int x, int y, out Texture2D texture, out string error)
Use this event to provide your own tile textures as they're needed.
If your event function returns true, the texture will be used for the tile and no other sources will be tried (provide an error string if the texture cannot be provided and you don't want Globe to try other sources, otherwise set error to null).
If the event function returns false, any texture wil be ignored.

string **OnTileURLRequest**(string url, TILE_SERVER server, int zoomLevel, int x, int y)
Use this event to change the tile url on the fly. The event handler received the proposed url but you can return any custom url using the parameters provided of server, zoom level, tile x and y. Or just return the url provided as a result if you want to keep that url.

OnTileRecomputed (List<TileInfo> visibleTiles)
This event is called each time tiles visibility is computed. It receives a list of currently visible tiles.

OnRaycast ()
This event is called when the asset is performing a raycast to determiner cursor position on the globe. You can hook your own event delegate to provide custom ray object to be used. For example:

```
map.OnRaycast += () => new Ray(origin, direction);
```

Geodata file format description

Data for countries, provinces and cities are stored in text files located in WorldMapPoliticalMapGlobeEdition/Resources/Geodata folder.

- countries10.txt: data for country frontiers in high resolution
- countries110.txt: data for country frontiers in low resolution
- provinces10.txt: data for province borders (always in high resolution)
- cities10.txt: data for cities.

All data is packed in string fields using separators. JSON, although more human-friendly format, is not used to reduce the file size and optimize the loading time (especially for mobile devices).

Country file format

1. Contains a list of countries separated by |.
   Example: COUNTRY0|COUNTRY1|COUNTRY2…)

2. Each COUNTRY entry is a series of fields separated by $:
   COUNTRY = {NAME $ CONTINENT $ REGIONS $ HIDDEN $ FIPS 10.4 CODE $ ISO A2 $ ISO A3 $ ISO N3 $ LABEL VISIBILITY}

   Name = name of the country (as displayed on the map)
   Continent = continent name (useful to group countries)
   Hidden = 0 (visible) or 1 (invisible)
   FIPS / ISO codes = standard id codes for country
   Label visibility = 1 (visible) or 0 (invisible)

3. REGIONS is a list of polygons, separated by *:
   REGIONS = REGION0*REGION1*REGION2…

4. Each REGION is a list of latitude/longitude coordinates in the format:
   REGION = lat0 , lon0 ; lat1 , lon1 ; lat2, lon 2 ; …

5. The stored values of latitude and longitude are multiplied by 5.000.000 and rounded.

The method SetCountryGeoData() method in WPMCountries.cs is responsible of loading and extracting the file data.

Province file format

1. Contains a list of provinces separated by |.
   Example: PROVINCE0|PROVINCE1|PROVINCE2…)

2. Each PROVINCE entry is a series of fields separated by $:
   PROVINCE = {NAME $ COUNTRY_NAME $ REGIONS }

   Name = name of the country (as displayed on the map)
   Country_Name = the name of the country to which the province belongs to

3. REGIONS is a list of polygons, separated by *:
   REGIONS = REGION0*REGION1*REGION2…

4. Each REGION is a list of latitude/longitude coordinates in the format:
   REGION = lat0 , lon0 ; lat1 , lon1 ; lat2, lon 2 ; …

5. The stored values of latitude and longitude are multiplied by 5.000.000 and rounded.

The method SetProvincesGeoData() and ReadProvincePackedString() methods in WPMProvinces.cs are responsible of loading and extracting the file data.

City file format

1. Contains a list of cities separated by |.
   Example: CITY0|CITY1|CITY2…)

2. Each CITY entry is a series of fields separated by $:
   CITY = {NAME $ PROVINCE_NAME $ COUNTRY_NAME $ POPULATION $ X $ Y $ Z $ CLASS }

   Name = name of the city
   Province_Name = the name of the province
   Country_Name = the name of the country to which the province belongs to
   X, Y, Z = sphere coordinate of the city
   Population = metropolitan population
   Class = 1 (regular city), 2 (region capital) or 4 (country capital)

The method SetCityGeoData() method in WPMCities.cs is responsible of loading and extracting the file data.

Custom Attributes (demo scene 15)

Countries, provinces, cities and mount points metadata can be extended with your own set of attribures. We call this feature “Custom Attributes”.

Custom Attributes are stored in separated files in the same Geodata folder which contains countries, provinces and cities borders data. The default file names are “**countryAttrib”**, “**provincesAttrib”** and “**citiesAttrib”**.

Assigning and retrieving your own attributes

Demo scene #15 covers all uses cases regarding Custom Attributes. For example, to add a few attributes to a country, like Canada you would do:

```
Country canada = map.GetCountry("Canada");

 // Add language as a custom attribute
canada.attrib["Language"] = "French";

// Add the date of British North America Act, 1867
canada.attrib["ConstitutionDate"] = new DateTime(1867, 7, 1);

// Add the land area in km2
canada.attrib["AreaKm2"] = 9984670;
```

As you can see, a new property called “attrib” has been added to each country (same for provinces and cities). The attrib property is in fact a JSONObject capable or parsing and printing jSON-compliant data. It supports basic types like numbers and string, also dates and booleans, arrays and other JSON objects.

The attrib property is indexed so you can access the top-level fields of the JSONObject by its name or index number. To retrieve the values of the custom attributes added above, you’d do:

```
string language = canada.attrib["Language"]);
DateTime constitutionDate = canada.attrib["ConstitutionDate"].d; // Note the use of .d to force cast the internal number representation to DateTime
float countryArea = canada.attrib["AreaKm2"]);
```

Filtering by Custom Attributes

You can also launch a filtered seach of countries that match a predicate using Custom Attributes:

```
List<Country> results = new List<Country>();
countries = map.GetCountries(
        (attrib) => "French".Equals(attrib["Language"]) && attrib["AreaKm2"] > 1000000,
        results);
          );
Int matchesFound = results.Count);
```

The expression in bold is the predicate, expressed in lambda syntax. The GetCountries method as been overloaded so when you pass a lambda expression as above, it will iterate through all countries and pass its attrib JSONObject to your code, so you can interrogate it and return true if it matches your condition or not. In the example above, the lambda expression tests if the attributes passed contains a field Language which equals to "French" and also checks if the attribute "AreaKm2" is greater than 1000000.
Please note that you should check if the attributes list contains the field otherwise it will produce null exceptions in your predicate.

Importing / Exporting to JSON

The attrib object can parse a JSON-formatted string:

canada.attrib = new JSONObject(json); *// Import from raw jSON string*
int keyCount = canada.attrib.keys.Count; // Get the number of fields

And you can export current attributes of one country back to a JSON-formatted string:

string json = canada.attrib.Print();

To get the JSON-formatted string including all countries, you can call:

string jsonCountries = map.GetCountriesAttributes (true); // the true parameter will make the JSON string "pretty" (ie. adds tabs and end-of-line characters).

To get the JSON-formatted string including all countries, you can call:

string jsonCountries = map.GetCountriesAttributes (true); // the true parameter will make the JSON string "pretty" (ie. adds tabs and end-of-line characters).

To read all countries attributes from a custom string variable, call SetCountriesAttributes:

map.SetCountriesAttributes(jsonCountries);

The above method is called when you set the **countryAttributeFile** property.

Managing Custom Attributes

Custom Attributes added or modified will be lost if not persisted to file.

Using the Map Editor, you can manage custom attributes to countries, provinces, cities and mount points and save them to the Geodata folder (click "Save" button in the Map after making any change). *Beware that running your application without Saving changes will result in losing your changes!*

If you want to make changes to attributes and save them, you can get the JSON-formatted string as seen above for all attributes of all countries, provinces and ciites, and store it in your own database, file system, as user prefs, …

You can also have different custom attributes files. To reload the attributes file, just set the property countryAttributesFile (or provinceAttributesFile, cityAttributesFile) to a different name. The asset will try to find and load the data in the new file*. This file needs to be located inside Geodata folder.*

Additional Components

World Map Calculator

This component is useful to:

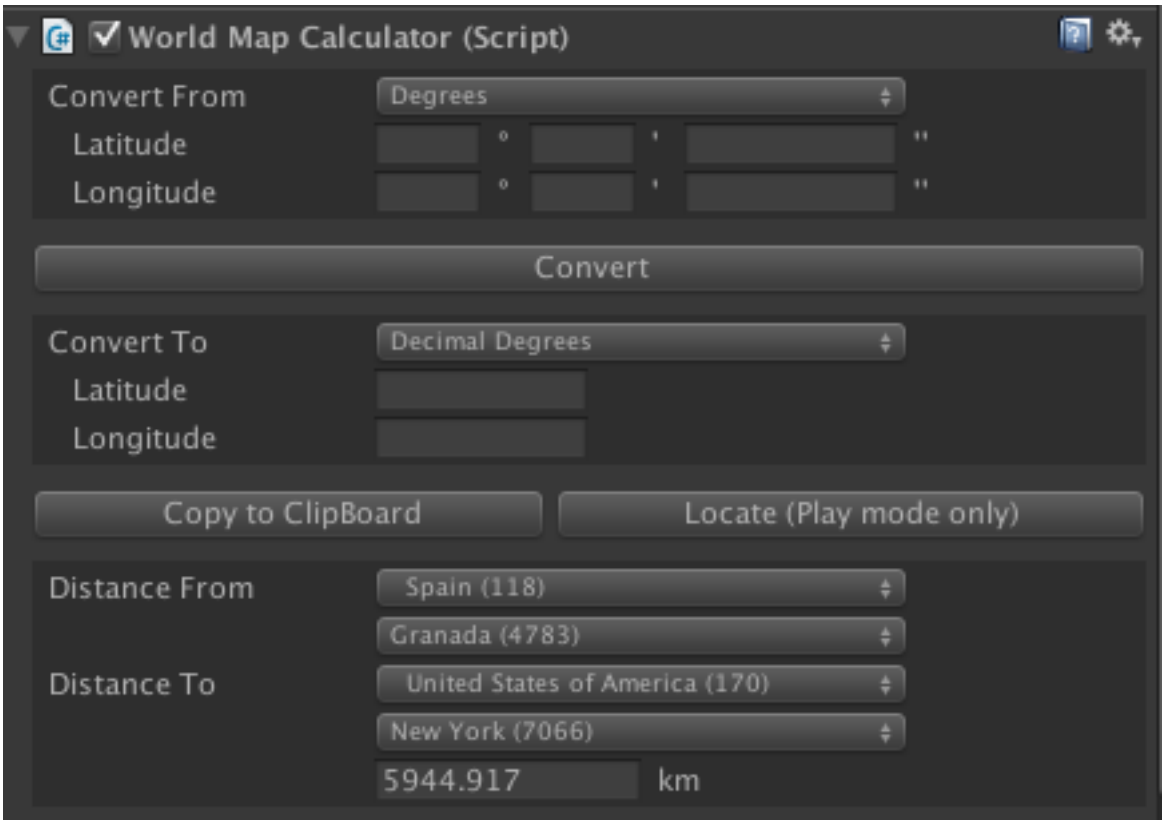
1. Convert units from one coordinate system to another (for instance from plane coordinates to degrees and viceversa).
2. Calculate the distance between cities.

You may also use this component to capture the current cursor coordinates and convert them to other coordinate system.

You may enable this component in two ways:

- From the Editor, clicking on the “Open Calculator” button at the bottom of the World Map Globe inspector.
- From code, using any of its API through **map.calc** accessor. The first time you use its API, it will automatically add the component to the globe gameObject.

On the Inspector you will see the following custom editor:



Converting coordinates from code

NEW! The asset provides a new **Conversion** static class which exposes many methods to quickly convert between lat/lon and map position which don't require the calculator component.

Example:

Vector3 localPos = Conversion.GetSpherePointFromLatLon(lat, lon);

In addition to the Conversion static class, you may use the calculator component from code. You may access the conversion API of this componet from code through **map.calc** property. The conversion task involves 3 steps:

1. Specify the source unit (eg. "**map.calc.fromUnit = UNIT_TYPE.DecimalDegrees**").
2. Assign the source parameters (eg. "**map.calc.fromLatDec = -15.281**")
3. Call **map.calc.Convert()** method.
4. Obtain the results from the fields **map.calc.to*** (eg. "**map.calc.toLatDegrees**", "**map.calc.toLatMinutes**", …).

Note that the conversion will provide results for decimal degrees, degrees and spherical coordinates. You don't have to specify the destination unit (that's only for the inspector window, in the API the conversion is done for the 3 types).

To convert from Decimal Degrees to any other unit you use:

```
map.calc.fromUnit = UNIT_TYPE.DecimalDegrees
map.calc.fromLatDec = <decimal degree for latitude>
map.calc.fromLonDec = <decimal degree for longitude>
map.calc.Convert()
```

To convert from Degrees, you do:

```
map.calc.fromUnit = UNIT_TYPE.Degrees
map.calc.fromLatDegrees = <degree for latitude>
map.calc.fromLatMinutes = <minutes for latitude>
map.calc.fromLatSeconds = <seconds for latitude>
map.calc.fromLonDegrees = <degree for longitude>
map.calc.fromLonMinutes = <minutes for longitude >
map.calc.fromLonSeconds = <seconds for longitude >
map.calc.Convert()
```

And finally, to convert from X, Y, Z (normalized) you use:

```
map.calc.fromUnit = UNIT_TYPE.SphereCoordinates
map.calc.fromX = <X position in local sphere coordinates >
map.calc.fromY = <Y position in local sphere coordinates >
map.calc.fromZ = <Z position in local sphere coordinates>
map.calc.Convert()
```

The results will be stored in (you pick what you need):

map.calc.toLatDec = <decimal degree for latitude>
map.calc.toLonDec = <decimal degree for longitude>
map.calc.toLatDegrees = <degree for latitude>
map.calc.toLatMinutes = <minutes for latitude>
map.calc.toLatSeconds = <seconds for latitude>
map.calc.toLonDegrees = <degree for longitude>
map.calc.toLonMinutes = <minutes for longitude >
map.calc.toLonSeconds = <seconds for longitude >
map.calc.toX = <X position in local sphere coordinates >
map.calc.toY = <Y position in local sphere coordinates >
map.calc.toZ = <Z position in local sphere coordinates>

You may also use the property **map.calc.captureCursor = true**, and that will continuously convert the current coordinates of the cursor (mouse) until it's set to false or you right-click the game window.

Using the distance calculator from code

The component includes the following two APIs to calculate the distances in meters between two coordinates (latitude/longitude) or two cities of the current selected catalogue.

map.calc.Distance(float latDec1, float lonDec1, float latDec2, float lonDec2)

map.calc.Distance(City city1, City city2)

map.calc.Distance(Vector3 spherePosition1, Vector3 spherePosition2)

Using the Conversion static class

This class provides conversion methods that you can use directly without having an instance of the globe in the scene. It's the new way to convert coordinates and we recommend you to use it instead of the map.calc component:

Vector2 GetUVFromLatLon(float lat, float lon)
Returns the texture coordinates (UV) of a latitude/longitude assuming the texture fills the entire globe.

Vector2 GetLatLonFromUV(Vector2 uv)
Returns the latitude/longitude from a UV coordinate.

Vector2 GetUVFromSpherePoint(Vector3 p)
Returns the texture coordinates (UV) from a point on the globe sphere.

Vector3 GetSpherePointFromLatLon(double lat, double lon, double altitude = 0)
Vector3 GetSpherePointFromLatLon(float lat, float lon, float altitude = 0f)
Vector3 GetSpherePointFromLatLon(Vector2 latLon, float altitude = 0)
Returns the sphere local position from a latitude/longitude and optional altitude.

GetLatLonFromSpherePoint(Vector3 p, out double lat, out double lon)
GetLatLonFromSpherePoint(Vector3 p, out float lat, out float lon)
GetLatLonFromSpherePoint(Vector3 p, out Vector2 latLon)
Vector2 GetLatLonFromSpherePoint(Vector3 p)
Returns the latitude/longitude from a local sphere position

float Distance(Vector3 position1, Vector3 position2)
float Distance(float latDec1, float lonDec1, float latDec2, float lonDec2)
Returns the distance in meters from two positions.

GetTileFromLatLon(int zoomLevel, float lat, float lon, out int xtile, out int ytile)
Returns the x/y indices of a tile from a zoom level, latitude and longitude.

Vector2 GetLatLonFromTile(float x, float y, int zoomLevel)
Returns the latitude/longitude of a tile.

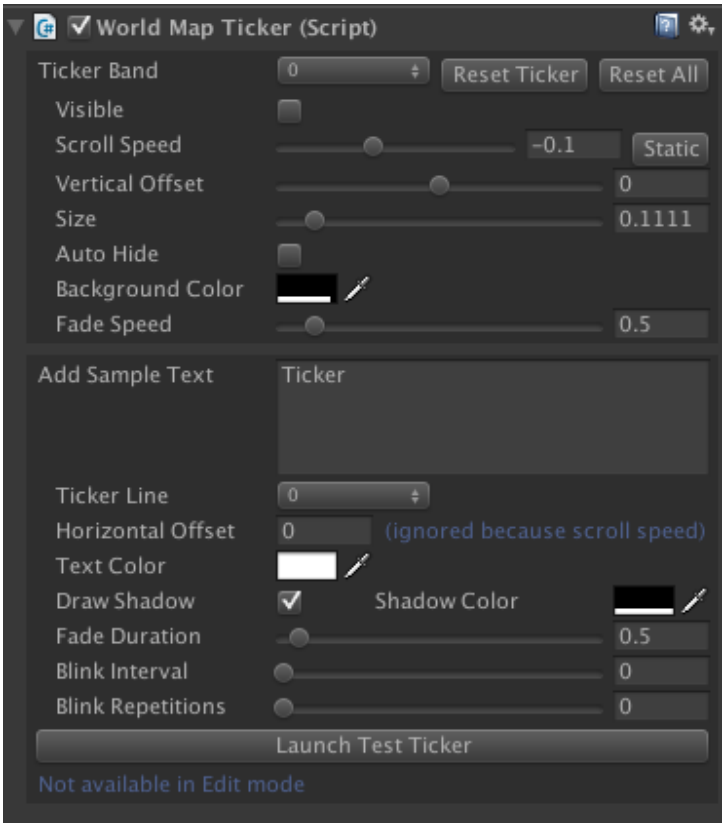
World Map Ticker Component

Use this component to show impact banners over the map. You can show different banners, each one with different look and effects. Also you can add any number of texts to any banner, and they will simply queue (if scrolling is enabled).

Similarly to the World Map Calculator component, you may enable this component in two ways:

- From the Editor, clicking on the "Open Ticker" button at the bottom of the World Map Globe inspector.
- From code, using any of its API through **map.ticker** accessor. The first time you use its API, it will automatically add the component to the globe gameObject.

On the Inspector you will see the following custom editor:



The top half of the inspector corresponds to the Ticker Bands configurator. You may customize the look&feel of the 9 available ticker bands (this number could be incremented if needed though). Notes:

- Ticker bands are where the ticker texts (second half of the inspector) scrolls or appears.
- A ticker band can be of two types: scrollable or static. You make a ticker band static setting its scroll speed to zero.
- Auto-hide will make the ticker band invisible when there're no remaining texts on the band.

- The fade speed controls how quickly should the band appear/disappear. Set it to zero to disable the fade effect.

It's important to note that everything you change on the inspector can be done using the API (more below).

In the second half of the inspector you can configure and create a sample ticker text. Notes:

3. Horizontal offset allows you to control the horizontal position of the text (0 equals to zero longitude, being the range -0.5 to 0.5).
4. Setting fade duration to zero will disable fading effect.
5. Setting blink interval to zero will disable blinking and setting repetitions to zero will make the text blink forever.

The API can be accessed through map.ticker property and exposes the following methods/fields:

map.ticker.NUM_TICKERS: number of available bands (slots).

map.ticker.tickerBands: array with the ticker bands objects. Modifying any of its properties has effect immediately.

map.ticker.GetTickerTextCount(): returns the number of ticker texts currently on the scene. When a ticker text scrolls outside the ticker band it's removed so it helps to determine if the ticker bands are empty.

map.ticker.GetTickerTextCount(tickerBandIndex): same but for one specific ticker band.

**map.ticker.GetTickerBandsActiveCount(**): returns the number of active (visible) ticker bands.

map.ticker.ResetTickerBands(): will reset all ticker bands to their default values and removes any ticker text they contain.

map.ticker.ResetTickerBand(tickerBandIndex): same but for an specific ticker band.

map.ticker.AddTickerText(tickerText object): adds one ticker text object to a ticker band. The ticker text object contains all the neccesary information.

The demo.cs script used in the Demo scene contains the following code showing how to use the API:

```
// Sample code to show how tickers work
void TickerSample() {
   map.ticker.ResetTickerBands();

   // Configure 1st ticker band: a red band in the northern hemisphere
   TickerBand tickerBand = map.ticker.tickerBands[0];
   tickerBand.verticalOffset = 0.2f;
   tickerBand.backgroundColor = new Color(1,0,0,0.9f);
   tickerBand.scrollSpeed = 0;    // static band
```

```
    tickerBand.visible = true;
    tickerBand.autoHide = true;

    // Prepare a static, blinking, text for the red band
    TickerText tickerText = new TickerText(0, "WARNING!!");
    tickerText.textColor = Color.yellow;
    tickerText.blinkInterval = 0.2f;
    tickerText.horizontalOffset = 0.1f;
    tickerText.duration = 10.0f;

    // Draw it!
    map.ticker.AddTickerText(tickerText);

    // Configure second ticker band (below the red band)
    tickerBand = map.ticker.tickerBands[1];
    tickerBand.verticalOffset = 0.1f;
    tickerBand.verticalSize = 0.05f;
    tickerBand.backgroundColor = new Color(0,0,1,0.9f);
    tickerBand.visible = true;
    tickerBand.autoHide = true;

    // Prepare a ticker text
    tickerText = new TickerText(1, "INCOMING MISSLE!!");
    tickerText.textColor = Color.white;
    tickerText.horizontalOffsetAutomatic = true;

    // Draw it!
    map.ticker.AddTickerText(tickerText);
}
```

World Map Decorator

This component is used to decorate parts of the map. Current decorator version supports:

- ✓ Customizing the label of a country
- ✓ Colorize a country with a custom color
- ✓ Assign a texture to a country, with scale, offset and rotation options.

You may use this component in two ways:

- From the Editor, clicking on the "Open Decorator" button at the bottom of the World Map Globe inspector.
- From code, using any of its API through **map.decorator** accessor. The first time you use its API, it will automatically add the component to the globe gameObject.

In the Editor, this component has this interface (on the right):

Important!
Decorators also expose an API but it's better to use the public existing API at runtime to colorize/texture countries and provinces (see ToggleCountrySurface and similar functions).
Use decorators preferably at design time to customize the look of the map.

The API of this component has several methods but the most important are:

map.decorator.SetCountryDecorator(int groupIndex, string countryName, CountryDecorator decorator)

This will assign a decorator to specified country. Decorators are objects that contains customization options and belong to one of the existing groups. This way you can enable/disable a group and all decorators of that group will be enabled/disabled at once (for instance, you may group several countries in the same group).

map.decorator.RemoveCountryDecorator(int groupIndex, string countryName)

This method will remove a decorator from the group and its effects will be removed.

A decorator object has the following fields:

- **countryName**: the name of the country to be decorated. It will be assigned automatically when using SetCountryDecorator method.
- **customLabel**: leave it as "" to preserve current country label.
- **isColorized**: if the country is colorized.
- **fillColor**: the colorizing color.
- **labelOverridesColor**: if the color of the label is specified.
- **labelColor**: the color of the label.
- **labelVisible**: sets the label visible or hidden.
- **labelOffset**: specifies a manual offset for the label with respect to the country center. A default value of (0,0) will make the label to automatically shift if needed.
- **labelRotation**: manual rotation for the label in degrees (0-359). If set to zero, the label can be automatically rotated by the system.
- **texture**: the texture to assign to the country.
- **textureScale**, **textureOffset** and **textureRotation** allows to tweak how the texture is mapped to the surface.

World Map Editor Component

Use this component to modify the provided maps interactively from Unity Editor (it doesn't work in play mode). To open the Map Editor, click on the "Open Editor" button at the bottom of the World Map inspector.

On the Inspector you will see the following custom editor:

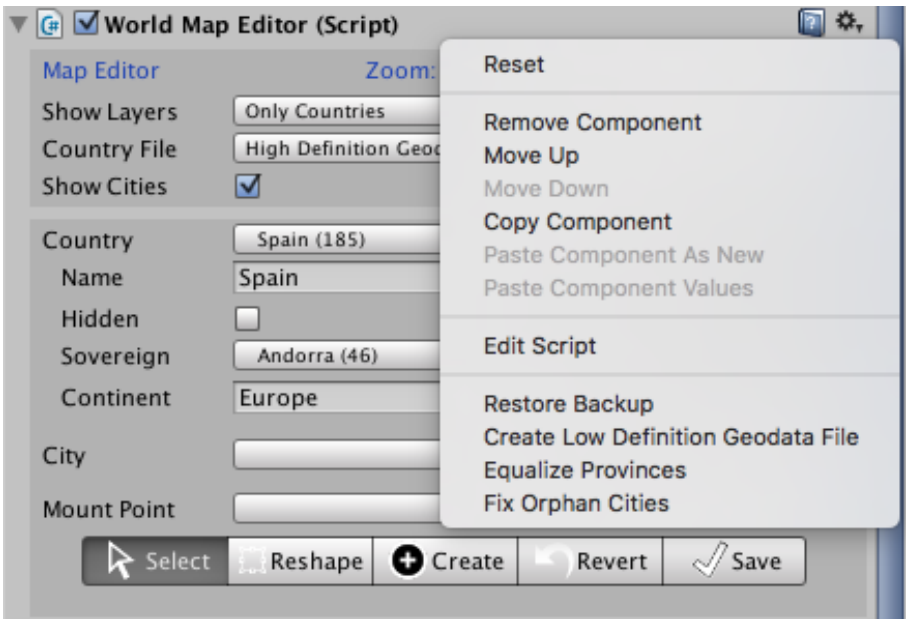
Description:

- **Show Layers**: choose whether to visualize countries or countries + provinces.hich layer to modify.
- **Country File**: choose which file to edit:
  - Low-definition geodata file (110m:1 scale)
  - High-definition geodata file (30m:1 scale)
- **Country:** the currently selected country. You can change its name or "sell" it to another country clicking on transfer.
- **Province/State:** the currently selected province/state if provinces are visible (see Show Layers above). As with countries, you can change the province's name ore ven transfer it ot another country.
- **City:** the currently selected city.

Main toolbar

- **Select:** allows you to select any country, province or city in the Scene view. Just click over the map!
- **Reshape:** once you have either a country, province or city selected, you can apply modifications. These modifications are located under the Reshape mode (see below).
- **Create:** enable the creation of cities, provinces or countries.
- **Revert**: will discard changes and reload data from current files (in Resources/Geodata folder).
- **Save**: will save changes to files in Resources/Geodata foler.

If you click the gear icon on the inspector title bar, you will see 2 additional options:



- **Restore Backup**: the first time you save changes to disk, a backup of the original geodata files will be performed. The backed up files are located in Backup folder inside the main asset folder. You may manually replace the contents of the Resources/Geodata folder by the Backup contents manually as well. This option do that for you.

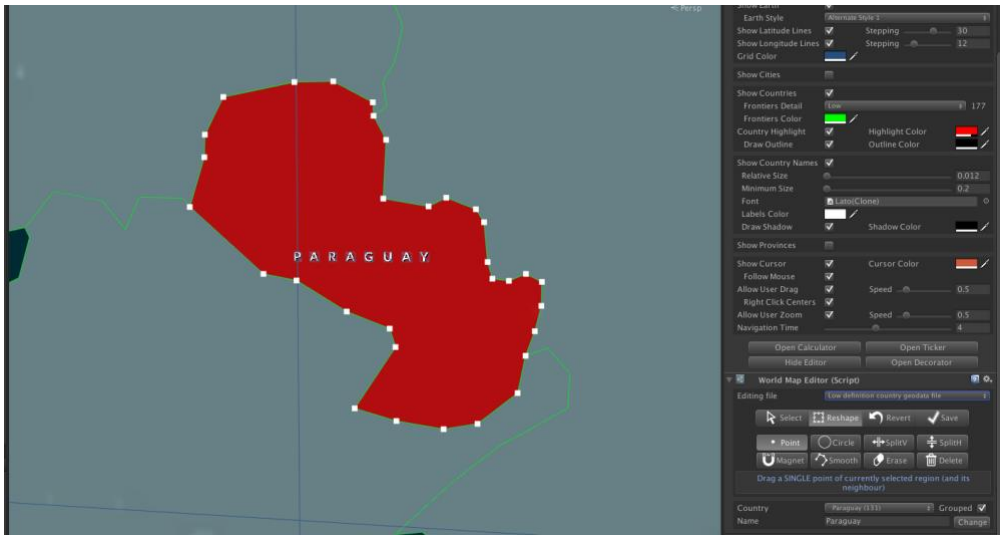
- **Create Low Definition Geodata File:** this option is only available when the high-definition geodata file is active. It will automatically create a simplistic and reduced version (in terms of points) and replace the low-definition geodata file. This is useful only if you use the high-definition geodata file. If you only use the low-definition geodata file, then you may just change this map alone.

- **Equalize Provinces:** this option allows you to merge provinces in each country so the resulting number is in the given range. Useful for generating simplified provinces map.

- **Fix Orphan Cities:** this option will search any city without province or country assigned and fix it. It will assign the country that surrounds the city (or the nearest one). It will also take the provine surrounding the city and assign to it. Once you execute this option, remember to save!

Reshaping options

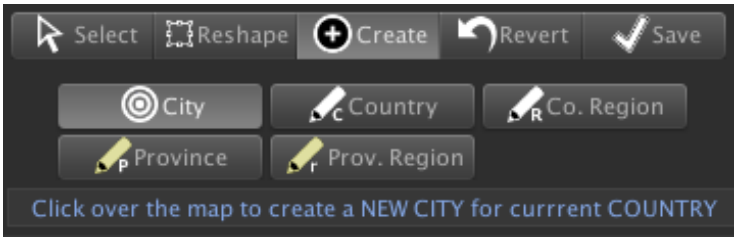
When you select a country, the Reshape main option will show the following tools:



- **Point tool**: will allow you to move one point at a time. Note that the corresponding point of the neighbour will also be moved along (if it exits). This way the frontier between two regions is easily modified in one step, avoiding gaps between adjacent regions.
- **Circle tool:** similar to the point tool but affects all points inside a circle. The width of the circle can be changed in the inspector. Note that points nearer to the center of the circle will move faster than the farther ones unless you check the “Constant Move” option below.
- **SplitV**: will split vertically the country or region. The splitted region will form a new country/province with name “New X” (X = original country name)
- **SplitH**: same but horizontally.
- **Magnet**: this useful works by clicking repeatedly over a group of points that belong to different regions. It will try to make them to join fixing the frontier. Note that there must be a sufficient number of free points so they can be fused. You can toggle on the option “Agressive Mode” which will move all points in the circle to the nearest points of other region and also will remove duplicates.
- **Smooth**: will create new points around the border of the selected region.
- **Erase**: will remove the points inside the selection circle.
- **Delete**: will delete selected region or if there’re no more regiosn in the current country or province, this will remove the entity completely (it disappear from the country /province array).

Create options

In “Create mode” you can add new cities, provinces or countries to the map:



Note that a country is comprissed of one or more regions. Many countries have only one region, but those with islands or colonies have more than one. So you can add new regions to the selected country or create a new country. When you create a new country, the editor automatically creates the first / main region.

Also note that the main region of a country is the biggest one in terms of euclidean area.
Provinces have also regions, and can have more than one.

Editing Tips

This section contains useful tips for a correct usage of the Map Editor. Please read carefully before contacting us with any issue.

1. Before start making changes, determine if you need the high-definition frontiers file or not. Current release of the asset is quite optimized so unless you have specific low-end platform requirements we'd recommend you to use the high definition mode.
   Of course if you don't need such detail in your project, then you can just work with the low-definition file. Note that the high-def and low-def files are different. That means that changes to one file will not affect the other. This may duplicate your job, so it's important to decide if you want to modify both maps or only the low-def map.

2. We strongly recommend using the editor with a globe scale of at least 1000. You may switch between normal scale (1,1,1) and "editing scale" (1000,1000,1000) pressing the "Toggle Zoom" button in the editor inspector.

3. When you decide to modify the high-definition file, you will want to use the command "Create Low Definition Geodata File" from the gear command, and review the low-def map afterwards.

4. If you make any mistake using the Point/Circle tool, you can Undo (Control/Command + Z or Undo command from the Edit menu). Please note that undo is not supported for complex operations, like creating a new country, deleting or transfering regions. So save often!

5. You can use the Revert button and this will reload the geodata files from disk (changes in memory wiil be lost) – so if you saved before performing a complex operation and it went "bad" you must click "Revert" (which in fact acts as an Undo except for you have to save first!).

6. If you modified the geodata files in Resources/Geodata and want to recover original files, you can use the Restore Backup command from the gear icon, or manually replace the Resourcess/Geodata files with those in the Backup folder. As a last resort you may replace currnet files with the originals in the asset .unitypackage.

7. Starting V4, there're two new buttons in the Editor inspector:

   - Redraw: this will delete any children from the globe and redraws every object/layer. This is a reset button for the scene but won't discard any change to the frontiers.
   - Sanitize: this option can be useful if for any reason the frontiers of a country goes wrong. This option will check for self-crossing segments in the polygon and will remove them. It can't fix every problem, but most of the time you don't see a country correctly filled in color is due to a self-crossing polygon.

8. Remember to visit us at kronnect.com for new updates and additional quesitons/support. Thanks.

World Flags and Weather Symbols

This package is available as a separate purchase and includes +270 vector and raster images of country flags and weather symbols:

For more information, please consult the Asset Store page:
https://www.assetstore.unity3d.com/#!/content/69010

Once imported into the project, the names of the flag texture files equal to the country names used in our map assets so you can add flag icons or texture countries with their flag with minimal effort.

The code to texture the country surface with its flag would be:

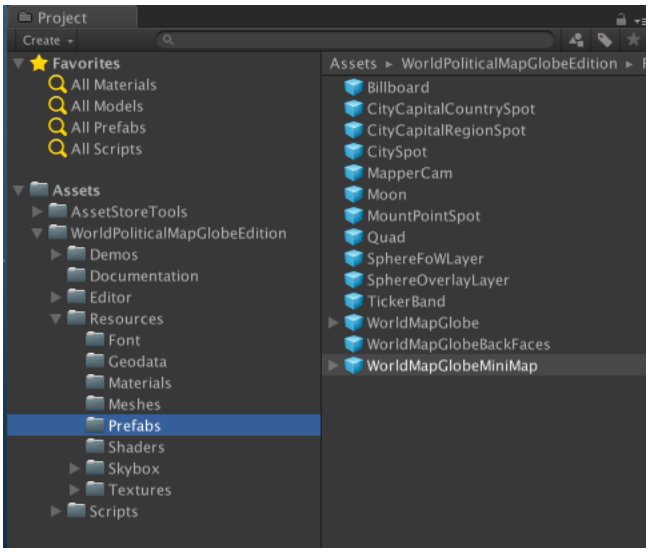
```
// Get reference to the API
WorldMapGlobe map = WorldMapGlobe.instance;


// Choose a country
string countryName = “China”;


// Load texture for the country
Texture2D flagTexture = Resources.Load<Texture2D> ("Flags/png/" + countryName);


// Apply texture to the country main region (ignore islands)
int countryIndex = map.GetCountryIndex(countryName);
map.ToggleCountryMainRegionSurface(countryIndex, true, flagTexture);
```

Mini Map

The mini map is a Canvas UI element that enables quick world navigation by just cliking on a flat 2D map view.

To add a mini-map to your scene just locate and drag&drop the WorldMapGlobeMiniMap prefab from Resources/Prefabs to your scene.

Feel free to position/scale the minimap on the screen as needed.



Once added, expand the WorldMapGlobeMiniMap gameobject and select MiniMap Image to access some options: