WORLD POLITICAL MAP
2D EDITION
with Map Editor

Contents

Introduction

Thank you for purchasing!

World Political Map – 2D 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 +7100 most important cities in the world.
- Colorize and also highlight the regions of countries and provinces/states as mouse hovers them.
- Automatically draws country labels, with placement options.
- Per country texture support!
- Viewport rendering targets (supports cropping)
- Quickly locate and center any country, city, province or custom location.
- Imaginary lines: draw custom latitude, longitude and cursor lines.
- 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 (6 styles included)…
- Works on Android and iOS.
- Comprehensive API and extra components: **Calculator**, **Tickers**, **Decorator** and **Map Editor**.
- Dedicated and responsive support forum.

You can use this asset to show or allow the player to choose a location in your strategy game or application, in mission briefings, interactive reports, statistical and educational software, Earth hud locator, etc.

QuickStart and Demo Scene

1. Import the asset into your project or create an empty project.
2. Open any demo scene located in "Demos" folder.
3. Run and experiment with the demonstration.

The Demo scene contains a WorldMap2D 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.
Visit our Support Forum for usage tips and access to the latest beta releases.

Kronnect Games
Email: contact@kronnect.com
Kronnect Support Forum: https://www.kronnect.com

How to use the asset in your project

Select from the top menu "GameObject / 3D Object / World Map 2D Edition- Main Map" option. You can also drag the prefab "WorldMap2D" from "Resources/Prefabs" folder to your scene. A WorldMap2D Game Object will appear in the hierarcy of your scene. Select it to show custom properties:

Custom Inspector Properties

Fit Width/Height/Center Map: controls and center how the map can be moved over the screen.

Render Viewport: when it has assigned a viewport gameobject, the map will show inside that viewport instead of the normal gameobject. Read "Using the Viewport feature" for more details.

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

You may want to not hide the Earth, but instead use the CutOut style, which will hide the Earth, but will prevent the geographic elements and lines to be seen when they're on the back of the sphere.

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

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).

Grid Color: modifies the color of the material of the grid (latitude and longitude lines).

Show Cities: activate/deactivate the layer of cities.

- **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.

- **Show Frontiers:** show/hide all frontiers. It applies to all countries, however you can colorize individual countries using the API.

- **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 (50:000:000:1 scale). Note that choosing high detail can impact performance on low-end devices.

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

- **Thin Lines:** when set to true, country frontiers lines won't get thicker when zooming in (useful for VR and mobile).

- **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.

- **Include All Regions**: when enabled, all regions of the current highligted country will be highlighted as well. If disabled (default behaviour), only the territory under the mouse will be highlighted. For instance, if you pass the mouse over USA and this option is enabled, Alaska will also be highlighted.

- **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:

  - **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.
  - **Font**: allows you to choose a different default font for labels (factory default is "Lato"). Note that using the decorator component you can assign individual fonts to countries.
  - **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.

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

- **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.

- **Navigation Time**: time in seconds for the fly to commands. Set it to zero to instant movements.

- **Autorotation Speed:** will make the Earth continuously rotate around its axis. Set it to zero to disable autorotation.

- **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.

- **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).

- **Screen Edge Scroll:** enables auto displacement of map when mouse is positioned on the edges of the screen.

- **Static Camera:** by default, it's the camera that moves when user drags or zooms in/out. If you set this property to true, the map will move itself, not the camera.

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

Using the Viewport feature

The asset allows to render the map inside a Viewport game object. The benefits for doing this is to allow cropping the map inside the rectangle defined by the viewport gameobject area when panning or zoomin in.

To use the viewport feature:

1- Select the top menu GameObject / 3D Object / World Map 2D Edition – Viewport to create a new Viewport gameobject in the scene (you can also drag & drop the prefab from Resources/Prefabs).
2- Assign the new viewport gameobject in the scene to the viewport property of the WorldMap in the inspector (you can also do that using code, check the demo.cs script for example)
3- That's all! The map will show up inside the viewport.

To deactivate the viewport:

1- Select the viewport field in the inspector of WorldMap component and click delete (basically, remove the reference to the viewport game object).
2- Delete the viewport game object from the scene.

The demo scene includes sample code illustrating the use of viewport using code (see demo.cs). Here's a video of the feature in action: https://youtu.be/qJtDkEsZ4ZE

Positioning the map on the screen

WPM 2D Edition can be positioned on your scene in any location with a custom rotation and scale. However, you may want to use as part of your UI, for example, as if it were a Canvas UI element, drawn on a screen overlay.

V5.2 includes an option that mimic the screen overlay mode of UI elements. It works by moving and parenting the viewport in front of the main camera at the position and size specified by a normalized rectangle, where 0 is the left or bottom, and 1 is the top or right position of the screen.
Note that this option is only available to render viewports. The reason is that using the viewport is the only way to proper zoom in or pan the map without it moving outside the constraint rectangle (cropped rectangle).

To access this option, the viewport should be enabled and assigned to the map:

Once you assign a render viewport to the map, check the "Screen Overlay Mode" toggle and specify the rectangle of the map in the screen.

Included Fonts

WMSK includes several free fonts inside Resources/Font folder. You may remove or add your own fonts as desired. When using a new font, make sure you set its font size to 160. To assign a new font to the map asset, just drag and drop it into the Font property in the inspector (or use the circle selector next to it).

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 points of interest (POIs) or 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:
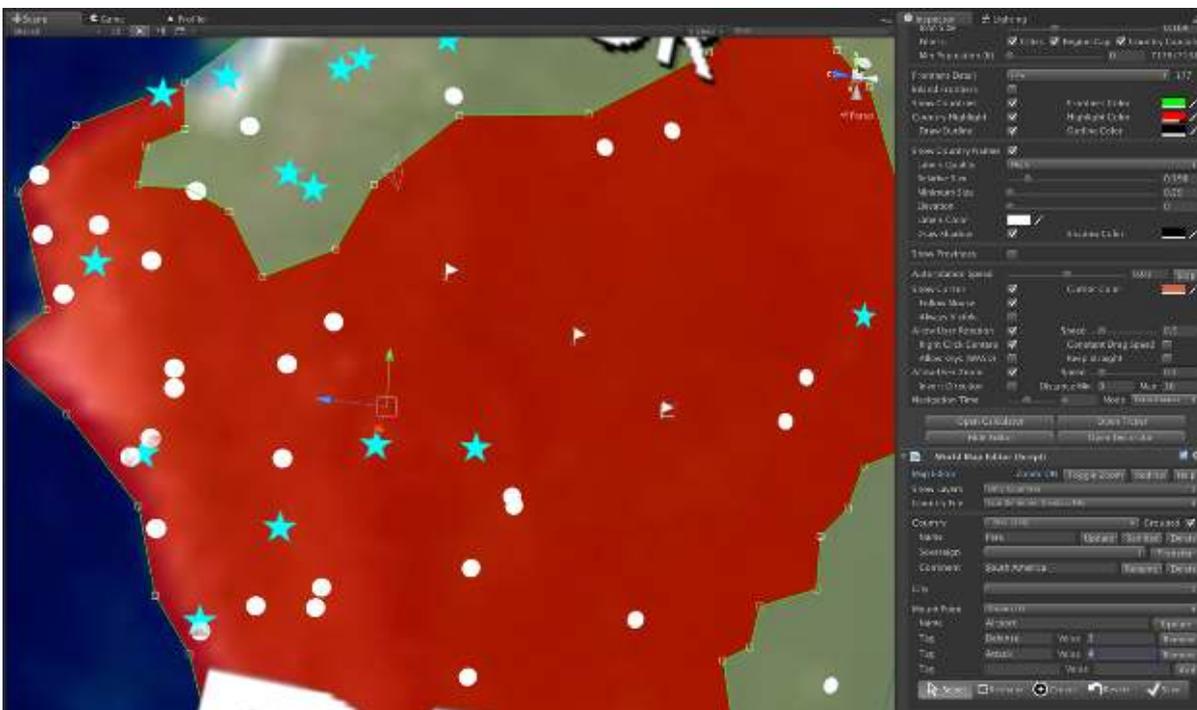
Reducing application build size

To reduce the size of your game, you can completely remove the following stuff:

- Demos folder.
- Textures of styles not used in Resources/Textures folder.
- Provinces data from Resources/Geodata folder if you don't use province borders. Same for cities.

Programming Guide (API)

You can instantiate the prefab "WorldMap2D" 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 WPMF;

WorldMap2D map;

void Start () {
map = WorldMap2D.instance;
        ...
}
```

(Note that you can have more WorldMap2D 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 WorldMap2D component of the GameObject).

Most of the public API and properties are located in WorldMap2D script inside Scripts/Core folder. Below is a list of them:

Public Events

public event OnCountryEvent **OnCountryEnter**;
public event OnCountryEvent **OnCountryExit**;
public event OnCountryClick **OnCountryClick**;
public event OnCountryHighlight **OnCountryHighlight**;

public event OnProvinceEvent **OnProvinceEnter**;
public event OnProvinceEvent **OnProvinceExit**;
public event OnProvinceClick **OnProvinceClick**;
public event OnProvinceHighlight **OnProvinceHighlight**;

public event OnCityEnter **OnCityEnter**;
public event OnCityEnter **OnCityExit**;
public event OnCityClick **OnCityClick**;

public event OnSimpleMapEvent **OnDragStart**;
public event OnSimpleMapEvent **OnDragEnd**;

Public Properties & Methods

Country related

map.countries: the array of Country objects. Note that the number and indexes of countries varies between the low and high-definition geodata files (reloaded when you change the frontiersQuality property in the inspector or in the API).

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

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.GetCountryIndex(ray, out countryIndex, out regionIndex): find the country pointed by the ray.

map.GetCountryNames(groupByContinent): returns an array with the country names, optionally grouped by continent.

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.countryRegionHighlighted: returns the Region object for the highlighted country (or null). Note that many contries have more than one region. The field mainRegionIndex of the Country object specified which region is bigger (usually the main body, being the rest islands or foreign regions).

map.countryRegionHIghlightedIndex: returns the index of the region of currently highlighted country for the regions field of country object.

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

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.frontiersThinLines: makes countries frontiers lines always thin irrespective of zoom level.

map.RenameCountry(oldName, newName): allows to change the country's name. Use this method instead of changing the name field of the country object.

map.BlinkCountry(country, color1, color2, duration, speed): makes the country specified toggle between color1 and color2 for duration in seconds and at speed rate.

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.GetCountryRegionZoomExtents(index): gets the required zoom level to show a custom country within screen borders.

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.SetCountryRegionMaterial: manually sets the material used by a country region.

map.CountryNeighbours(countryIndex): returns the list of country neighbours.

map.CountryNeighboursOfMainRegion(countryIndex): same but taking into account only the main region of the province (just in case the province could have more than one land region).

map.CountryNeighboursOfCurrentRegion(): same but taking into account the currently highlighted province.

Province related

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.provinceRegionHighlighted: returns the highlighted region of the province/state (or null).

map.provinceRegionHighlightedIndex: returns the index of the region highlighted for the regions field of province object).

map.provinceLastClicked: returns the last province clicked by the user..

map.provinceRegionLastClicked: returns the last province's region clicked by the user..

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

map.enableProvinceHighlight: toggles province highlighting when mouse is over.

map.provincesFillColor: color for the highlight of provinces.

map.provincesColor: color for provinces/states color.

map.GetProvinceIndex(name): returns the index of the province in the array. Please note that there some provinces with same name. You may want to use GetProvinceIndex(country, name) instead.

map.GetProvinceIndex(country, name): returns the index of the province inside the province array of the country object.

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.GetProvinceNames(groupByCountry): returns an array with the province names, optionally grouped by country.

map.RenameProvince(oldName, newName): allows to change the province's name. Use this method instead of changing the name field of the province object.

map.DrawProvinces(countryIndex, includeNeighbours, forceRefresh): draws borders of the provinces for the given country. Optionally can add the neighbouts province's borders. This method is used internally – to show provinces and names for specific countries please check the code of "Show State Names" in demo scene 1.

map.HideProvinces(): hides the provinces borders.

map.BlinkProvince(province, color1, color2, duration, speed): makes the province specified toggle between color1 and color2 for duration in seconds and at speed rate.

map.FlyToProvince(name/index, duration, zoomLevel): start navigation to specified province, optionally passing a duration in seconds (*navigationTime* by default) and the desired final zoomLevel. The list of provinces names can be obtained through the provinces property.

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

map.SetProvinceRegionMaterial(provinceIndex): manually sets the material used by a province region.

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

map.ProvinceNeighbours(provinceIndex): returns the list of province neighbours.

map.ProvinceNeighboursOfMainRegion(provinceIndex): same but taking into account only the main region of the province (just in case the province could have more than one land region).

map.ProvinceNeighboursOfCurrentRegion(): same but taking into account the currently highlighted province.

Cities related

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

map.GetCityNames: return an array with the names of the cities.

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

map.cityHighlightedIndex: returns the city index under the mouse cursor (or -1 if none).

map.lastCityClicked: returns the city clicked by the user.

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.cityClassAlwaysShow: combiantion of bitwise flags to specify classes of cities that will be drawn irrespective of other filters like minimum population. See CITY_CLASS_FILTER* constants.

map.citiesColor: color for the normal cities icons.

map.citiesRegionCapitalColor: color for the region capital cities icons.

map.citiesCountryCapitalColor: color for the country capital cities icons.

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.cityHitTestRadius: specifies the radius multiplier for mouse selection of cities. Default value is 200.

Earth related

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

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

map.earthColor: the currently color used in the Earth when style = SolidColor.

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.gridLinesColor: color of latitude and longitude lines.

User interaction

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

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

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

map.invertZoomDirection: controls the zoom in/out direction when using mouse wheel.

map.zoomMinDistance / map.zoomMaxDistance: limits the amount of zoom user can perform.

map.allowUserKeys/map.dragFlipDirection: enables/disables user drag with WASD keys and direction.

map.allowScrollOnScreenEdges: enables/disables autodisplacement of the map when mouse is positioned on the edges of the screen.

map.dragConstantSpeed: disables drag acceleration/damping.

map.zoomConstantSpeed: disables zoom acceleration/damping.

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

map.mouseDragSensibility: multiplying factor for the drag functionality.

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.fitWindowWidth: makes the map occupy the width of the screen.

map.fitWindowHeight: makes the map occupy also the height of the screen.

map.CenterMap(): positions the map in front of the main camera.

map.FlyToLocation (x, y, z): same but specifying the location in local Unity spherical coordinates.

Labels related

map.showCountryNames: toggles countriy labels on/off.

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.countryLabelsFont: Font for the country labels.

Mount Points related

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 and province.

Other

map.SetZoomLevel(level): apply one-time zoom level from 0 (closest) to 1 (farther).

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

map.HideContinentSurface(name): uncolorize/hide countries belonging to the specified continent.

map.AddMarker(marker, planeLocation, markerScale): adds a custom marker (provided by you) to the map at plane location with specified scale (markerScale).

map.AddLine(start, end, color, elevation, drawingDuration, lineWidth, fadeAfter): adds an animated line to the map from start to end position and with color, elevation and other options.

map.calc: accesor to the Calculator component.

map.ticker: accesor to the Tickers component.

map.decorator: accesor to the Decorator component.

Additional Components

World Map Calculator

This component is useful to:

- Convert units from one coordinate system to another (for instance from plane coordinates to degrees and viceversa).
- 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 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 map gameObject.

On the Inspector you will see the following custom editor:
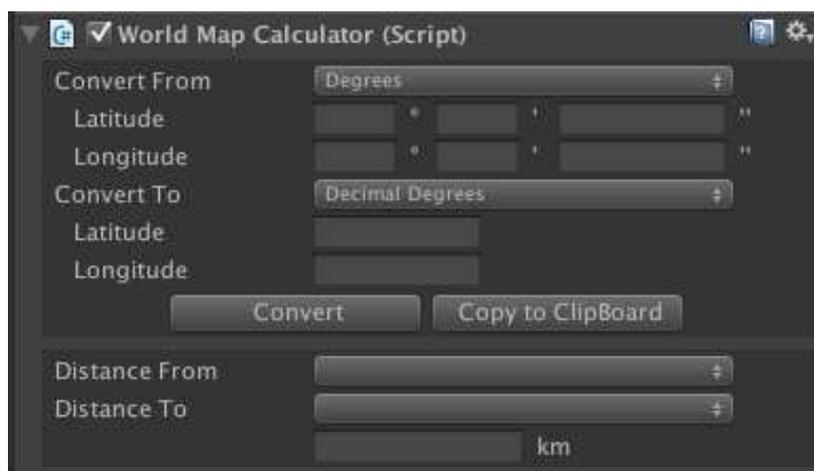
Using the converter from code

You may access the conversion API of this componet from code through **map.calc** property. The conversión 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 plane 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 (normalized) you use:

> **map.calc.fromUnit = UNIT_TYPE.PlaneCoordinates**
> **map.calc.fromX = <X position in local sphere coordinates >**
> **map.calc.fromY = <Y 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 >**

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 press the 'C' key.

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)

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 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 map gameObject.

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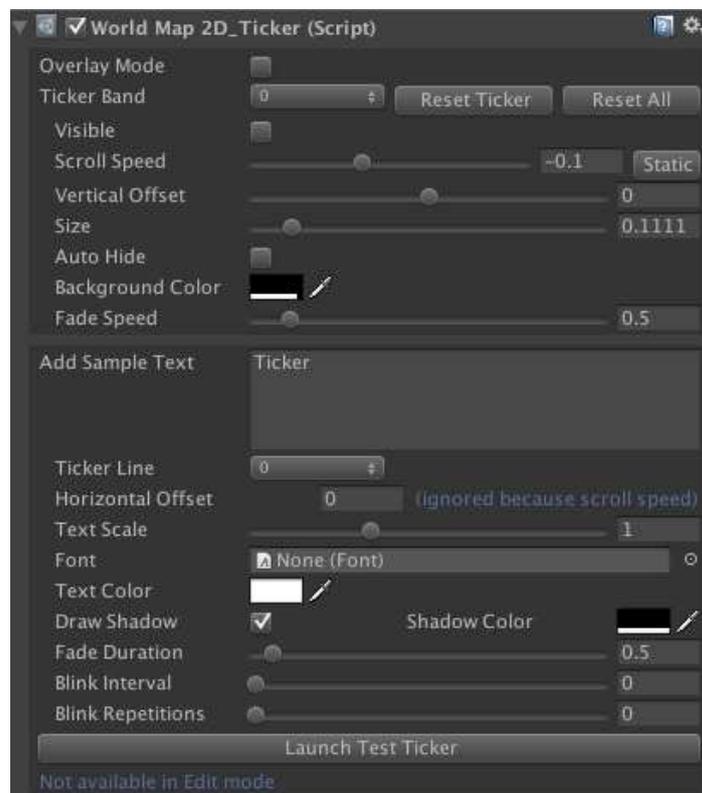
The top half of the inspector corresponds to the Ticker Bands configurator. You may customize the look&feel of the 9 available ticker bands (this number could be incremented if needed though). Notes:

- 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:
- 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).
- Setting fade duration to zero will disable fading effect.
- 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.overlayMode: when enabled, ticker texts will be displayed on top of everything at nearclip distance of main camera.

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;

    // 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 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 map gameObject.

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**: if the label is visible (default = true);

- **labelOffset**: horizontal and vertical position offset of the label relative to the center of the country.

- **labelRotation**: rotation of the label in degrees.

- **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)
- **City File**: choose which file to edit. Please note that changes in one file don't propagate to the other:
  - Principal city catalogue
  - Principal + medium sized cities catalogue
- **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:
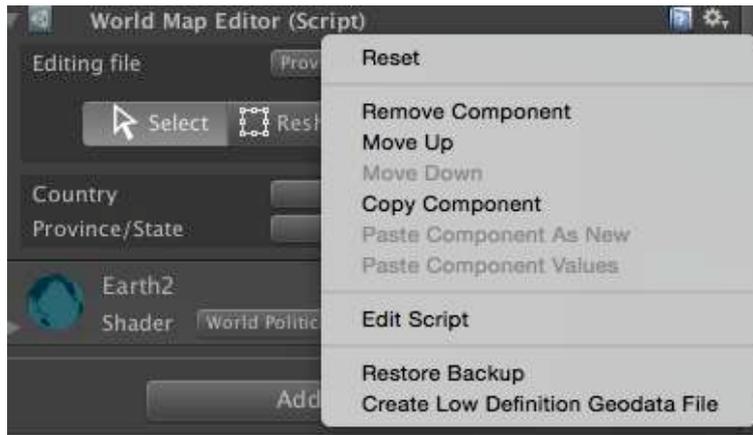
- **Restore Backup**: the first time you save changes to disk, a backup of the original geodata files will be performed. The backed up files are located in Backup folder inside the main asset folder. You may manually replace the contents of the Resources/Geodata folder by the Backup contents manually 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.

Reshaping options

When you select a country, the Reshape main option will show the following tools:
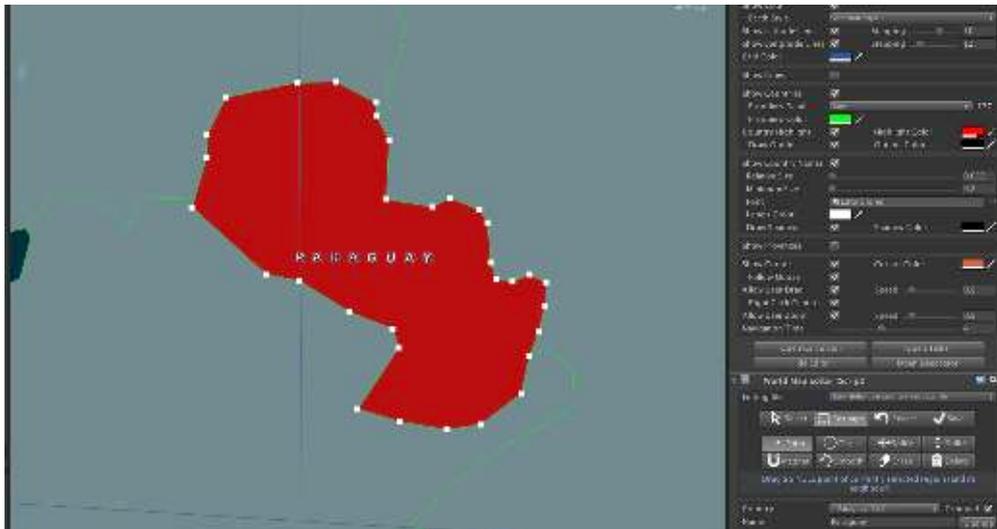
- **Point tool**: will allow you to move one point at a time. Note that the corresponding point of the neighbour will also be moved along (if it exits). This way the frontier between two regions is easily modified in one step, avoiding gaps between adjacent regions.
- **Circle tool:** similar to the 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.

Map Editor Tips

- Before start making changes, determine if you need the high-definition file or not. If you don't need it for 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.
- The high-definition file has lots of points. Current operation in this mode on some big countries (like Russia or Antartica) can be quite slow on some machines (although functional).
- You may change temporarily the scale of the map gameobject to 2000,1000,1 to make easier both the zoom and selection operations.
- If you decide to modify the high-definition file, you should complete all your modifications first in this map. Then use the command "Create Low Definition Geodata File" from the gear command, and adjust the low-def map afterwards.
- If you make any mistake using the Point/Circle tool, you can Undo (Control/Command + Z or Undo command from the Edit menu).
- Alternatively you can use the Revert option and this will reload the geodata files from disk (changes in memory wiil be lost).
- 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.
- Remember to visit us at kronnect.com for help and new updates.

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
WorldMap2D map = WorldMap2D.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);
```