

Terratype

Umbraco Multi map provider

Installation

Installing via Nuget

This Umbraco package can be installed via Nuget

The first part is the Terratype framework, which coordinates the different map providers, which can be found here

<https://www.nuget.org/packages/Terratype/>

```
PM> Install-Package Terratype
```

And the second part is a Terratype provider, which coordinates the configuration, editing and rendering of a particular interactive map. Note, there are no restriction on the number of providers that can be installed. You can, if you prefer use two or more different providers simultaneous, and existing Terratype content can be switched between providers seamlessly without the need to write any conversion code. You can even use Google Maps in the backend of Umbraco for the content editor and use Leaflet to render the same content as interactive maps in Razor on the frontend. Currently there are two providers available

- **Google Maps**

Mapping system operated by Google. Requires an API key and is free up to 25,000 map loads per 24 hour period, then \$0.50 per 1000 addition requests. Very customisable with 26 map styles including satellite and street view. Has search facilities that can be restrict to individual countries.

<https://www.nuget.org/packages/Terratype.GoogleMapsV3>

```
PM> Install-Package Terratype.GoogleMapsV3
```

- **Leaflet**

This is a Javascript framework that displays maps from, what are called, Tile Servers, which is a type of Open Source standard for interactive maps. As well as large established companies like Google.cn and Bing operating Tile Servers, there is OpenStreetMap which is a free collaborative open source mapping system. Leaflet only contains a subset of features compared to other providers, but one of its great advantages is that you can configure Leaflet to use different Tile Servers for different resolutions, so that as you zoom in or out of the map different Tile Servers can be used. Currently there are about 20 Tile Servers available including offerings from OpenStreetMap, OpenSeaMap, OpenTopoMap, Tianditu and Google China.

<https://www.nuget.org/packages/Terratype.LeafletV1>

```
PM> Install-Package Terratype.LeafletV1
```

- **Bing Maps**

Bing Maps (previously Live Search Maps, Windows Live Maps, Windows Live Local, and MSN Virtual Earth) is a web mapping service provided as a part of Microsoft's Bing suite of search engines and powered by the Bing Maps for Enterprise framework. Contains a few styles and search.

<https://www.nuget.org/packages/Terratype.BingMapsV8>

```
PM> Install-Package Terratype.BingMapsV8
```

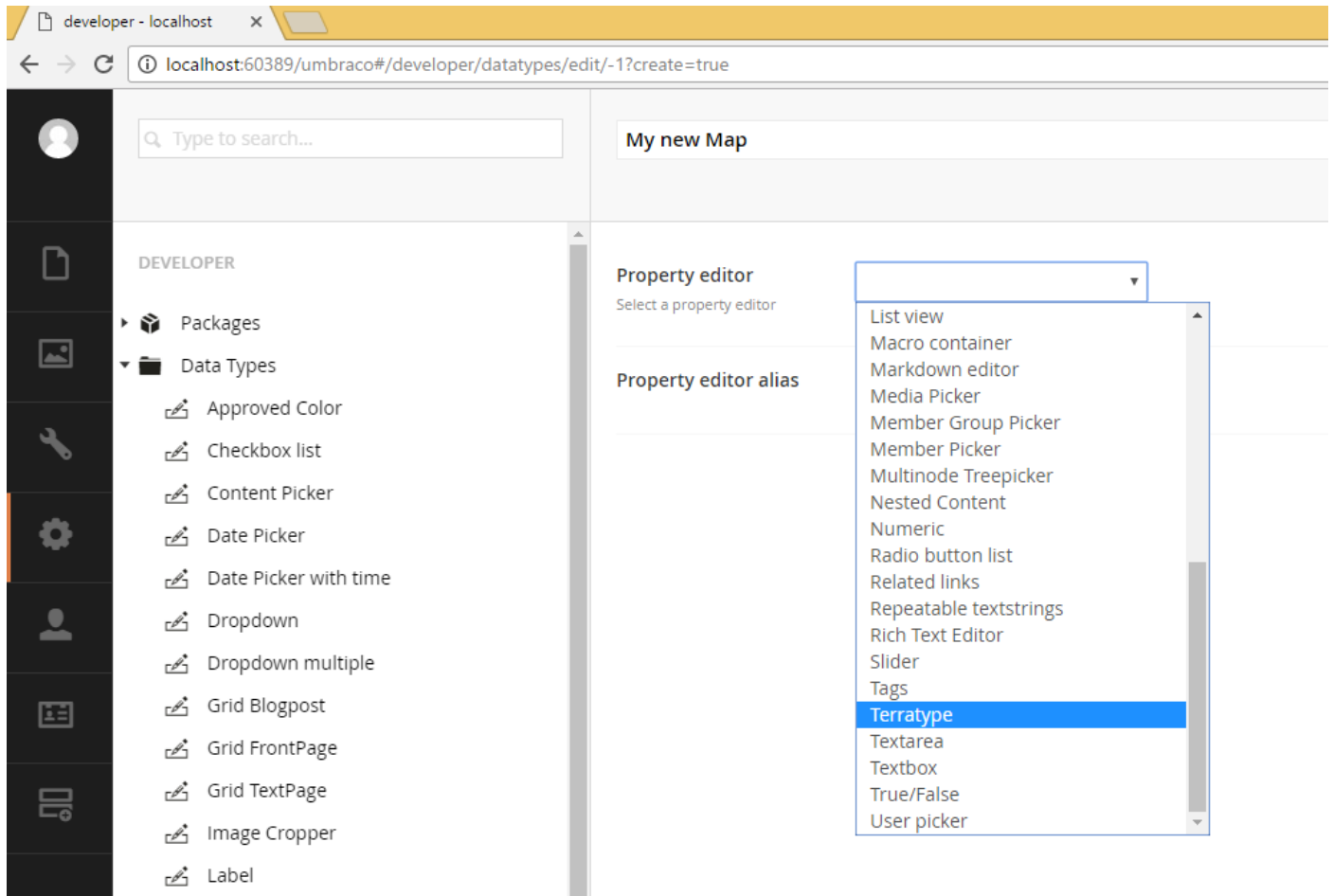
Installing via Umbraco Package

<https://our.umbraco.org/projects/backoffice-extensions/terratype/>

This installation contains all available map providers in one simple package.

Data Type

Once installed, create a new data type based off Terratype property editor.



Select the Map Provider you wish to use. You can only select from Providers that you have installed (via Nuget you have to install them individually, but the Umbraco Package contains them all). Each provider has different configuration settings, but they have a few items in common:-

Coordinate System

Different map providers and/or tile servers use different systems for coordinates. While the main one is known as WGS-84, and was adopted around most of the world, there have always been other systems, with some being more prevalent in different countries.

- [WGS-84](#)
Standard World Geodetic System as used by GPS system, and adopted worldwide except for China (see GCJ-02). Normal display format is Latitude,Longitude where Latitude goes from -90.0 to +90.0 and Longitude from -180.0 to +180
- [GCJ-02](#)
Under Chinese state law regarding restrictions on geographic data, you must use GCJ-02 encoded coordinates in China or for displaying Chinese coordinates correctly. Normally there is a difference of 100 to 700 metres between WGS-84 and GCJ-02

Icon

A map has to have an icon to indicate where the map is pointing to. This marker can be configured either from a predefined list of 25 icons, or allows the entry of any icon. Many of the predefined ones are sources from a Google CDN, but a few of the Umbraco logo ones are actually sourced as a relative path to your Umbraco instance. As you configure your icon, you may notice that this dynamically changes the marker as it is being displayed on the map above.

- **Image Url**
An absolute or relative path to the image you wish to use. You are restricted to using jpg, gif or png image files. Make sure that if you are running your Umbraco website on https protocol, then not to use images from an http source as they might have difficulty being accessed due to browser security settings. As you type the Image Url, Terratype will try and access the image file and will set the Width and Height property accordingly, or show an appropriate error if it can't either access the image file or it is in the wrong format.
- **Width and Height in Pixels**
If the image is larger or smaller than the size requested, then the image will be scaled to fit the Width and Height entered.
- **Horizontal and Vertical Anchor**
Within each image a particular pixel should indicate the exact location this icon is marking on the map. This pixel can be specified by counting the number of pixels from the top left hand corner, with a zero horizontal anchor indicating the left hand edge of the image, and the right hand edge being the width of the image in pixels minus 1. Vertical anchor go from zero referring to the top of the image, and height – 1 equalling the bottom of the image. The middle of the image can be specified as the anchor point by entering the width / 2 in Horizontal anchor and height / 2 in vertical anchor.
Easier to use, is the use of predefined anchor points; top, center, bottom and left, center and right. These are calculated from the Width and Height entered previously.

Label

A label is a popup info box that can appear above the map icon, when clicked by the user. This feature is switched off by default, and requires to be Enabled for it to function. The contents of the Label, the actual text displayed to a user of the map when they click the icon, is configurable by the content editor when the map is edited in a content node. You can selected how the content editor can edit the label, though note this doesn't change how an end user sees the map, if the map is rendered in a Razor view.

- **As an overlay when you click the map icon**
Whenever the content editor can drag and drop the map icon to specify the map's location, they can also click the icon to show an Umbraco overlay which then allows the editor to edit the label's content. The disadvantage of this mode, is that it might not be obvious to an editor that this functionality exists, if they haven't been informed.
Currently only one style for the label is selectable called 'Standard' which allows the entry of content using a Rich Text Editor.
- **Below the Map**
As well as showing the map, there is also a Rich Text Editor displayed below the map allowing the content editor the chance to edit the label's content. The disadvantage of this mode, is that the Rich Text Editor takes up a lot of space, and can be displeasing to view alongside the map. Though it is quick to read and amend.

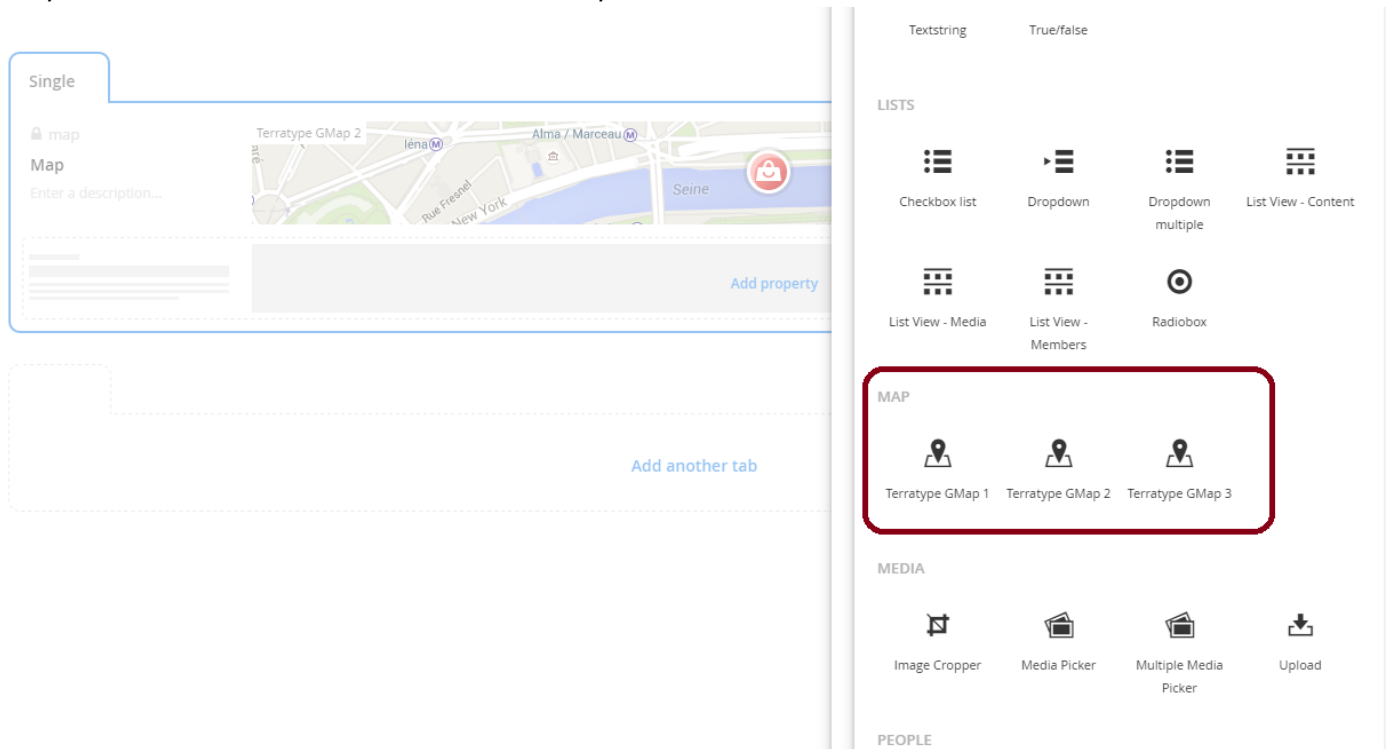
Currently there is only one style for a label called 'Standard' which allows the Content Editor to configure the label using a Rich Text Editor which allows Bold, Italics, Alignment, Bullet & Numbered lists, Url picker, Media picker and an Embed picker (Which I don't know what it does – so maybe someone will tell me one day, but it's a standard thing in Umbraco, so its included here)

Grid

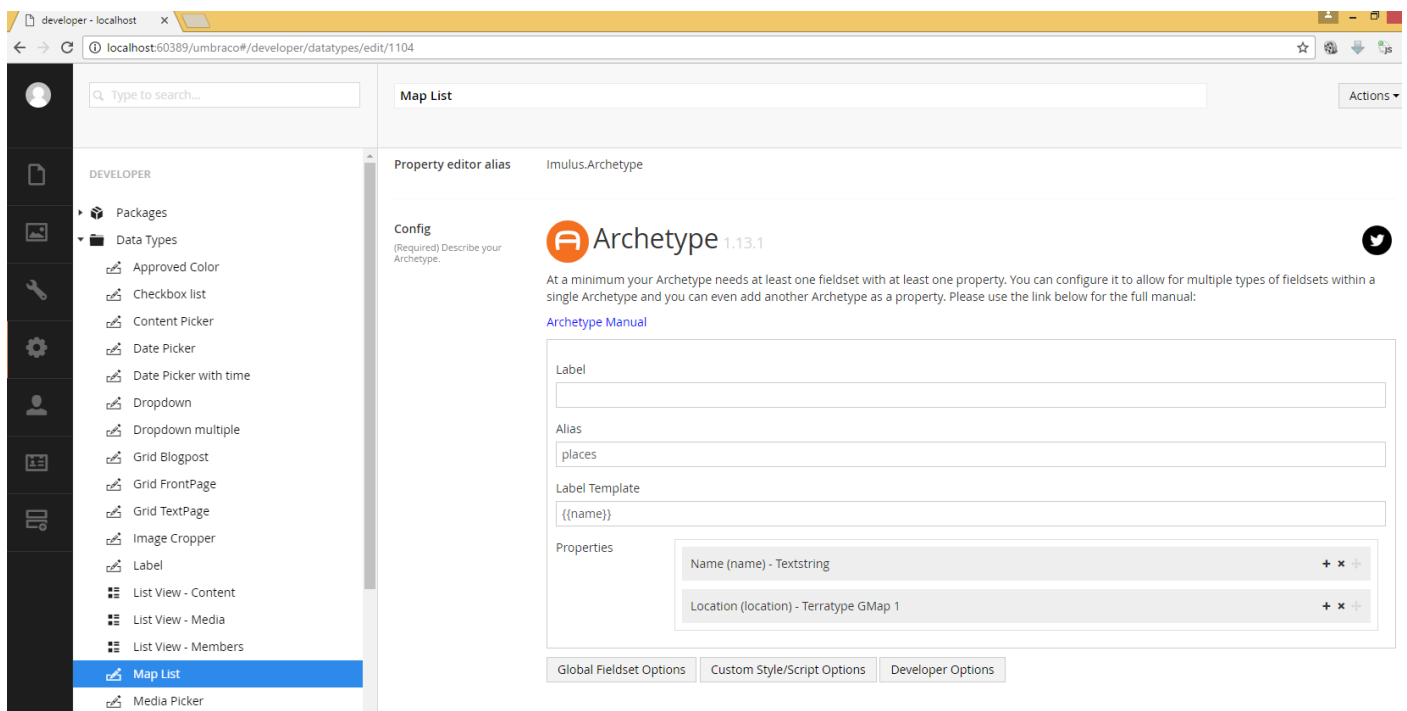
A great feature is allowing maps to be added to any [Grid Editors](#). This allows a Content Editor to add a Terratype map to a grid, though as part of Grid Editor configuration you can select if and when any data type can be entered including a Terratype map, but as default a Grid Editor will allow any type. To keep it simple for Content Editors, Terratype maps are just referred to as a Map type and you can give each Terratype map a name that the Content Editor can recognise.

Document Type

Once a new data type has been saved, you can add this data type to your document types. All Terratype Data Types that you create will be listed under MAP section in your editor list



If you require to create multiple locations, it is best to create a single map for each location and then use complex data types or separate content nodes to turn those maps into lists. So for example, create an [Archetype](#) datatype called Map List, where each entry contains a Terratype Map, along with other relevant meta data like Name, Address and Phone number for example, like



Then once this Archetype is placed within a Document Type, the editor creates a list like this

content - localhost x developer - localhost x

localhost:60389/umbraco#/content/content/edit/1093

Archetype Map

Content Properties

CONTENT

- Home
 - Learn
 - Explore
 - Extend
 - Blog
 - Contact
 - All Map Test
 - Nested Map
 - Archetype Map**
 - Grid Map
 - Single
- Recycle Bin

Rome + + ⌵ ×

Berlin + + ⌵ ×

Paris + + ⌵ ×

London + + ⌵ ×

New York + + ⌵ ×

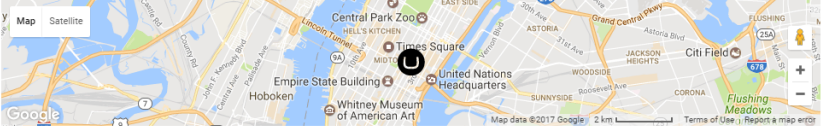
Name New York

Address Umbraco, LLC c/o Thomas Martin LLP 228 Park Avenue S, Suite 300 New York, NY 10003 USA

Phone +1.206.445.0048

Location

Search...



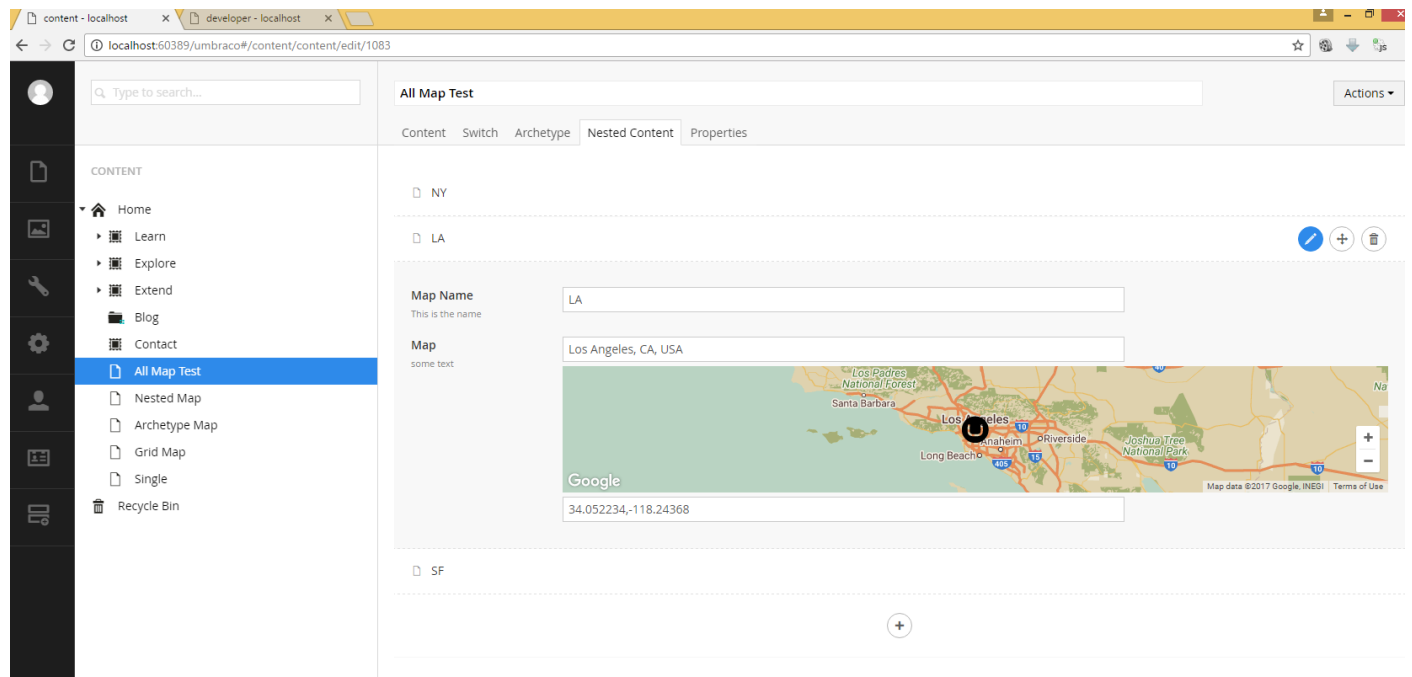
40.754326,-73.976209

Madrid + + ⌵ ×

Lisbon + + ⌵ ×

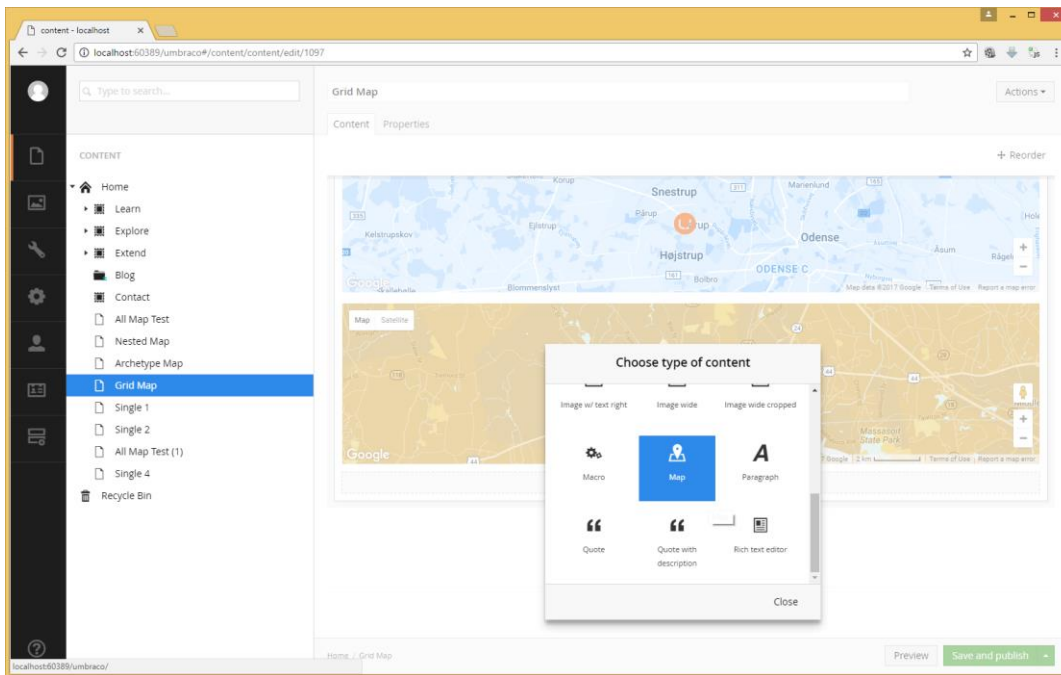
Content Editor

The content editor can select one location, and also the zoom level is also recorded. If search is available, then the last search term is remembered, though this value has no bearing on the behaviour of the map. As the location can be moved afterwards.

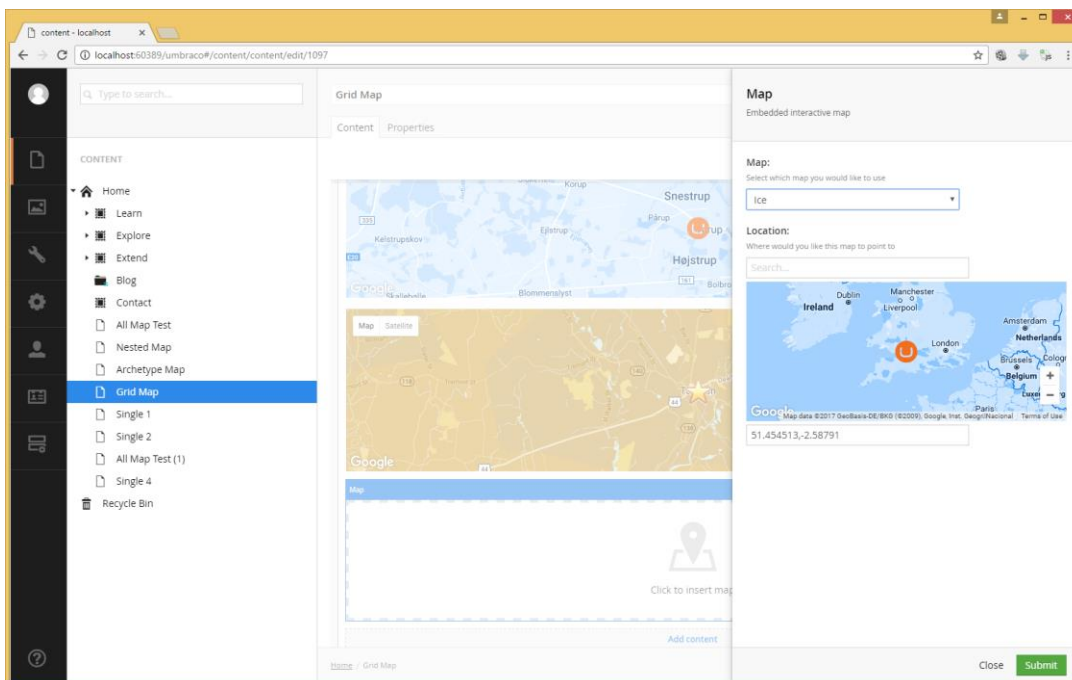


Grid Editor

A grid editor can add any Maps to a Grid control. First select Map from the Content Choices.



Once selected click to add a map and a Map overlay panel will appear allowing configuration of this map.



Once happy with the select, choose Submit.

As standard with all grids, a grid can be rendered using the following Razor command

```
@CurrentPage.GetGridHtml(Html, "grid_alias")
```

Model

Terratype.Models.Model

```
namespace Terratype.Models
{
    public class Model
    {
        /// <summary>
        /// Which provider is this map using to render
        /// </summary>
        public Provider Provider { get; internal set; }

        /// <summary>
        /// Where is this map pointing to
        /// </summary>
        public Position Position { get; set; }

        /// <summary>
        /// Image marker to display this location on the map
        /// </summary>
        public Icon Icon { get; internal set; }

        /// <summary>
        /// Current map zoom
        /// </summary>
        public int Zoom { get; set; }

        /// <summary>
        /// Last search request
        /// </summary>

        public string Lookup { get; set; }

        /// <summary>
        /// Current map height
        /// </summary>
        public int Height { get; set; }
    }
}
```

Provider will be a strongly typed version of the a Map Provider previously selected when creating the Data type. The Position will also be a strongly typed version. If you know the Provider or Position type, you can cast to the type required. Eg.

```
var GMapProvider = (Terratype.Providers.GoogleMapsV3) Model.Content.Map.Provider;
```

This then allows access to all the settings that are relevant to GoogleMapsV3, though any changes to them can not be saved. The only way to change settings is via the Data type editor.

Terratype.Models.Position

```
public abstract class Position
{
    /// <summary>
    /// Unique identifier of coordinate system
    /// </summary>
    public abstract string Id { get; }

    /// <summary>
    /// To display position to user
    /// </summary>
    /// <returns></returns>
    public override string ToString()

    /// <summary>
    /// Parses human readable position
    /// </summary>
    public virtual void Parse(string datum)

    /// <summary>
    /// Parses human readable position if possible
    /// </summary>
    public virtual bool TryParse(string datum)

    /// <summary>
    /// Convert the current position to a Wgs84 location
    /// </summary>
    /// <returns>A Wgs84 location</returns>
    public abstract LatLng ToWgs84();

    /// <summary>
    /// Set the position to the Wgs84 location provided
    /// </summary>
    public abstract void FromWgs84(LatLng wgs84Position);

    /// <summary>
    /// DatumType varies for each coordinate system, and contains precise values
    /// that can be used to perform calculations on. So for example GCJ-02 has Latitude &
    /// Longitude that represents the current GCJ-02 position.
    /// </summary>
    public DatumType Datum;
}
```

If you plan to perform calculations on a location, and not just display a location to a user, then you have a choice of using either ToWGS84() and FromWGS84() which guareentees to use Wgs84 coordinates regardless of what system the map has been setup with or each Coordinate System has its own Datum value, of a type specific to itself, that can be used to get or set a precise location.

Render

Any map can be rendered using razor with the command `@Html.Terratype()`

First you will need to include Terratype at the top of your razor page

`@using Terratype;`

Specification

`@Html.Terratype(Options, Map, Label)`

Options: Optional class that is used to set extra values that can be used to control how this map is rendered

- **Height:** Height of the map in pixels, this value is separate from the height set for the content editor. Most map providers require a physical pixel height, as they can't display maps to relative dimensions.
- **Language:** The language you would like the map to use, this can be a 2 or 4 letter code; 'en' for English. This value is passed directly to the Map Provider. If not set then the current language will be used.
- **MapSetId:** When you make Multiple calls to `@Html.Terratype()` with the same MapSetId, then all those locations will be displayed on the same map. The first map rendered of the set will dictate the settings used, like map styles, api keys etc. Though you can override this with Zoom, Position & Provider options
- **Zoom:** Regardless of the zoom set in the map, use this zoom value instead. Can be left null to ignore.
- **Position:** The starting position to use for this map, again this can be left null if either AutoFit is true or the first map of the MapSet is to be used to set the center point.
- **Provider:** You can override the provider, or any value within the provider with your own custom ones. Note that the values are merged between the provider of the first map in the Map set and any values you set here, with this Provider taking precedence.
- **AutoShowLabel:** Show the label as the map is first rendered
- **AutoRecenterAfterRefresh:** Each time the map refreshes re-centre back to original position
- **AutoFit:** Try and display all markers on the map. This will center the map and calculate the appropriate zoom level required to show all markers at once
- **Tag:** Reference value returned for a position when monitoring `onClick()` event
- **DomMonitorType:** Declare which type of DOM monitoring you wish the rendered map to use. Dom monitoring is used to denote when changes happen to the DOM surrounding the map (resizing, hidden, reshown) and redraws the map accordingly.

Currently **Javascript** is the default, except for Bing maps which has its own internal Dom monitoring and actively handles redrawing when necessary.

jQuery can only be used when all DOM manipulation is done via jQuery. So for example if the map is located on a tab control, the hiding and showing of tabs must use jQuery or a jQuery plugin. It is your responsible to include the jQuery library on your webpage, if the library isn't present, then Terratype will fail back on using Javascript monitoring anyway.

Disable will stop any sort of monitoring of updates to the view or layout of a webpage. This will mean the Map might never appear, if it's initially located on a hidden element. Only use this option for very simple pages with no DOM changes.

Map: The content property of the map.

Dynamics: If you are using Dynamics in your razor templates, usually denoted by

`@inherits Umbraco.Web.Mvc.UmbracoTemplatePage`

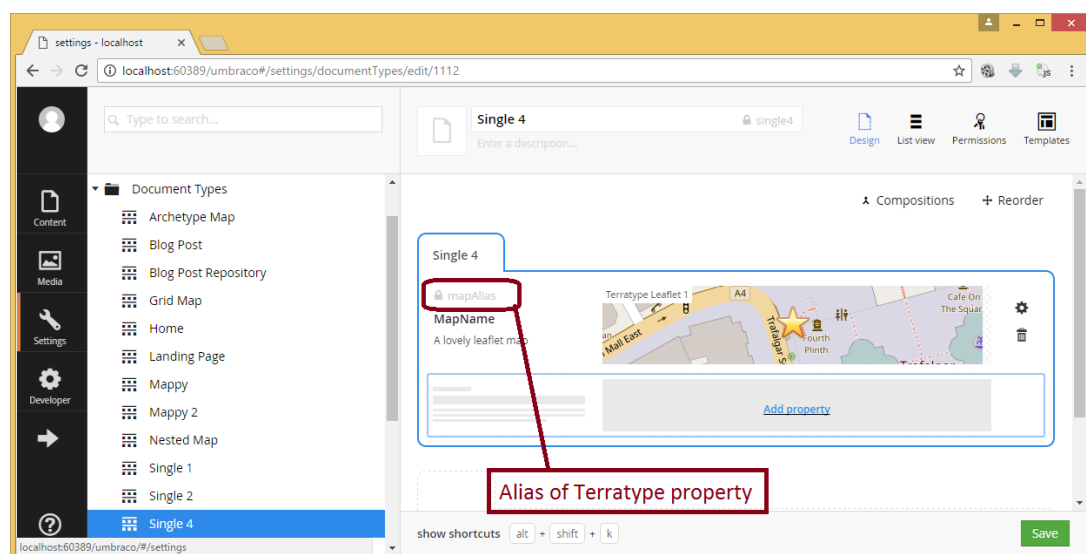
at the top of the Razor page, then any or the following string, `IPropertyProperty` or object can be used to specify the map you wish to render:-

`@Html.Terratype("mapAlias")`

`@Html.Terratype(Model.Content.GetProperty("mapAlias"))`

`@Html.Terratype(Model.Content.GetProperty("mapAlias").Value)`

Note these examples use 'mapAlias', but you will need to use the actual alias of the Terratype property that you specified when creating your Document Type, as shown:-



Strongly Type: If you are using Strongly Type classes in your razor templates, usually denoted by

`@inherits Umbraco.Web.Mvc.UmbracoTemplatePage<MyDocTypeAlias>`

At the top of the Razor page, please note 'MyDocTypeAlias' should actually be the alias of the document type you have that is containing the Terratype property.

Then you should already have a strongly typed property of type `Terratype.Models.Model` in your `Model.Content`, so

`@Html.Terratype(Model.Content.MapAlias)`

will work.

Label: Razor code and/or Html code that is displayed if the user clicks this icon. This has to be wrapped in `<text>` so that the razor engine knows this is html that you wish to embed as an argument to `@Html.Terratype()`. There is no restriction to what html or other razor commands that you place between the `<text>`, except other `<text>` commands. If no Razor and/or Html code is present then any label created or edited by the content editor will be used, if this option has been enabled for this map.

Javascript Events

There are six globally accessible events that you can hook into when a map is rendered. These have to be used in a script tag at the bottom of your page (After all `Html.Terratype()` methods)

- `terratype.onInit(function (provider) { alert(provider.id) })`:
This is called once for each map provider used, so once for Google Maps, once for Bing Maps, once for Leaflet, etc.
- `terratype.onLoad(function (provider, map) { alert(provider.id + ',' + map.id) })`:
This is called once for each map present
- `terratype.onRender(function (provider, map) { alert(provider.id + ',' + map.id) })`:
This is called once the map has actually rendered on the screen, this only happens if the map scrolls into view or some event allows the map to be displayed, eg. The tab the map is on becomes visible.
- `terratype.onRefresh(function (provider, map) { alert(provider.id + ',' + map.id) })`:
This is called whenever the map needs to refresh itself, this might be because the map is resized or the tab the map is on has returned into view
- `terratype.onZoom(function (provider, map, zoom) { alert(zoom) })`:
This is called whenever the map changes zoom level
- `terratype.onClick(function (provider, map, marker) { alert(marker.tag) })`:
This is called every time an map icon is clicked that has a label attached
- `terratype.init()`
Call to load any maps that are added to the page after the initial render. Useful for pages that are created using js frameworks like Angular or React

The passed arguments in the call backs are

- **provider**
id: String. Unique identifier for this provider. Eg. `'Terratype.GoogleMapsV3'`
maps: Array of **map** objects.
- **map**
id: Int. Mapset of this map, as set in Options
zoom: Int. Current zoom level of map
handle: Object. Handle to the internal map provider object.
Either a `google.maps.Map`, `Microsoft.Maps.Map` or `L.map` object
isVisible():Function. Return **true** if the map is currently in view
refresh():Function. Call to redisplay the map
getPosition(tag):Function. Returns a **position** object with the matching **tag** value
positions: Array of **position** objects
- **position**
tag: String. Tag set in options
handle: object. Underlying object that references an internal map marker
Either a `google.maps.Marker`, `Microsoft.Maps.Pushpin` or `L.icon` object

Example 1 – Single map with 1 icon using Dynamics

```
@Html.Terratype("mapAlias")
```

This will display a map from property `mapAlias` with whatever label the content editor has created.

Example 2 – Single map with 1 icon using Dynamics

```
@Html.Terratype("mapAlias",  
    @<text>  
        <div>  
            This icon is at  
            @((Model.Content.GetProperty("mapAlias").Value as Terratype.Models.Model).Position)  
        </div>  
    </text>  
)
```

This will display a map from property `mapAlias` with a label.

Example 3 – Archetype map with lots of icons

```
@foreach (var record in Model.Content.Archetype)  
{  
    var name = record.GetValue<string>("name");  
    var map = record.GetValue<Terratype.Models.Model>("location");  
  
    @Html.Terratype(new Options { MapSetId = 1 }, map,  
        @<text>  
            @name is at @map.Position  
        </text>  
    )  
}
```

Given there is an Archetype with alias called 'archetype', created with two properties; name and location, this will render all locations created by the content editor on one single map (This is because all the maps rendered will have the same `MapSetId` of 1)

Example 4 – Map with custom icon

```
@Html.Terratype(new Options
{
    Icon = new Terratype.Models.Icon
    {
        Url = new Uri("https://d30y9cdsu7xlg0.cloudfront.net/png/4096-200.png"),
        Size = new Terratype.Models.Icon.SizeDefinition
        {
            Width = 48,
            Height = 48
        },
        Anchor = new Terratype.Models.Icon.AnchorDefinition
        {
            Horizontal = new Terratype.Models.AnchorHorizontal(
                Terratype.Models.AnchorHorizontal.Style.Center),
            Vertical = new Terratype.Models.AnchorVertical(
                Terratype.Models.AnchorVertical.Style.Center)
        }
    }
}, Model.Content.Map)
```

Displays a map with a new Map icon, as defined in Url

Example 5 – Map listening to click events

```
@Html.Terratype(new Options { MapSetId = 66, AutoShowLabel = true, Tag = "marker1" }, Map,
    @<text>
        Popup text
    </text>
)
<script>
    terratype.onClick(function (provider, map, marker) {
        console.log('onClick: ' + provider.id + ', ' + map.id + ', ' + marker.tag);
    });
</script>
```

Console output would be

onClick: Terratype.GoogleMapsV3, 66, marker1

Example 6 – Nested Content map with lots of icons and hooks into events and displays reaction in console log

```
@Html.Terratype(new Options
{
    Provider = new Terratype.Providers.GoogleMapsV3()
    {
        Variety = new Terratype.Providers.GoogleMapsV3.VarietyDefinition()
        {
            Satellite = true
        }
    },
    Height = 1000,
    MapSetId = 2,
    Zoom = 5,
    Position = new Terratype.CoordinateSystems.Wgs84("-30,130")
})

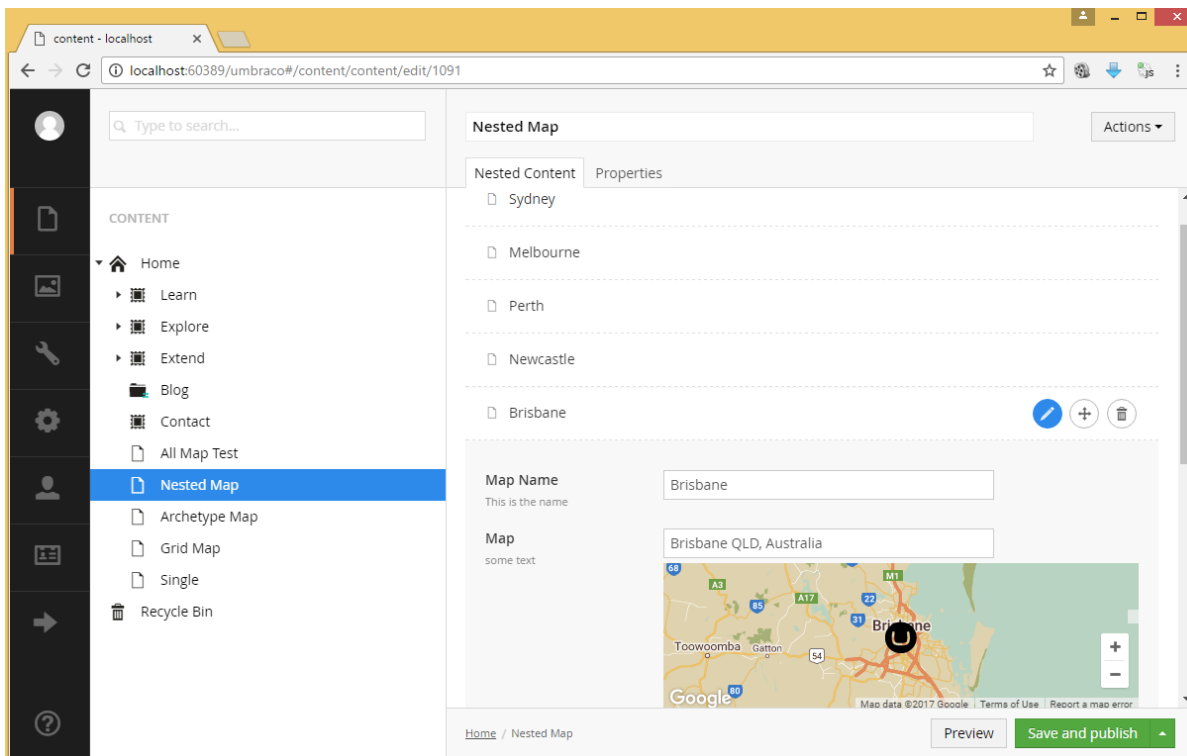
@foreach (var record in Model.Content.Nested)
{
    var name = record.GetProperty<string>("mapName");
    var map = record.GetProperty<Terratype.Models.Model>("map");

    @Html.Terratype(new Options { MapSetId = 2, Tag = "@name" }, map,
        @<text>
            @name is at @map.Position
        </text>
    )
}

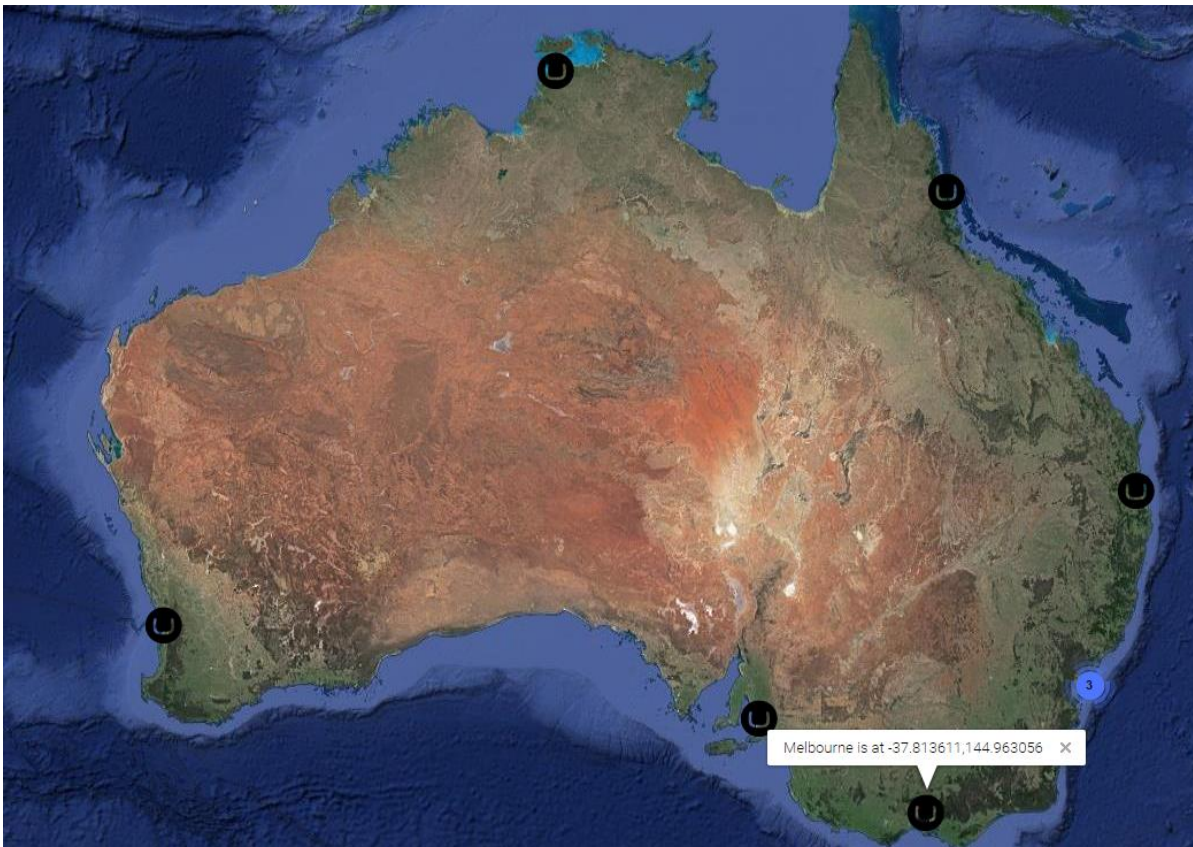
<script>
    terratype.onInit(function (provider) {
        console.log('onInit: ' + provider.id);
    });
    terratype.onLoad(function (provider, map) {
        console.log('onLoad: ' + provider.id + ', ' + map.id);
    });
    terratype.onRender(function (provider, map) {
        console.log('onRender: ' + provider.id + ', ' + map.id);
    });
    terratype.onRefresh(function (provider, map) {
        console.log('onRefresh: ' + provider.id + ', ' + map.id);
    });
    terratype.onZoom(function (provider, map, zoom) {
        console.log('onZoom: ' + provider.id + ', ' + map.id + ', ' + zoom);
    });
    terratype.onClick(function (provider, map, marker) {
        console.log('onClick: ' + provider.id + ', ' + map.id + ', ' + marker.tag);
    });
</script>
```

Given that there is a nested content property called **Nested** that itself has two properties of **mapName** and **map**, then this uses Terratype options to create a GoogleMapsV3 provider with Satellite view of height 1000 pixels and with a zoom of 5 and is centred on the location -30,130 (which so happens to be over the middle of Australia). And then for each entry in the nested content is rendered all on this one map. Each entry has its own label of **@name** is at **@map.Position** which displays the name and position of each icon

This is an example of what the content editor sees



And this is the view in the browser, once the razor code is rendered, using the data above



Each time the user clicked on Melbourne the console.log would output

onClick: Terratype.GoogleMapsV3, 2, Melbourne