

# 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
```

The second part is a provider that delivers map services for Terratype. So, for example, if you want to use Google Maps V3, then install Terratype.GoogleMapsV3 nuget package, from

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

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

Without a provider, Terratype can't do anything.

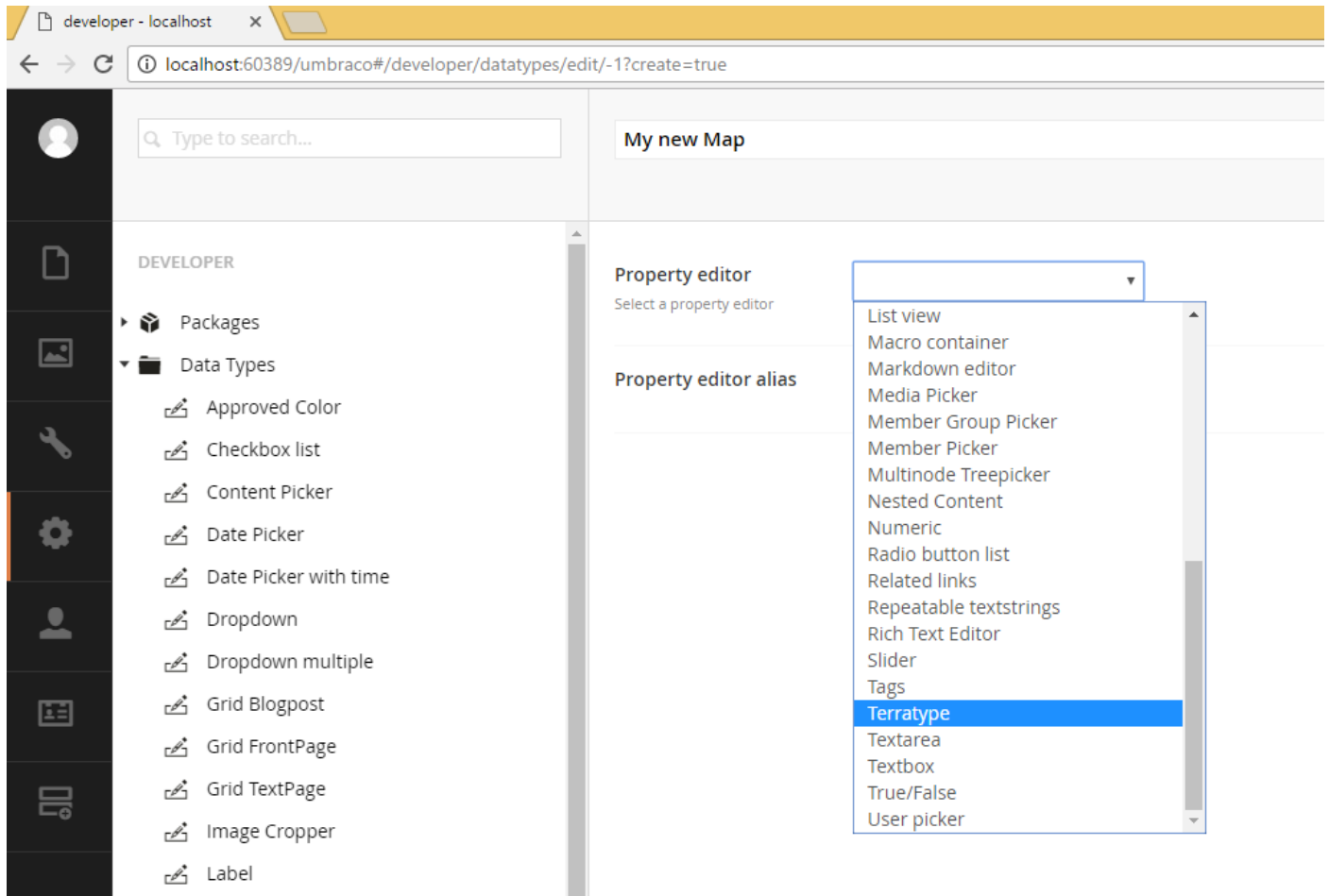
### Installing via Umbraco Package

<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.

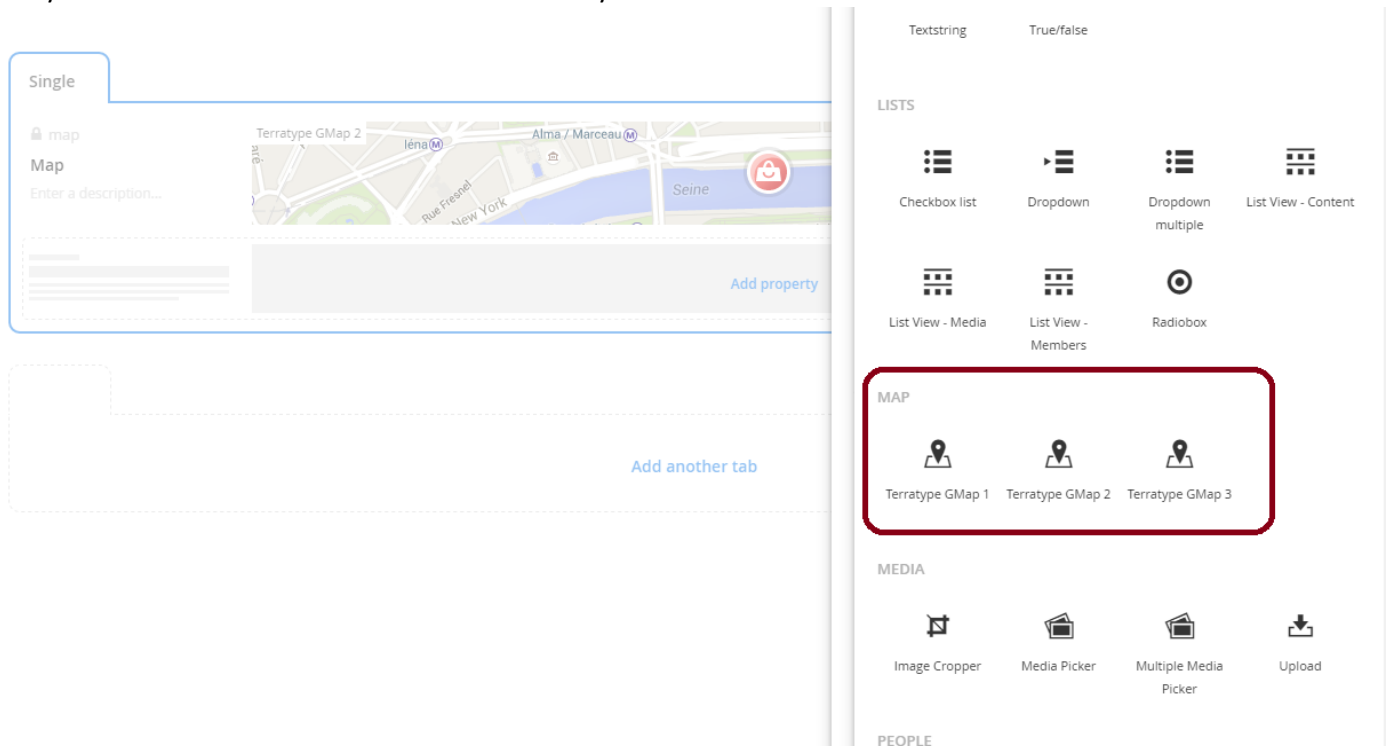


Select the Map Provider you wish to use. You can only select from Providers that you have installed via Nuget. Each provider has different configuration settings, but in common is that you require you to specify which coordinate system you wish to use.

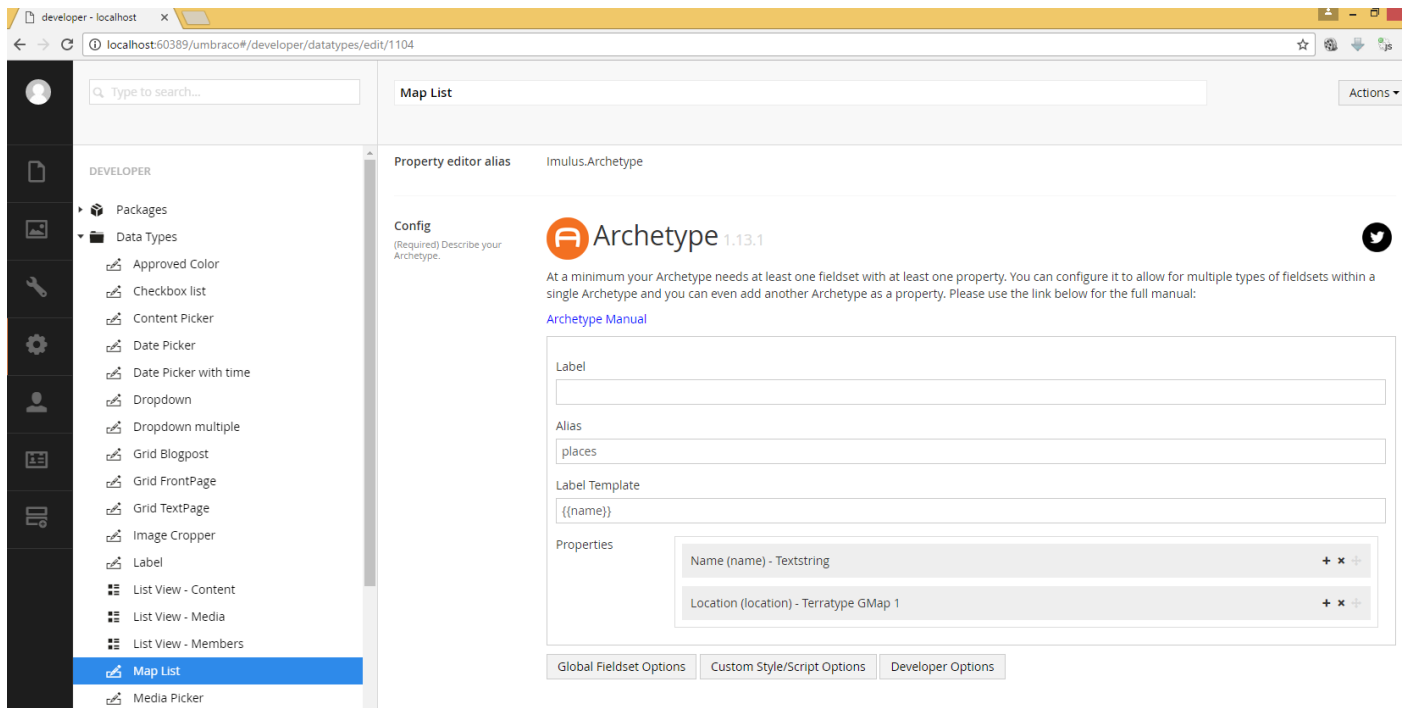
- [WGS-84](#)  
Standard World Geodetic System as used by GPS system, and adopted worldwide except for China (see GCJ-02). Normal display format is Latitude,Longitude where Latitude goes from -90.0 to +90.0 and Longitude from -180.0 to +180
- [GCJ-02](#)  
Under Chinese state law regarding restrictions 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

## 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 + + ⌵ ×

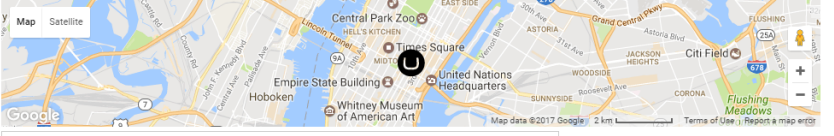
Name New York

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...



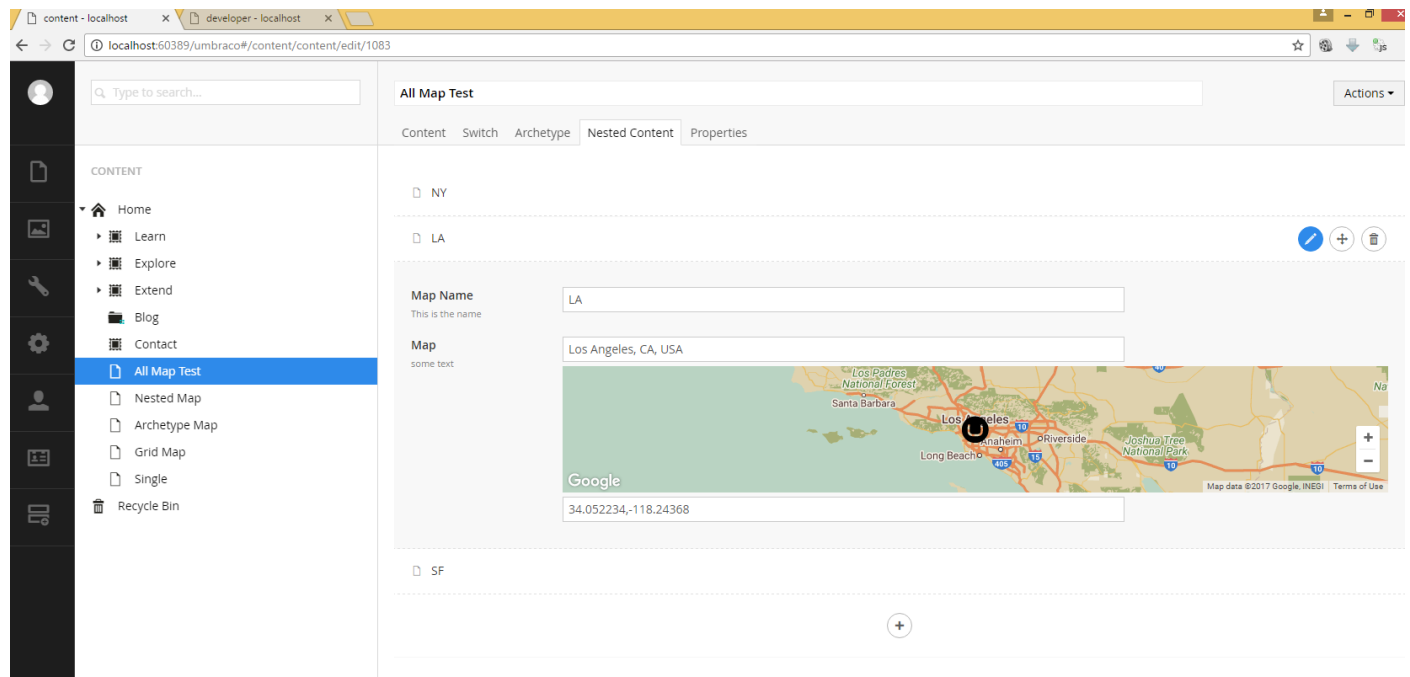
40.754326,-73.976209

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.



# 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; }
    }
}
```

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);
}
```

If you plan to perform calculations on a location, and not just display a location to a user, then always use ToWGS84() and FromWGS84(), as these will give you a precise location regardless of the coordinate system being employed by the map provider.

## 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 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.

**Map:** The content property of the map.

**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 no label will be displayed when a user clicks the icon.

### Example 1 – Single map with 1 icon

```
@Html.Terratype(Model.Content.Map,
    @<text>
        <div style="background-color:yellow;">
            This icon is at @Model.Content.Map.Position
        </div>
    </text>
)
```

This will display a map from content variable **Map** with a label that contains a yellow div.

### Example 2 – 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 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 3 – Nested Content map with lots of icons

```

@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")
})

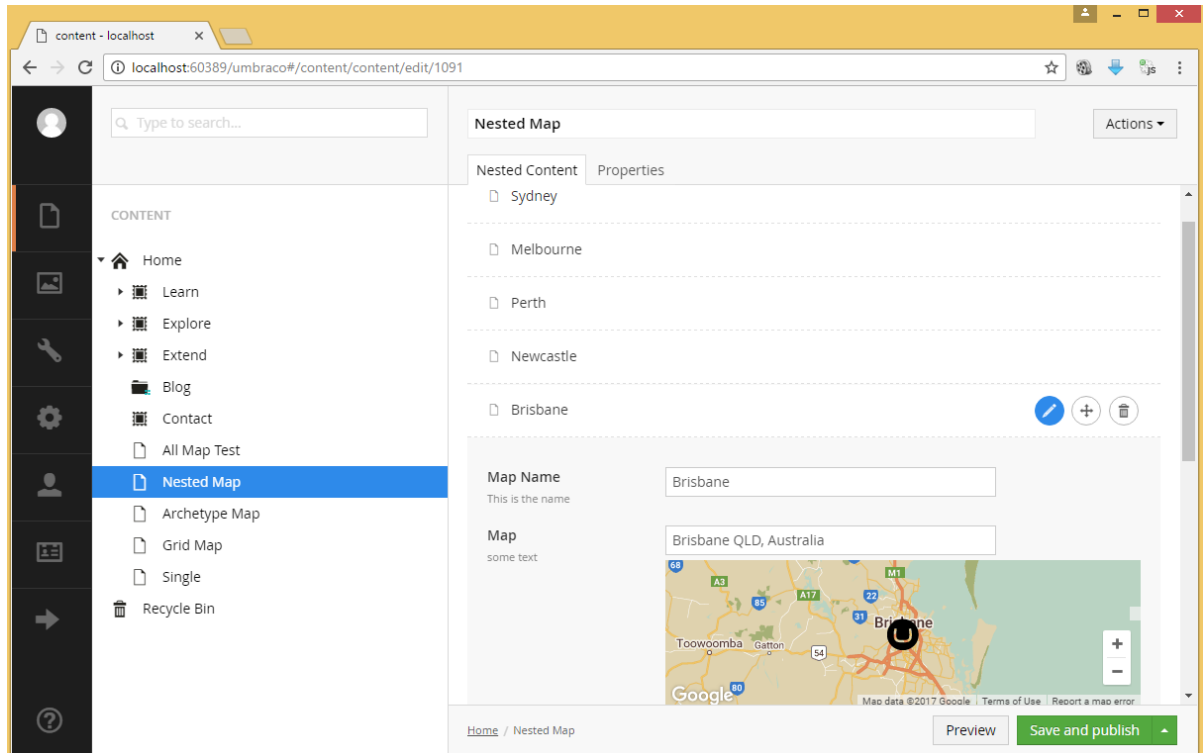
@foreach (var record in Model.Content.Nested)
{
    var name = record.GetProperty<string>("mapName");
    var map = record.GetProperty<Terratype.Models.Model>("map");

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

```

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

