TMS FNC Maps
DEVELOPERS GUIDE

May 2022
Copyright © 2021 - 2022 by tmssoftware.com bvba
Web: http://www.tmssoftware.com
Email: info@tmssoftware.com

Index

Availability

Supported frameworks and platforms
- VCL Win32/Win64
- FMX Win32/Win64, macOS, iOS, Android, Linux
- LCL Win32/Win64, macOS, iOS, Android, numerous Linux variants including Raspbian
- WEB: Chrome, Edge, Firefox, …

Supported IDE's
- Delphi XE7 and C++ Builder XE7 or newer releases
- Lazarus 1.4.4 with FPC 2.6.4 or newer official releases
- TMS WEB Core for Visual Studio Code 1.3 or newer releases

Important Notice: TMS FNC Maps requires TMS FNC Core (separately available at the My Products page)

Online references

TMS software website:
http://www.tmssoftware.com

TMS FNC Maps page:
http://www.tmssoftware.com/site/tmsfncmaps.asp

Description

TMS FNC Maps is a cross-framework, cross-platform and cross-service mapping component library. It includes an abstract map (TTMSFNCMaps) that is capable of rendering polygons (including rectangles and circles), polylines, markers, show HTML formatted popups and many more. It has events for mouse interaction as well as some basic options to configure the look and feel. Optionally, the underlying JavaScript can be modified to allow additional customization options. Also included is a component for calculating directions between 2 coordinates or addresses (TTMSFNCDirections), the ability to convert a coordinate to an address or vice versa (TTMSFNCGeoCoding) and retrieving your current location (TTMSFNCLocation). Below is a list of services that are currently supported.

- OpenLayers (OpenStreetMaps)
- TomTom
- Microsoft Azure Maps
- Microsoft Bing Maps
- Google Maps
- Here Maps
- MapBox
- MapKit JS

Getting Started

After installing TMS FNC Core, please follow instructions on getting the TTMSFNCWebBrowser up and running, which is the underlying component for TMS FNC Maps. More information about the browser and its capabilities can be found at the following link:

http://www.tmssoftware.biz/Download/Manuals/TMSFNCWebBrowserDevGuide.pdf

After installing TMS FNC Maps, the first thing that needs to be done is acquiring an API key (except for OpenLayers which is free and doesn't require an API key). Steps to obtain an API key for enabling the map as well as using directions, geocoding and location services can be found at the following page:

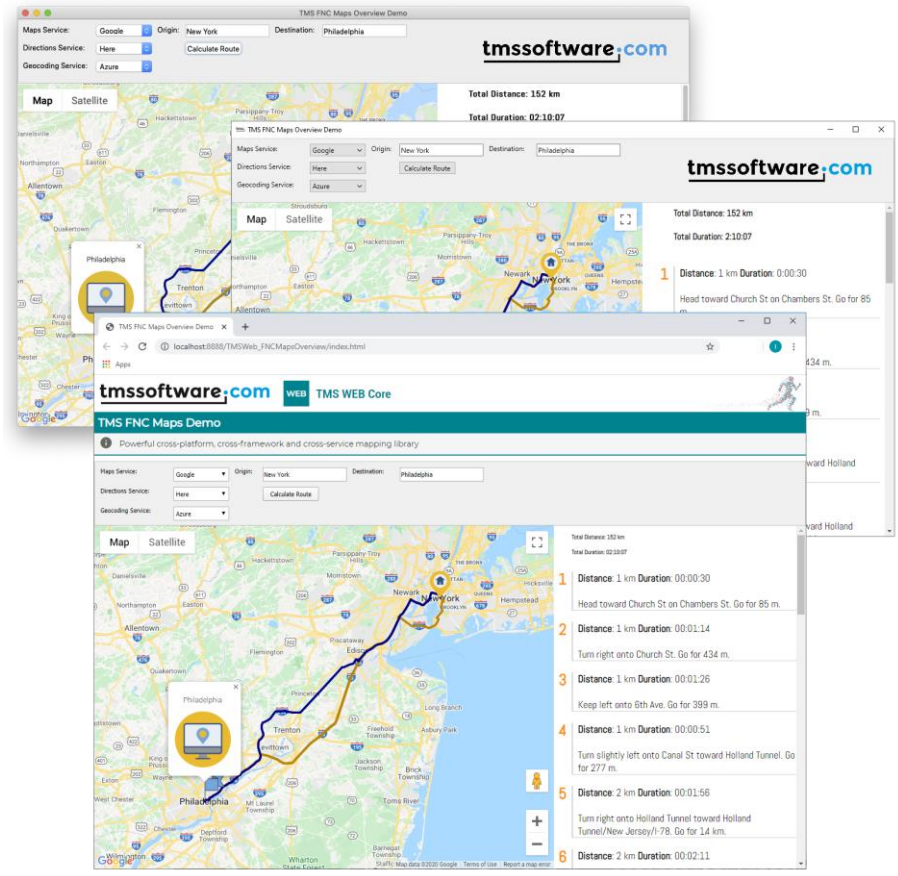
https://www.tmssoftware.com/site/cloudkey.asp

Click on the service you are using and follow the steps to obtain an API key. The API key that is requested can be used for all features that are available after installing TMS FNC Maps:

- TTMSFNCMaps (and descendants)
- TTMSFNCDirections
- TTMSFNCGeoCoding
- TTMSFNCLocation
- TTMSNFCElevation

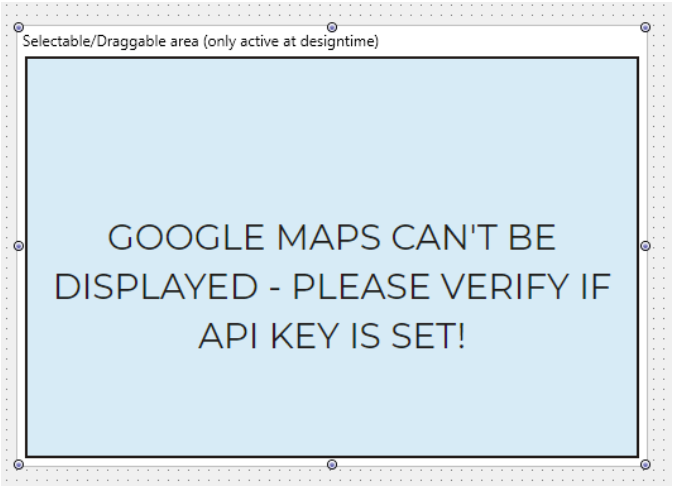
TTMSFNCMaps

TTMSFNCMaps is an abstract layer on top of the services that are listed under the “Description” chapter. It serves a way to display / manipulate and retrieve information of the chosen service in an abstract way, which means that whenever you are switching to another service, the code that was written will be 100% compatible and will behave exactly the same as the service you were originally been using, ofcourse under the disclaimer that the service does not change the underlying APIs.



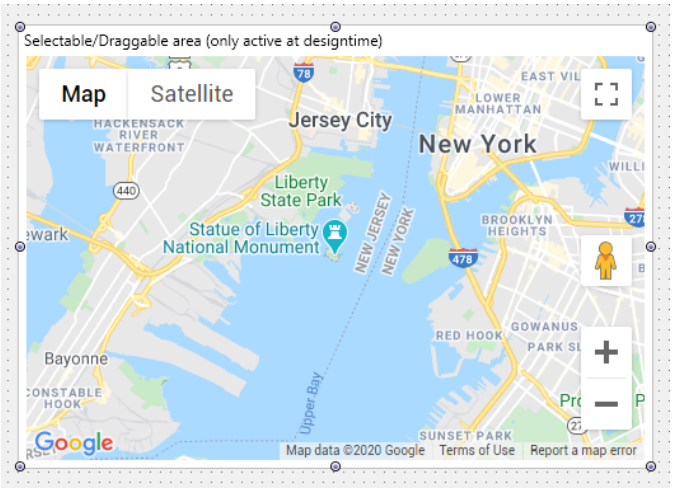
Initialization

When dropping an instance of TTMSFNCMaps (after successful installation of TMS FNC Core and TTMSFNCWebBrowser) on the form you'll notice the designtime message indicating that Google Maps can't be displayed because the API key is not set. Changing the Service property will show a different message depending on the chosen service. Please enter a value under the API key property after selecting the component at designtime, or specify an API key at runtime that matches the service selected under the Service property. Also note that the only selectable area (for moving, resizing) is the area at the top. The blue area is actually a real life mapping instance, so entering the API key at designtime will show you a live preview of the map.




```
procedure TForm1.FormCreate(Sender: TObject);
begin
  TMSFNCMaps1.APIKey := 'xxxxxxxxxxxxxxx';
end;
```

After setting the API key, you should see the map of the service of choice.



Dynamic switching

A feature of TTMSFNCMaps is dynamic switching. It allows switching to another service even though you have already added markers, polygons, polylines, ... The only settings that are not persisted are the default location and zoom which are re-initialized to the default values. Switching or initializing a specific service can be done with the following code:

```
procedure TForm1.ChangeService;
begin
  TMSFNCMaps1.Service := msBingMaps;
end;
```

Performance

When using the maps, adding markers, polyelements or changing existing properties of an object that has been added to the map it is always a good practice to wrap the code with BeginUpdate/EndUpdate. This ensures the JavaScript calls are bundled and this will lower execution times, which automatically increases performance. Below is a sample that demonstrates how to use BeginUpdate & EndUpdate calls for speeding up the process.

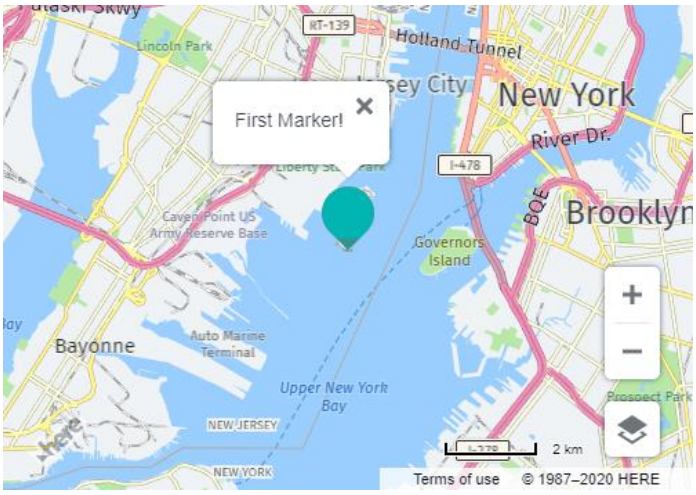
```
TMSFNCMaps1.BeginUpdate;
//add markers, polyelements, load GPX, GeoJSON file, …
TMSFNCMaps1.EndUpdate;
```

Markers

Markers identify a location on the map. Each service has its own marker icon that can (optionally) be changed. A marker is tied to a specific location (coordinate) on the map. The default location is the Statue of Liberty in New York (Latitude = 40.689247, Longitude = -74.044502). Below is a piece of sample code that adds a marker, and the result shown on the map. The marker class is TTMSFNCMapsMarker.

```
TMSFNCMaps1.BeginUpdate;
TMSFNCMaps1.AddMarker(DefaultCoordinate, 'First Marker!');
TMSFNCMaps1.EndUpdate;
```

The title of the marker can optionally be set, in this case the marker title is being shown as a hint in Google Maps. In Here Maps for example, the marker can be clicked when a Title is set which will show the value in a popup:



When a marker has been added, changing properties will automatically update the marker, such as changing the coordinate property to relocate the marker or specifying another title.

The marker icon can also be customized. Below is a sample that changes the default Google Maps icon to a custom icon.

```
TMSFNCMaps1.BeginUpdate;
TMSFNCMaps1.AddMarker(DefaultCoordinate, 'First Marker!',
'http://myIconURL.png');
TMSFNCMaps1.EndUpdate;
```

Please note that AddMarker is a helper function that accesses the Marker collection. The marker collection can also be accessed via

```
TMSFNCMaps1.Markers.Add;
```

PolyElements

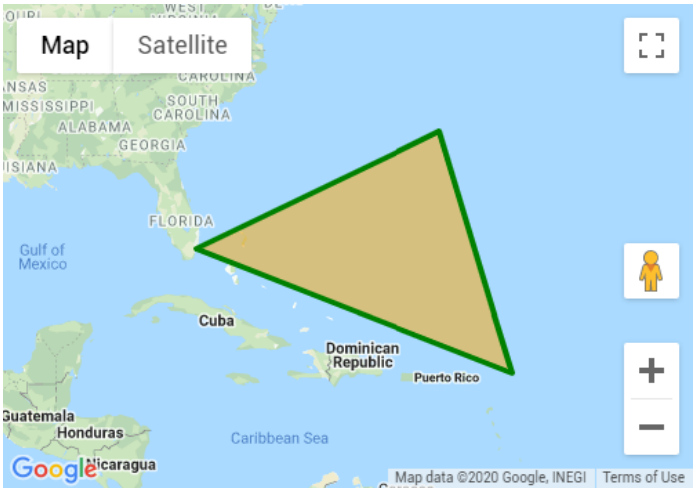
PolyElements can be added in the same way as adding Markers. PolyElements include one of the following types:

- TTMSFNCMapsPolyline

- TTMSFNCMapsPolygon
- TTMSFNCMapsRectangle
- TTMSFNCMapsCircle

Adding a polyline and polygon is done with AddPolyline and AddPolygon respectively. Both calls accept an array of coordinates (TTMSFNCMapsCoordinateRec). Both TTMSFNCMapsPolyline and TTMSFNCMapsPolygon have the ability to specify a stroke color, width and opacity. The TTMSFNCMapsPolygon has the ability to specify a fill color, width and opacity. The following sample demonstrates how to add a polygon that represents the Bermuda triangle. The code can also be replaced by AddPolyline, but without access to the Fill* properties.

```
procedure TForm1.AddMarkerToMap;
var
  arr: TTMSFNCMapsCoordinateRecArray;
  pg: TTMSFNCMapsPolygon;
begin
  SetLength(arr, 3);
  arr[0] := CreateCoordinate(25.789106, -80.226529);
  arr[1] := CreateCoordinate(18.4663188, -60.1057427);
  arr[2] := CreateCoordinate(32.294887, -64.781380);
  TMSFNCMaps1.BeginUpdate;
  pg := TMSFNCMaps1.AddPolygon(arr);
  pg.FillColor := gcOrange;
  pg.FillOpacity := 0.5;
  pg.StrokeColor := gcGreen;
  pg.StrokeWidth := 4;
  TMSFNCMaps1.EndUpdate;
end;
```



Adding a circle (TTMSFNCMapsCircle) is done by specifying a center coordinate and a radius (in meter). TTMSFNCMapsCircle inherits from TTMSFNCMapsPolygon and therefore also can specify fill and stroke properties.

Below is a sample that demonstrates how adding a circle is done with a radius of 200 km.

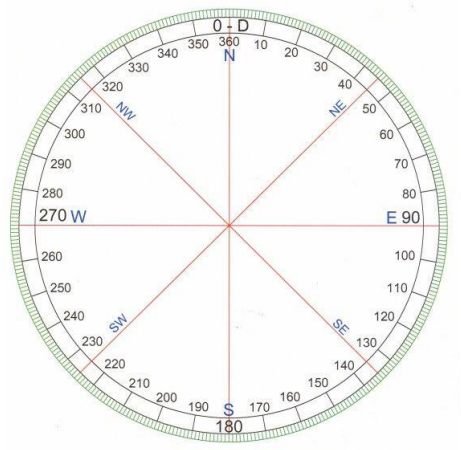
```
procedure TForm1.AddCircleToMap;
var
  c: TTMSFNCMapsCircle;
begin
  TMSFNCMaps1.BeginUpdate;
```

```
  c := TMSFNCMaps1.AddCircle(DefaultCoordinate, 200000);
  c.FillColor := gcOrange;
  c.FillOpacity := 0.5;
  c.StrokeColor := gcGreen;
  c.StrokeWidth := 4;
  TMSFNCMaps1.EndUpdate;
end;
```

Adding a rectangle (TTMSFNCMapsRectangle) is done with the AddRectangle call which accepts a bounds (TTMSFNCMapsBoundsRec). This is defined by a NorthEast and SouthWest coordinate. Below is a sample that demonstrates this.

```
procedure TForm1.AddRectangleToMap;
var
  c: TTMSFNCMapsRectangle;
  b: TTMSFNCMapsBoundsRec;
  ne, sw: TTMSFNCMapsCoordinateRec;
begin
  TMSFNCMaps1.BeginUpdate;
  ne := CalculateCoordinate(DefaultCoordinate, 45, 100000);
  sw := CalculateCoordinate(DefaultCoordinate, 225, 100000);
  b := CreateBounds(ne.Latitude, ne.Longitude, sw.Latitude, sw.Longitude);
  c := TMSFNCMaps1.AddRectangle(b);
  c.FillColor := gcOrange;
  c.FillOpacity := 0.5;
  c.StrokeColor := gcGreen;
  c.StrokeWidth := 4;
  TMSFNCMaps1.EndUpdate;
end;
```

There is a little more explanation necessary for this piece of code that adds a rectangle. The rectangle is 200km wide. The coordinates are calculated based on the center of the rectangle and the bearing (0-360).

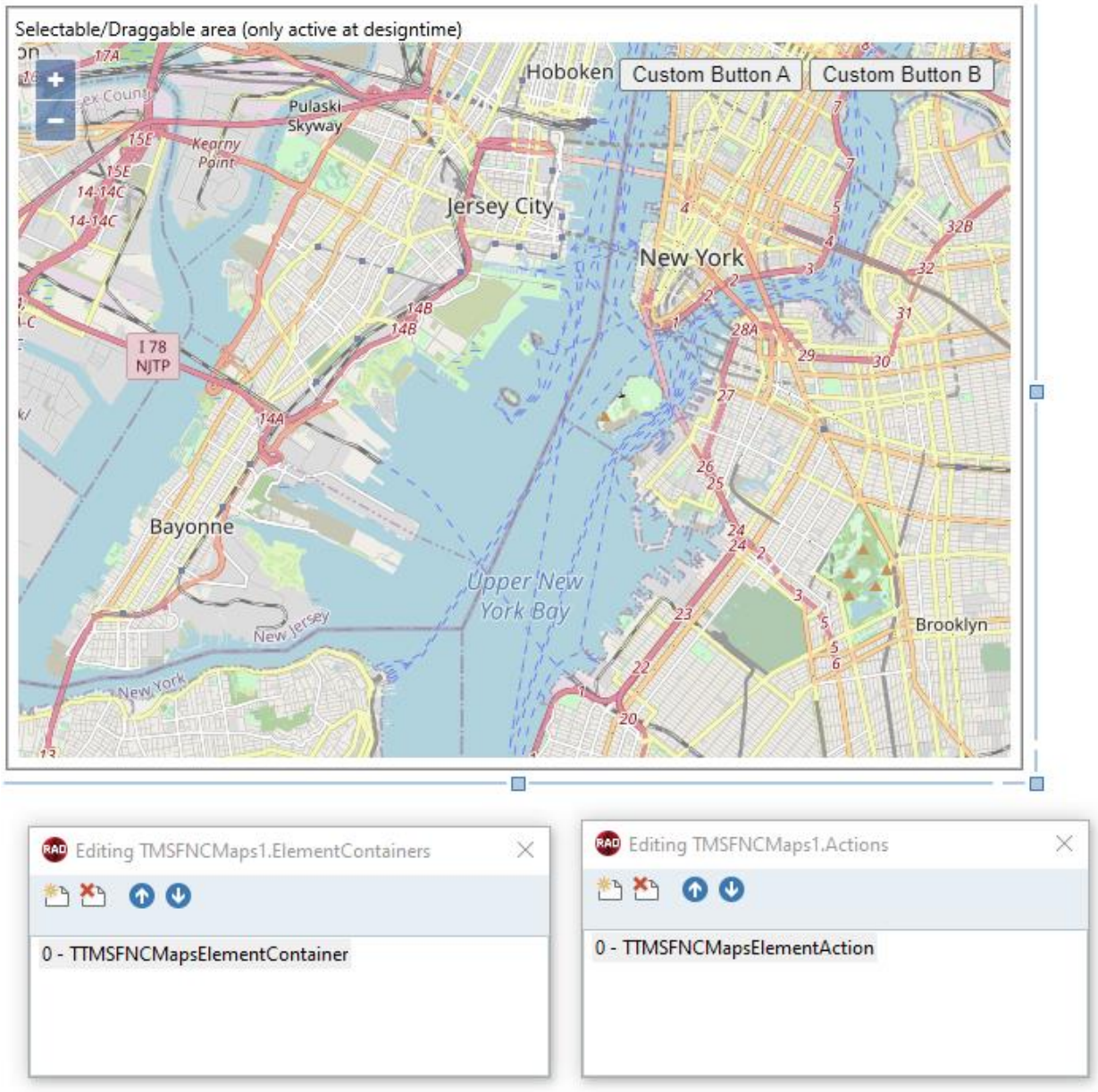


To calculate the northeast coordinate, we need to take the center coordinate and traval into the 45 degrees' direction. The parameter "Distance" is determining how far we actually travel. The CalculateCoordinate is a helper function in the *.TMSFNCMapsCommonTypes unit that calculates the coordinate based on these parameters. The same applies to the SouthWest coordinate, which lies in the 225 degrees' direction. Finally, we are able to create a bounds rectangle with these coordinate and plot it on the map.

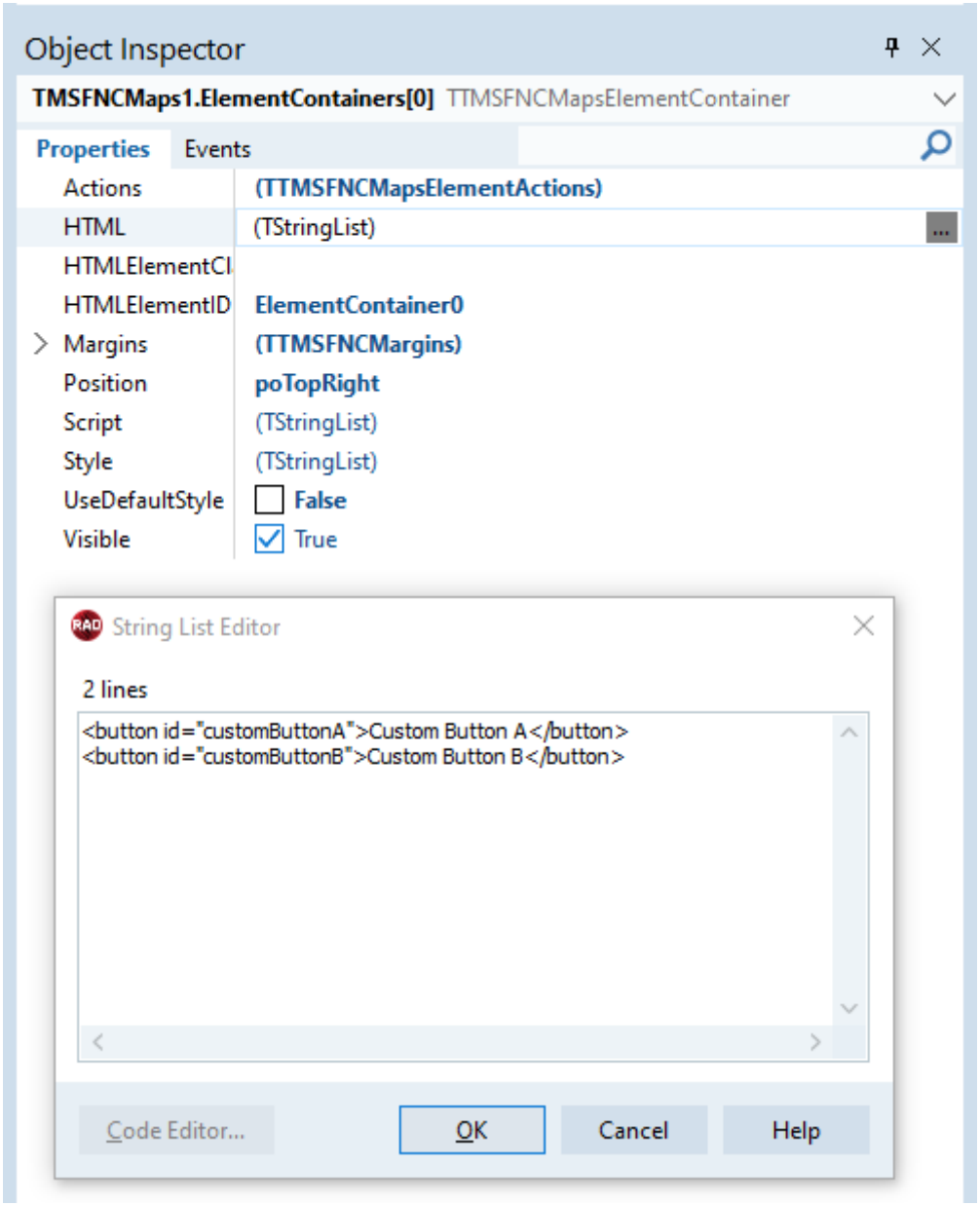
ElementContainers

Add custom HTML/CSS elements on top of the map with ElementContainers. Through interaction with the map, by using Actions, custom controls can be created. At runtime the content of the ElementContainers can be updated dynamically through JavaScript. ElementContainers are independent of the mapping service which means they will look and function identically regardless of your preferred mapping service. A preview is automatically available at design time to make editing the HTML/CSS elements as straightforward as possible.

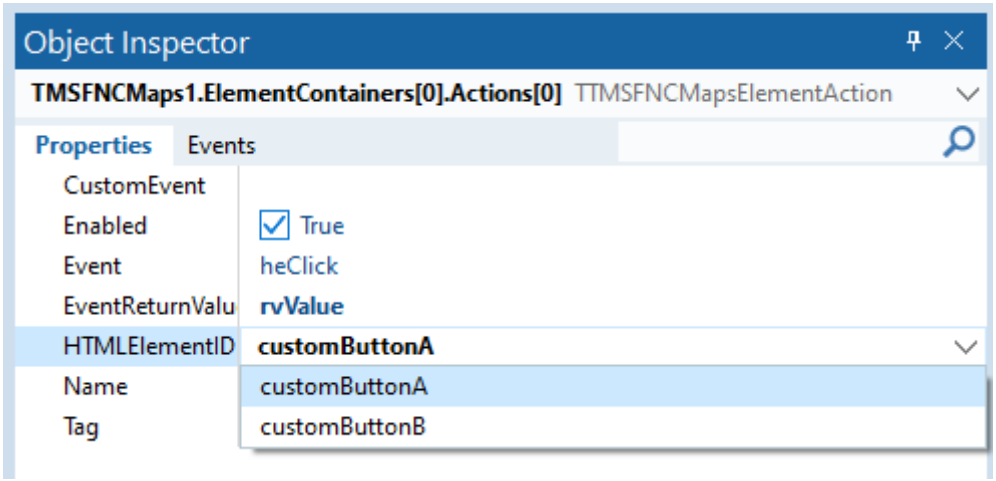
Actions



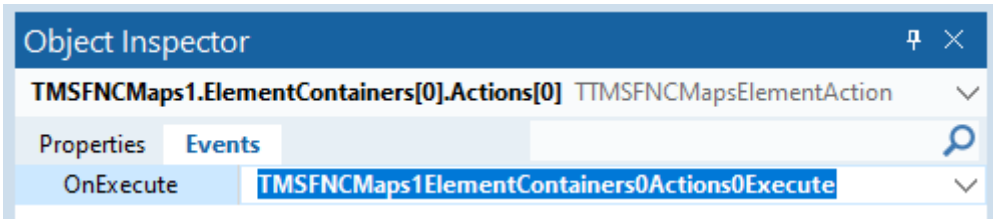
The Actions collection makes it possible to assign events to HTML objects in the ElementContainer based on their ID. The ID can be selected from the list of available IDs displayed on the HTMLElementID.



Common events like OnClick can be selected from the Event property list. If heCustom is selected, a custom event name can be entered in CustomEvent. The EventReturnValue can be used to define which value is returned in the Event parameter AEventData.CustomData.



The OnExecute event is used to assign an event handler.

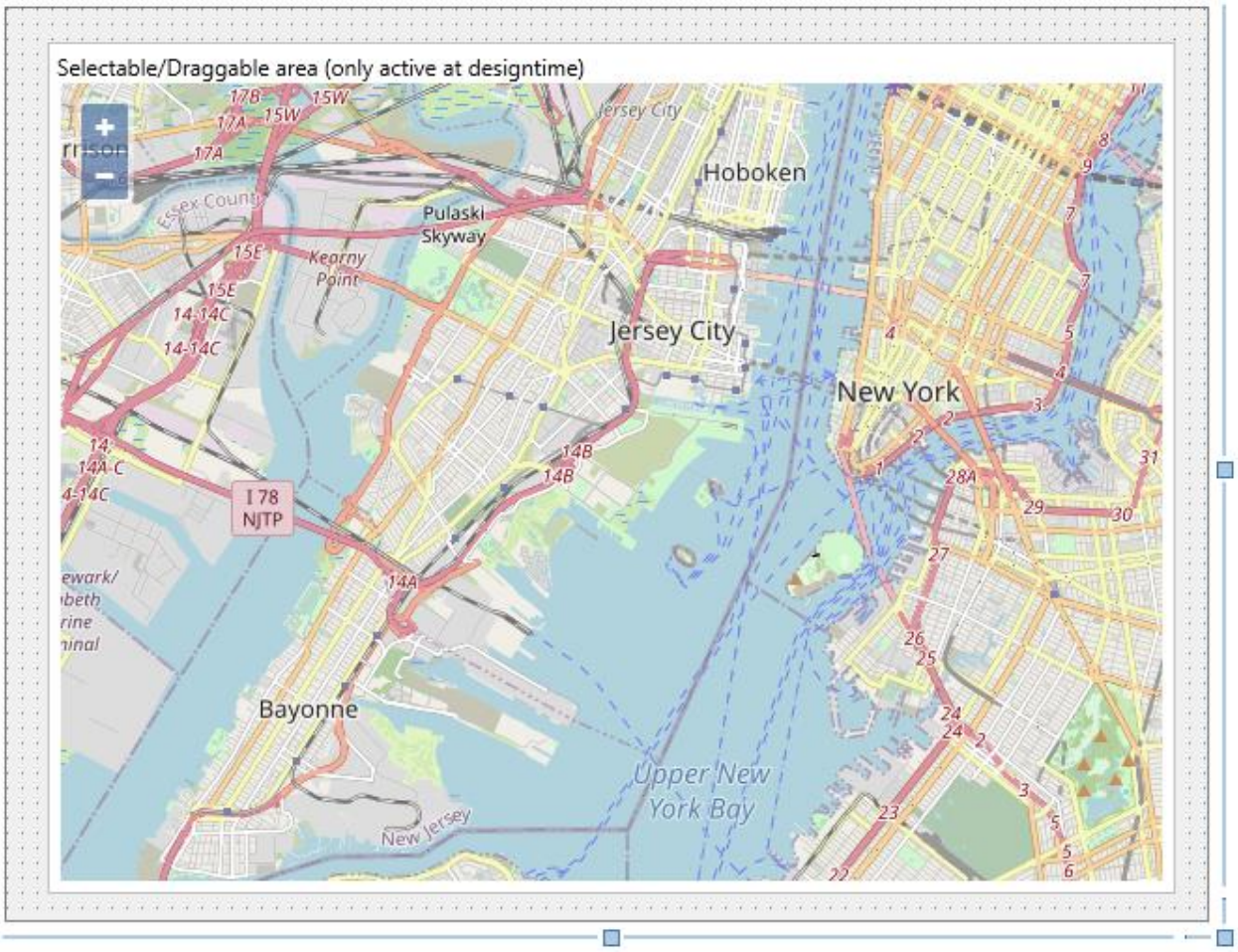



```
procedure TForm1.TMSFNCMaps1ElementContainers0Actions0Execute(Sender:
TObject;
  AEventData: TTMSFNCMapsEventData);
begin
  TTMSFNCUtils.Log(AEventData.CustomData);
end;
```

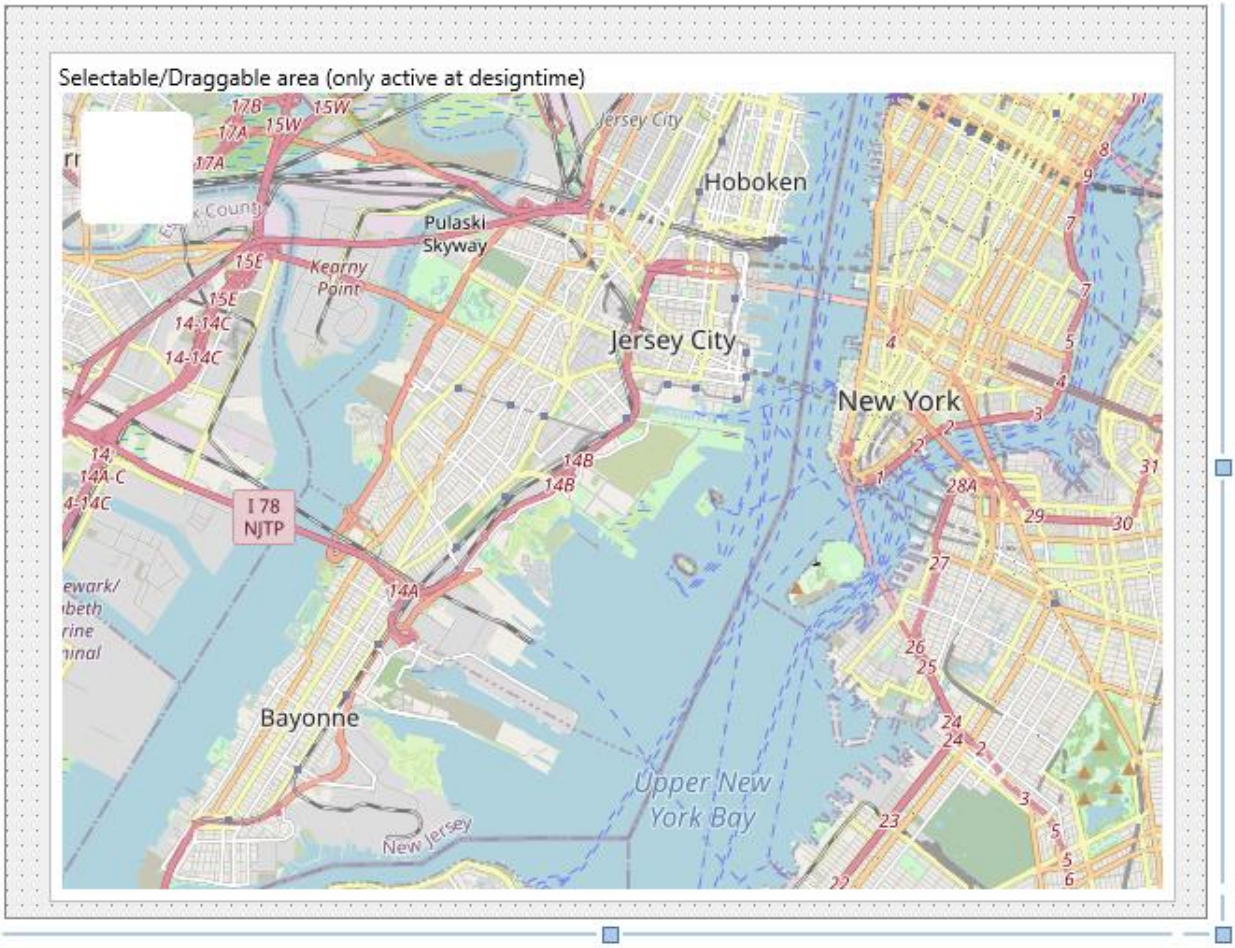
Tutorial

Creating a custom zoom slider control

1. Drop a TTMSFNCMaps on the form and select your preferred mapping service. We'll use msOpenLayers here so we don't need an API key.



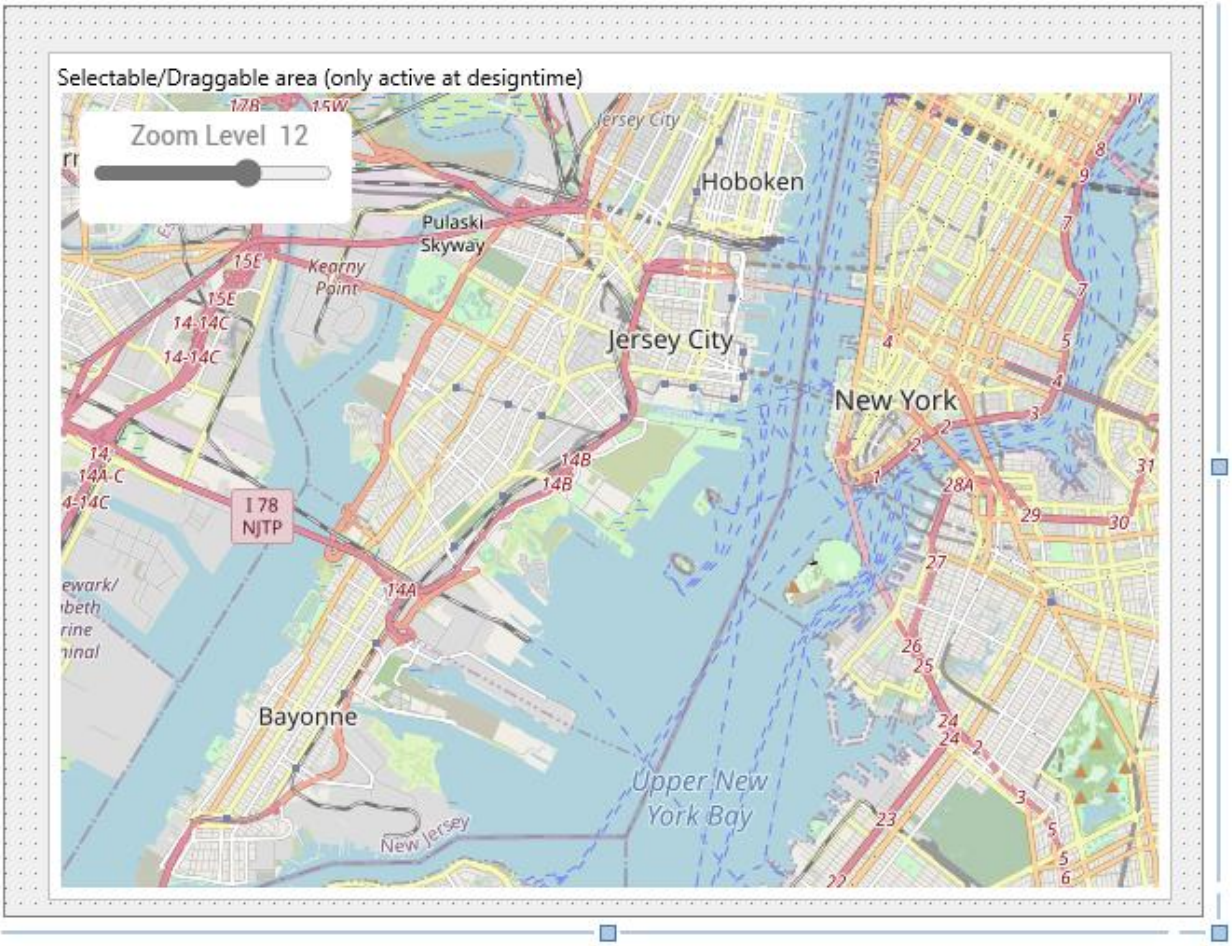
2. Add an item to the ElementContainers collection. An empty ElementContainer is displayed.



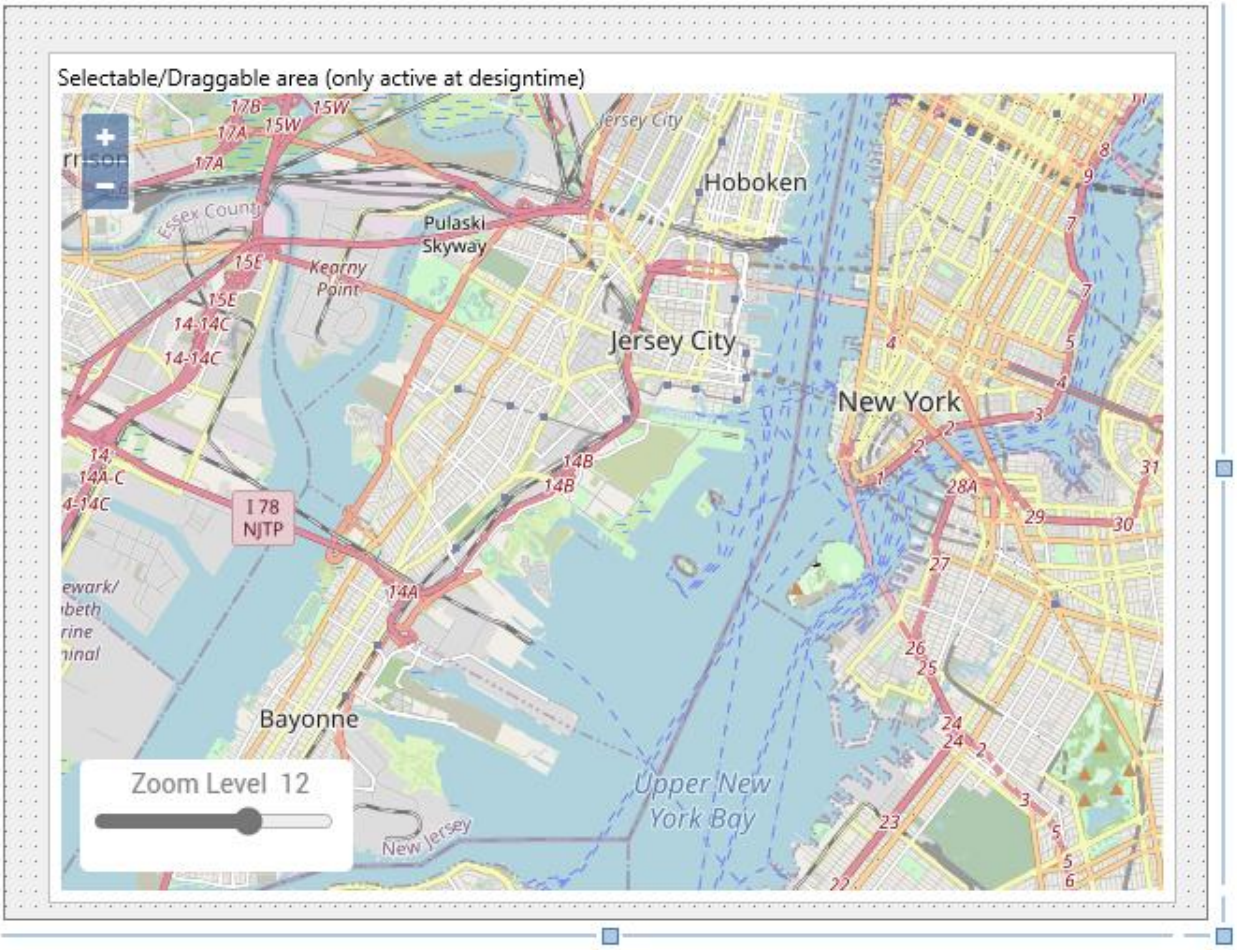
3. Add content to the HTML property.

   We'll add a title, a label and a slider element to the ElementContainer.

```
<div>
Zoom Level
<span id="customZoom1">12</span>
</div>
<div>
<input type="range" min="0" value="12" max="18" id="customRange1">
</div>
```

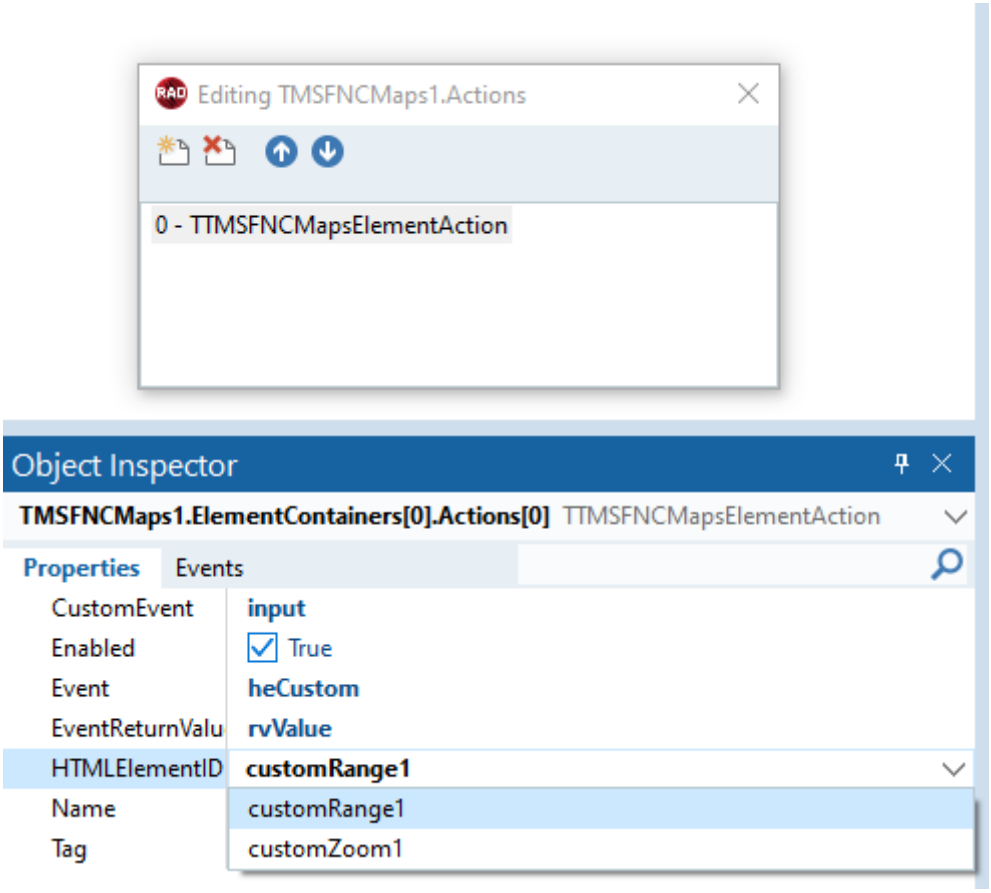


4. Position the container anywhere on the map with the Position and Margin properties. We'll place the container in the bottom left corner of the map: set Position to poBottomLeft.



5. Add an item to the Actions collection of the ElementContainer to interact with the map.

   Select the ID from the element to interact with from the HTMLElementID dropdown list. In this case pick “customRange1” to interact with the slider control. Set Event to heCustom and CustomEvent to input to catch the input event of the slider control. Set EventReturnValue to rvValue to retrieve the updated value of the slider control.



6. Assign the ElementAction's OnExecute event to add an event handler.

   The updated slider control value is returned as a string in the AEventData.CustomData property. We can use this value to update the map's zoom level with the code below:

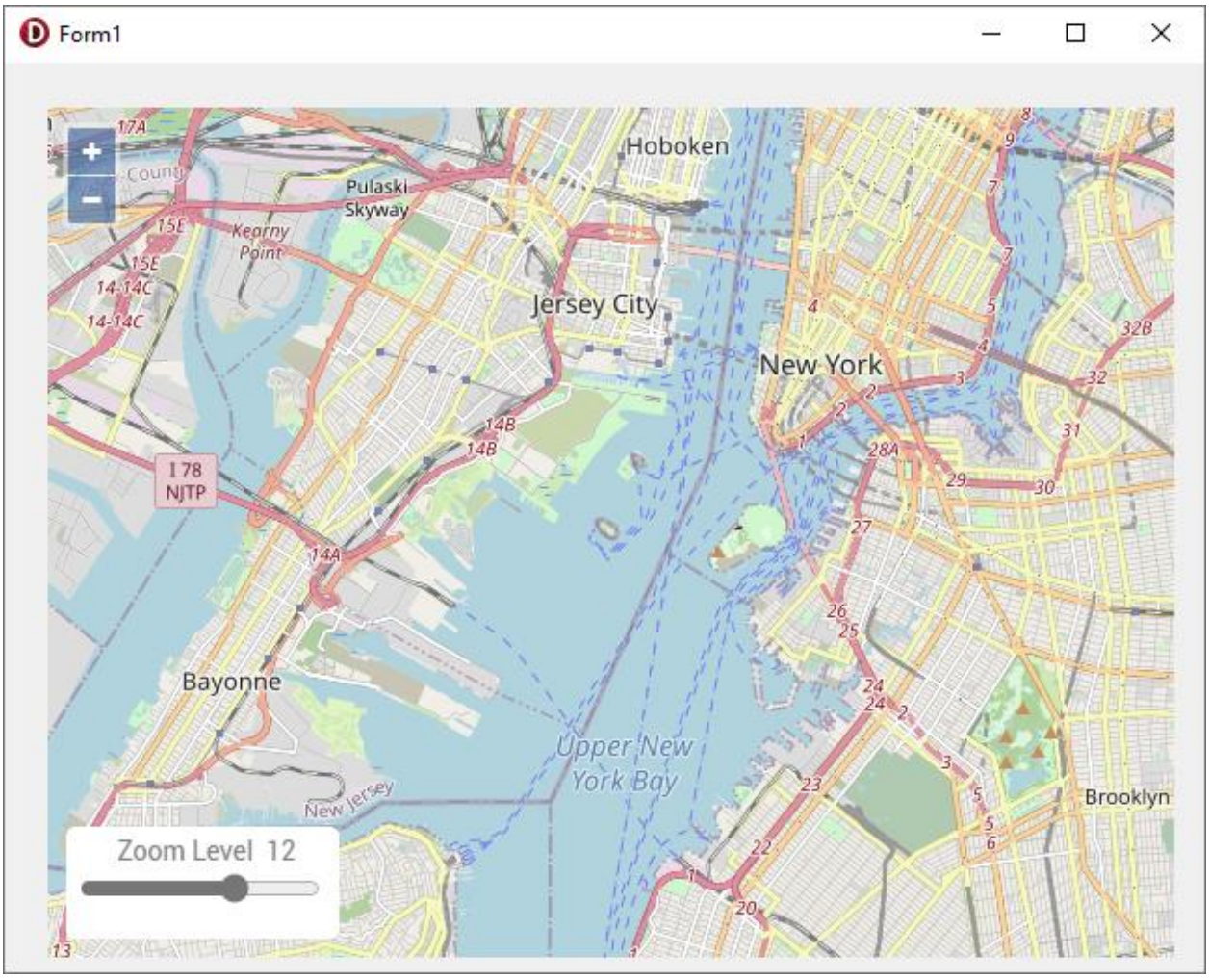
```
procedure TForm1.TMSFNCMaps1ElementContainers0Actions0Execute(Sender:
TObject;
  AEventData: TTMSFNCMapsEventData);
var
  f: Double;
  v: string;
begin
  f := 0;
  v := StringReplace(AEventData.CustomData, '"', '', [rfReplaceAll]);
  if TryStrToFloatDot(v, f) then
   TMSFNCMaps1.SetZoomLevel(f);
end;
```

7. Assign the TTMSFNCMaps's OnZoomChanged event to interact with the ElementContainer.

   When the map's zoom level changes, get the new zoom level and update the label and slider control in the ElementContainer.

```
procedure TForm1.TMSFNCMaps1ZoomChanged(Sender: TObject;
  AEventData: TTMSFNCMapsEventData);
begin
  TMSFNCMaps1.GetZoomLevel;
end;

procedure TForm1.TMSFNCMaps1GetZoomLevel(Sender: TObject; AZoomLevel:
Double);
begin

TMSFNCMaps1.ExecuteJavaScript('document.getElementById("customRange1"
).value = ' + FloatToStrDot(AZoomLevel));

TMSFNCMaps1.ExecuteJavaScript('document.getElementById("customZoom1")
.innerText = ' + FloatToStrDot(Round(AZoomLevel)));
end;
```

The custom control is now ready to be used:

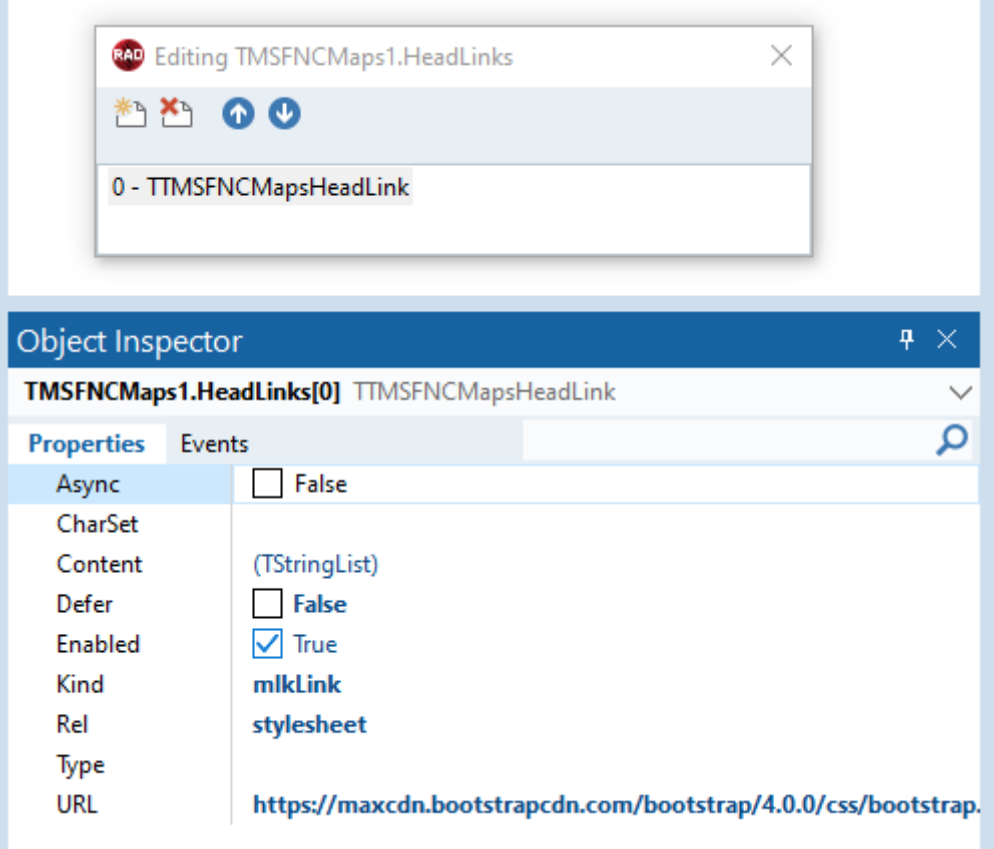


Add Bootstrap styling

Using Bootstrap is a straightforward way to improve the visuals of HTML elements. With a few adjustments Bootstrap styling can be added to ElementContainers.

1. Include the Bootstrap CSS library via the HeadLinks collection.

   Add an item to the HeadLinks collection; set Kind to mlkLink, URL to "https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css" and Rel to "stylesheet".

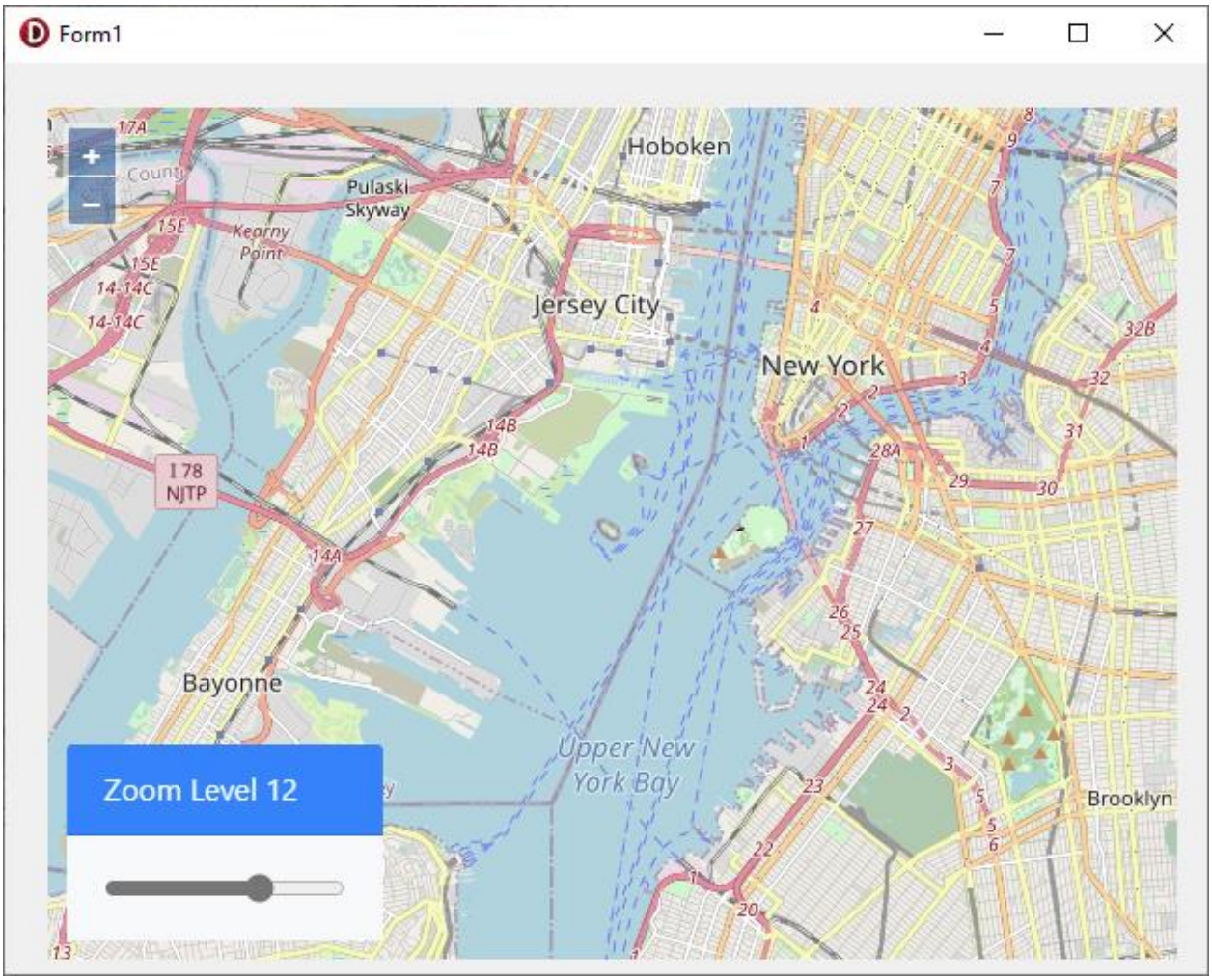


2. Add Bootstrap HTML classes to the ElementContainer's HTML.

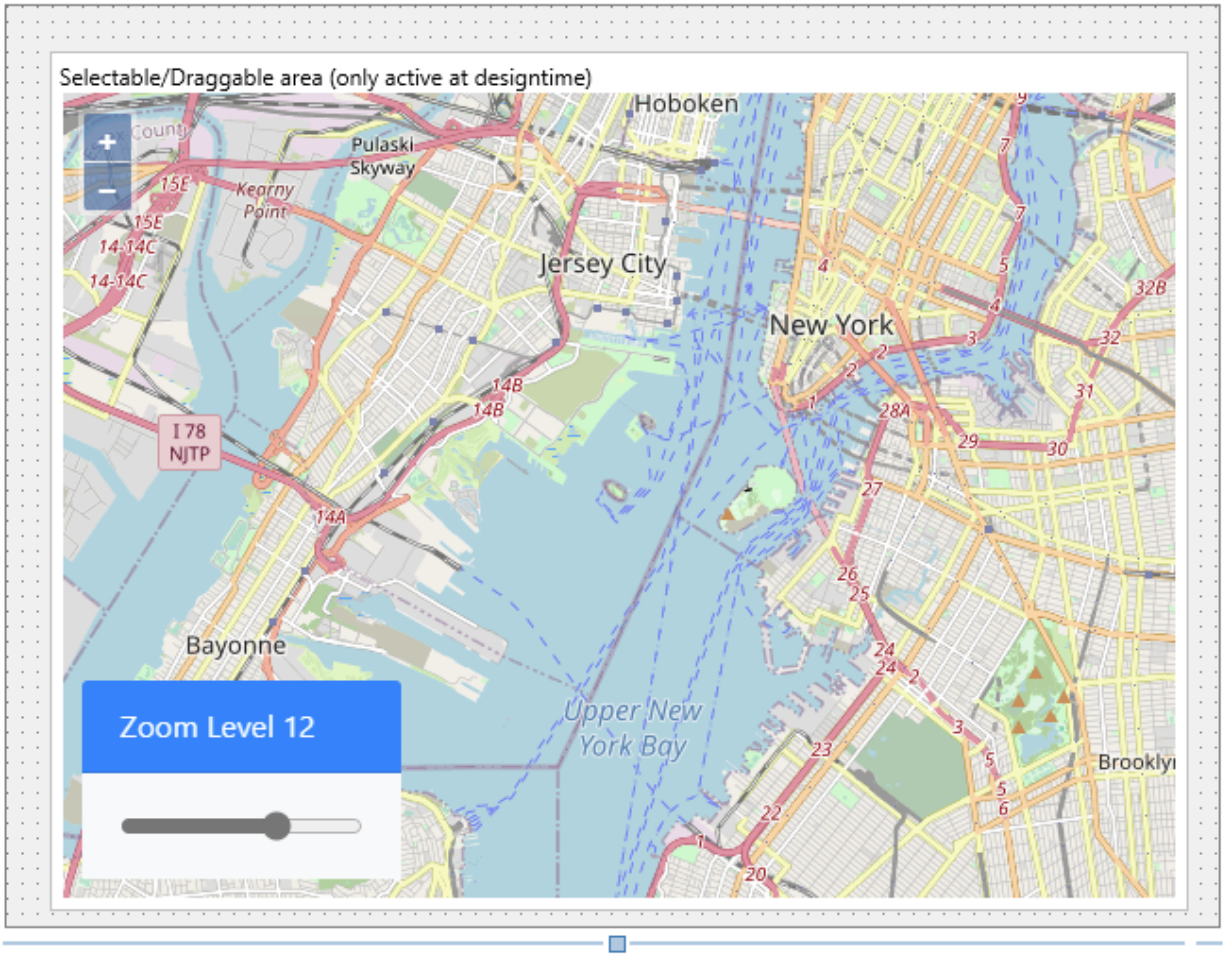
   Change the HTML value to:

```
<div for="customRange1" class="card-header bg-primary text-white">
Zoom Level
<span class="card-label" id="customZoom1">12</span>
</div>
<div class="card-body bg-light">
<input type="range" class="form-range" min="0" value="12" max="18"
id="customRange1">
</div>
```

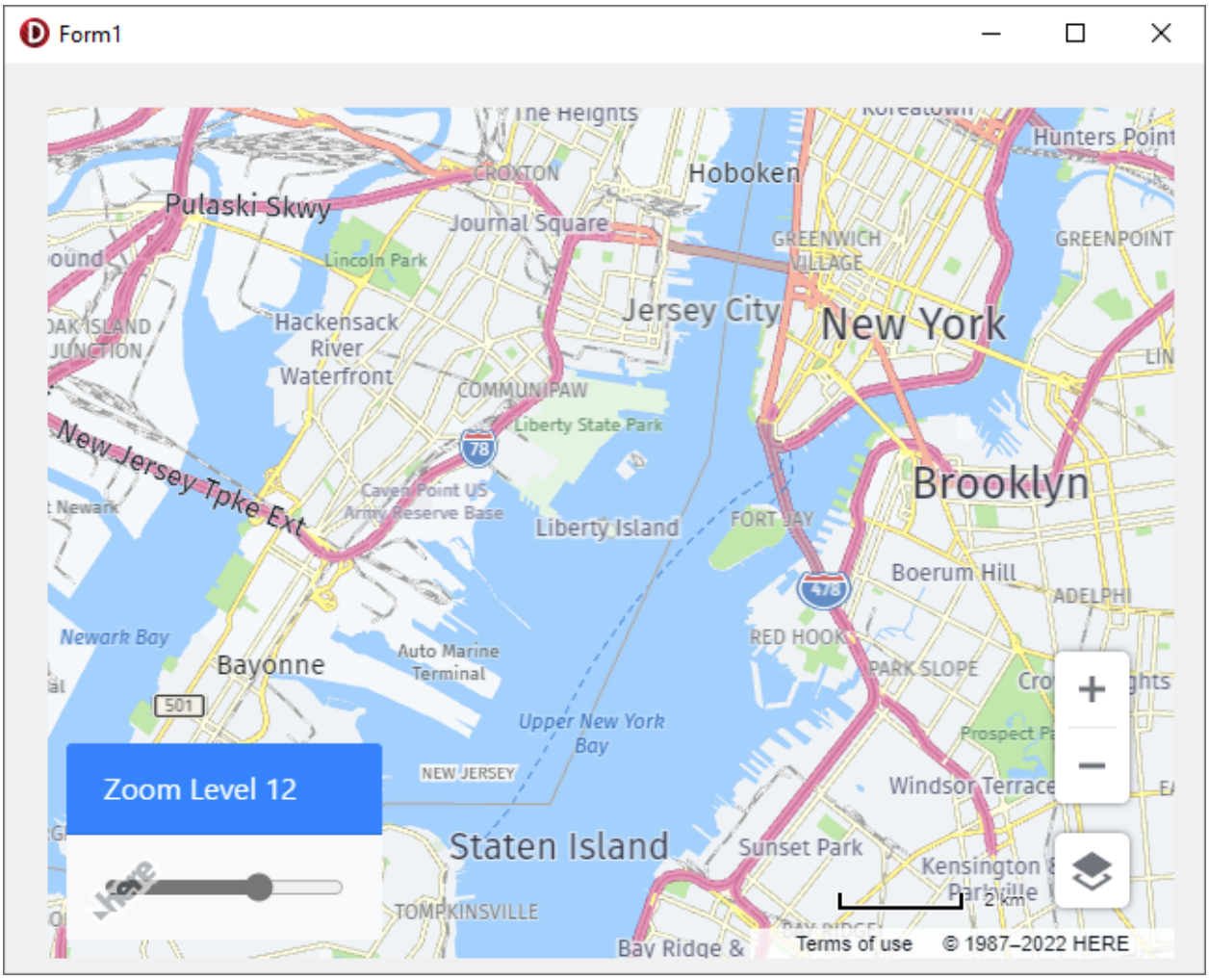
   Then set HTMLElementClassName to "card" and UseDefaulStyle to "False".
   Suddenly the custom control looks a lot prettier.



Notice that the ElementContainer's content with Bootstrap styling is fully visible at design-time as well.



Because ElementContainers are completely independent from the mapping service, the result looks exactly the same even when switching to a different service. The screenshot below shows the same ElementContainer with Service set to msHere.



Of course, the same result can also be achieved in code.

Example

```
procedure TForm1.FormCreate(Sender: TObject);
var
  ElementContainer: TTMSFNCMapsElementContainer;
  ElementAction: TTMSFNCMapsElementAction;
  html: TStringList;
begin
  TMSFNCMaps1.ClearElementContainers;
  TMSFNCMaps1.ClearHeadLinks;

  TMSFNCMaps1.BeginUpdate;


TMSFNCMaps1.AddHeadLink('https://maxcdn.bootstrapcdn.com/bootstrap/4.
0.0/css/bootstrap.min.css', mlkLink, 'stylesheet');

  html := TStringList.Create;
```

```
  html.Add('<div for="customRange1" class="card-header bg-primary
text-white">'
    + 'Zoom Level '
    + '<span class="card-label" id="customZoom1">12</span>'
    +' </div>'
    +' <div class="card-body bg-light">'
    +' <input type="range" class="form-range" min="0" value="12"
max="18" id="customRange1">'
    +'</div>');

  ElementContainer := TMSFNCMaps1.AddElementContainer(html);
  ElementContainer.HTMLElementClassName := 'card';
  ElementContainer.Position := poBottomRight;
  ElementContainer.Margins.Bottom := 15;
  ElementContainer.UseDefaultStyle := False;

  ElementAction := ElementContainer.AddAction('customRange1',
heCustom);
  ElementAction.CustomEvent := 'input';
  ElementAction.EventReturnValue := rvValue;
  ElementAction.OnExecute :=
TMSFNCMaps1ElementContainers0Actions0Execute;

  html.Free;

  TMSFNCMaps1.EndUpdate;
end;
```

Headlinks

The HeadLinks collection can be used to dynamically add custom CSS and JavaScript libraries to the TTMSNFNCMaps component. This feature can be used in combination with ElementContainers.

Example
Adding the Bootstrap and jQuery libraries via the HeadLinks collection:

```
TMSFNCMaps1.AddHeadLink('https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/cs
s/bootstrap.min.css', mlkLink, 'stylesheet');
TMSFNCMaps1.AddHeadLink(' https://code.jquery.com/jquery-3.2.1.slim.min.js',
mlkScript);
TMSFNCMaps1.AddHeadLink('https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js
/bootstrap.min.js', mlkScript);
```

Loading GPX/GeoJSON

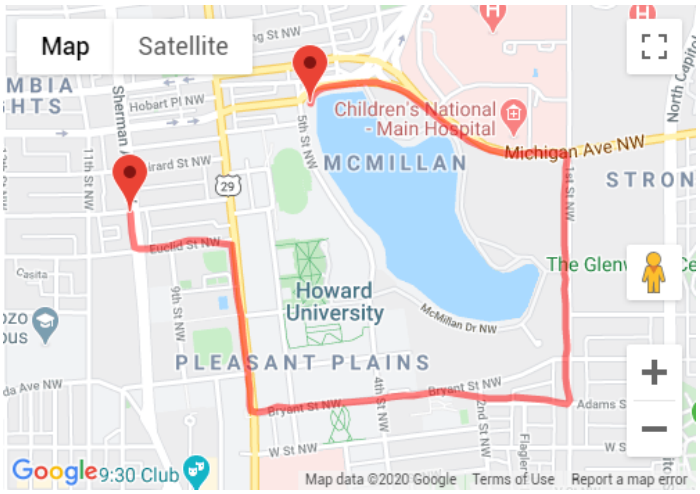
Loading GPX/ GeoJSON is done with a single call. Specify the GPX/GeoJSON file that you wish to load and the map will render the polyelements/lines that are retrieved inside the GPX/GeoJSON file. The optional parameters are to AutoDisplay the data that is loaded and to automatically zoom to the bounds of the data that has been entered. The AutoDisplay and ZoomToBounds parameters are both true by default. Below is a sample with a GPX file.

GPX

```
TMSFNCMaps1.BeginUpdate;
TMSFNCMaps1.LoadGPXFromFile('run.gpx');
TMSFNCMaps1.EndUpdate;
```

Changing the line that has automatically been added is as simply as taking the last element from the Polylines collection.

```
procedure TForm1.LoadGPX;
var
  pl: TTMSFNCMapsPolyline;
begin
  TMSFNCMaps1.BeginUpdate;
  TMSFNCMaps1.LoadGPXFromFile('run.gpx');
  pl := TMSFNCMaps1.Polylines[TMSFNCMaps1.Polylines.Count - 1];
  pl.StrokeColor := gcRed;
  pl.StrokeWidth := 5;
  pl.StrokeOpacity := 0.5;

  TMSFNCMaps1.AddMarker(pl.Coordinates[0].Coordinate.ToRec);
  TMSFNCMaps1.AddMarker(pl.Coordinates[pl.Coordinates.Count -
1].Coordinate.ToRec);
  TMSFNCMaps1.EndUpdate;
end;
```



For each Track found in the GPX file, the OnCreateGPXTrack event is triggered and for each Coordinate found, the OnCreateGPXSegment event.
The OnCreateGPXTrack AEventData parameter contains Data.Name/Segments information as well as the full XML node, so if extra data is available this can be parsed manually.
The OnCreateGPXSegment AEventData parameter contains Data.Longitude/Latitude/Elevation/TimeStamp information as well as the full XML node, so if extra data is available this can be parsed manually.

Examples:

- **Log standard gpx data**

```
procedure TForm1.TMSFNCMaps1CreateGPXSegment(Sender: TObject;
  AEventData: TTMSFNCMapsGPXSegmentEventData);
begin
  TTMSFNCUtils.Log(AEventData.Node.GetXML);
end;

procedure TForm1.TMSFNCMaps1CreateGPXSegment(Sender: TObject;
  AEventData: TTMSFNCMapsGPXSegmentEventData);
begin
  TTMSFNCUtils.Log('Lat:Lon: ' + FloatToStr(AEventData.Data.Longitude) + '
: ' + FloatToStr(AEventData.Data.Longitude)
    + ' Height: ' + FloatToStr(AEventData.Data.Elevation) + ' TimeStamp: '
+ DateTimeToStr(AEventData.Data.TimeStamp));
end;
```

```
Debug Output: Lat:Lon: 9.961047 : 9.961047 Height: 68.471 TimeStamp: 16/08/2020 14:55:42
Debug Output: Lat:Lon: 9.960927 : 9.960927 Height: 68.215 TimeStamp: 16/08/2020 14:55:44
```

- **Parse extension data and display it on the map**

If a given gpx file contains extra data beyond the standard available data, this can be parsed manually through AEventData.Node.
In this example the gpx file contains a heartrate value for each coordinate. The heartrate is extracted from the XML and a marker is added on the map along the route each time the value is above 190.

Sample gpx data:

```
<trkpt lat="50.7915" lon="3.1208">
  <ele>49</ele>
  <time>2020-04-07T13:32:17.000Z</time>
  <extensions>
    <hr>174</hr>
  </extensions>
</trkpt>
```

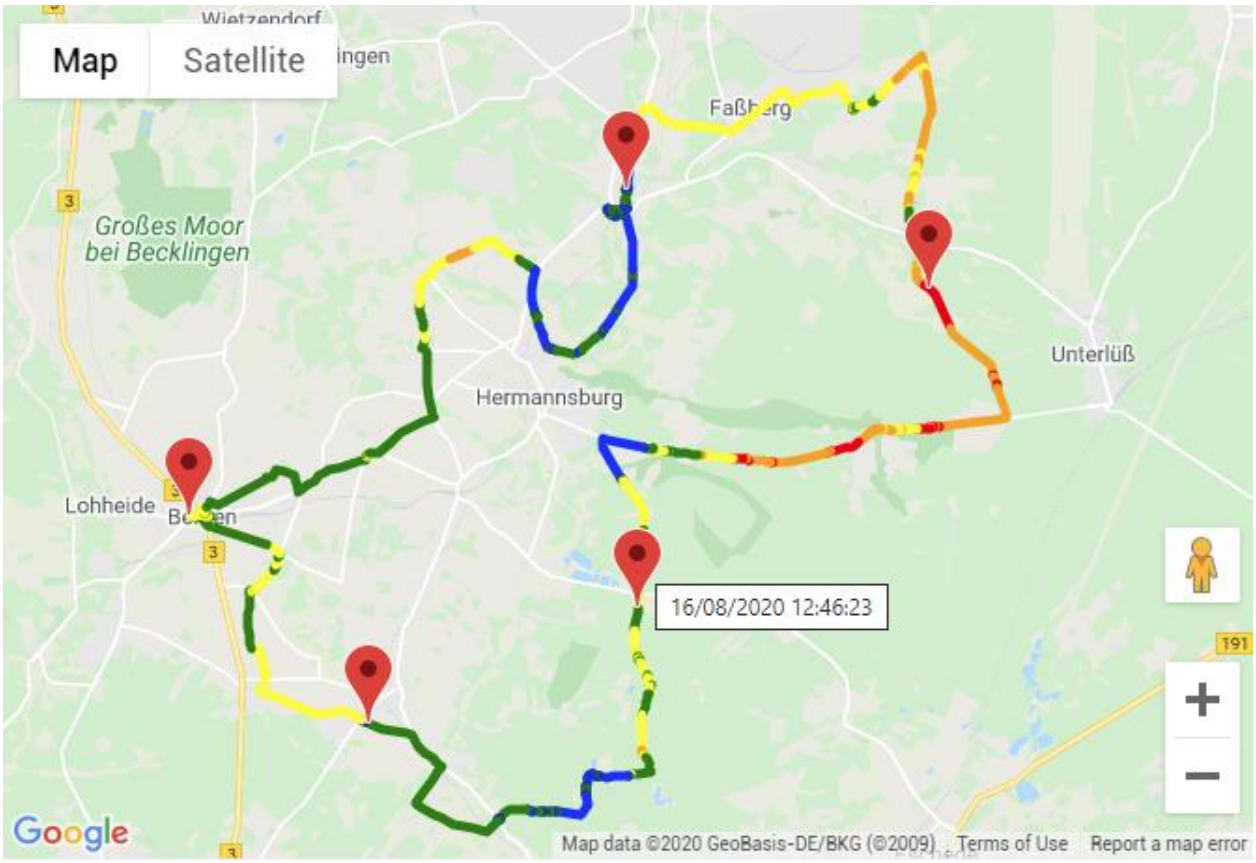
Code:

```
procedure TForm1.TMSFNCMaps1CreateGPXSegment(Sender: TObject;
  AEventData: TTMSFNCMapsGPXSegmentEventData);
var
  HeartRate: Integer;
begin
  HeartRate := StrToInt(NodeToString(FindNode(FindNode(AEventData.Node,
'extensions'), 'hr')));

  if HeartRate > 190 then
    TMSFNCMaps1.AddMarker(CreateCoordinate(AEventData.Data.Latitude,
AEventData.Data.Longitude), IntToStr(HeartRate));
end;
```

If a GPX file contains elevation and/or timestamp data, this data will also be available in the result of the LoadGPXFromFile call.

Elevation data can optionally be displayed with the ADisplayElevation parameter. This is achieved by adding multiple color coded polylines to the map. The colors to use can be provided in the AElevationColors parameter. If a single color is provided, the polyline will be displayed darker for lower parts and lighter for higher parts of the path. If no colors are provided, the AStrokeColor is used. If no AStrokeColor is provided, the default color is used.

Timestamp data can optionally be displayed through markers along the polyline path with the ADisplayTimeStamp parameter. To limit the amount of markers, the AMinSecondsBetweenTimeStamps can be used to set the minimum time between displayed markers.



GeoJSON

GeoJSON has multiple types of data, called “Features”, such as Point, Line, Polygon, etc … Below is a sample of a GeoJSON that is being loaded in the TTMSFNCMaps instance.

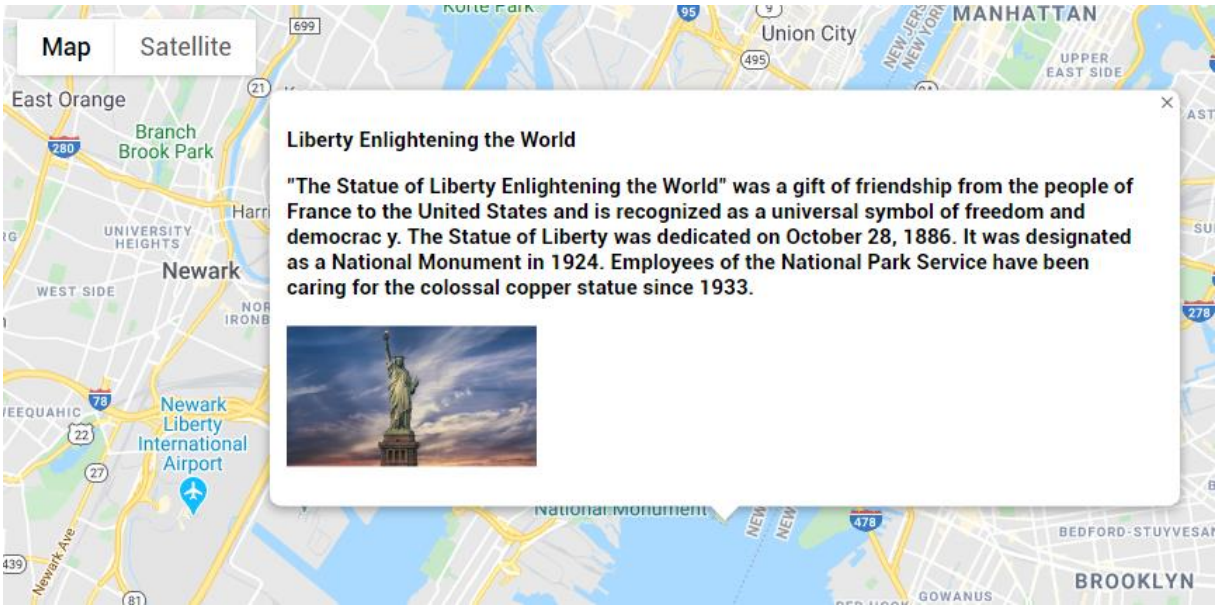
```
procedure TForm1.LoadGeoJSON;
var
  pl: TTMSFNCMapsPolygon;
  I: Integer;
begin
  TMSFNCMaps1.BeginUpdate;
  TMSFNCMaps1.LoadGeoJSONFromFile('data.geojson');
  for I := 0 to TMSFNCMaps1.Polygons.Count - 1 do
  begin
    pl := TMSFNCMaps1.Polygons[I];
    pl.FillColor := gcBlue;
    pl.FillOpacity := 0.2;
    pl.StrokeColor := gcBlue;
    pl.StrokeOpacity := 0.5;
  end;
  TMSFNCMaps1.EndUpdate;
end;
```


Map
Satellite
Norway
Baltic Sea
Estonia
Latvia
Lithuania
North Sea
Denmark
United Kingdom
Ireland
London
Netherlands
Berlin
Germany
Belgium
Paris
Prague
Czechia
Vienna
Slovakia
Austria
Hungary
France
Romania
Google
Map data ©2020 Google, ORION-ME
Terms of Use

Popups

Popups are informative messages that can be shown anywhere on the map. They are tied to a specific coordinate and can be shown and hidden programmatically in various cases, such as clicking on a marker, hovering over a circle, etc… A popup is capable of displaying HTML formatted text. Below is a sample that shows how to display a popup at the default coordinate with information on the Statue of Liberty in New York.

```
procedure TForm1.AddPopupToMap;
const
   s = '<div width="300"><h3>Liberty Enlightening the World<h3>' + LB +
   '"The Statue of Liberty Enlightening the World" was a gift of friendship
from the people of ' + LB +
   'France to the United States and is recognized as a universal symbol of
freedom and democrac' + LB +
   'y. The Statue of Liberty was dedicated on October 28, 1886.  It was
designated as a National' + LB +
   ' Monument in 1924.  Employees of the National Park Service have been
caring for the colossal copper statue since 1933.<br/><br/>' + LB +
   '<img height="100"
src="http://myserver.com/images/StatueOfLiberty.jpg"></img></div>';
begin
  TMSFNCMaps1.ShowPopup(DefaultCoordinate, s)
end;
```



Optional parameters are to offset the popup in pixels based on the default coordinate. When showing a popup, the return value is an ID, which can be used to close it with the ClosePopup method. The close all popups that are opened in the map, use CloseAllPopups;

Locale

The map is capable of changing the language that is displayed at some elements such as the maptype and the street/city names. Not all maps support a full translation and not all maps support every language. Please look up the languages that are supported depending on the chosen service. The way the map language is translated is based on the Locale string property which is set to en-US by default. A couple of examples of locales are:

en-US (English - United States)
en-GB (English - Great Britain)
fr-FR (French - France)
nl-NL (Dutch - Netherlands)
it-IT (Italian - Italy)
pt-BR (Portuguese - Brazil)

Applying this to the map is as simple as assigning a value to the Options.Locale property:

```
TMSFNCMaps1.Options.Locale := 'pt-BR';
TMSFNCMaps1.ReInitialize;
```

Note that a ReInitialize call is necessary to make sure the map is properly initialized.

Interaction

The map supports panning, zooming changing map type, scrolling with the mouse or keyboard and many more. Also programmatically, you are able to go to a specific location, zoom to a specific map bounds. Below is a list of methods that can be used to programmatically interact with the map.

```
TMSFNCMaps1.ZoomToBounds
```

The ZoomToBounds method has a couple of overloads to specify an array of coordinates, a bounds record, or a northeast/southwest coordinate. ZoomToBounds will automatically navigate to a specific area determined by the northeast and southwest coordinates either coming directly from the bounds parameter or automatically calculated based on an array of coordinates.

```
TMSFNCMaps1.SetCenterCoordinate
```

The SetCenterCoordinate will navigate to a specific location on the map without changing the zoom level.

```
TMSFNCMaps1.SetZoomLevel
```

The SetZoomLevel method can be used to set a specific zoom level at the current location on the map.

Some maps expose a zoom control and/or maptype control that allows changing the zoom level or switch to another map type such as a night mode or a satellite view depending on the chosen service. The properties to enable/disable these controls are available under Options. By default, the values are true for both properties. Below is a sample what happens when setting them to false like the code below.

```
TMSFNCMaps1.Options.ShowMapTypeControl := false;
TMSFNCMaps1.Options.ShowZoomControl := false;
```


```
TMSFNCMaps1.GetLatLonToXY
TMSFNCMaps1.GetCoordinateToXY

TMSFNCMaps1.GetXYToLatLon
TMSFNCMaps1.GetXYToCoordinate
```

Converts a given latitude and longitude coordinate to an XY coordinate or vice versa.

Note: Only supported for Google, Here, MapBox, MapKit. When called from a map event (i.e. OnMapClick), threading is required. See the Threading topic for further information.

Customization

Each service is implemented in a specific way so the polyelements, markers, rectangles, circles and every other aspect of the abstract TTMSFNCMaps class is working as expected even when switching to another service. If you are missing a particular feature/event or you want to change existing features, you have access to a series of customization events that allow you to change what happens. Through JavaScript there are a series of constants defined to get access to objects inside

JavaScript. A complete list of variables is found in the *.TMSFNCMaps.pas unit below the version number. Below is a list of events that allow customization.

`OnCustomizeCSS`
Event to customize the CSS, style information that is tied to the map, or to other objects in the map such as popups.

`OnCustomizeJavaScript`
Event that is called to completely customize all JavaScript that is used to render the map and add/update markers, polyelements and many more.

`OnCustomizeMap`
Event called when the map is being loaded. Additional map settings can be added here. The map identifier that can be used is the MAPVAR constant defined in the TMSFNCMaps unit.

`OnCustomizeGlobalVariables`
Event called when global variables are defined, which can be accessed from each JavaScript call that is available to the map.

`OnCustomizeMarker`
Event called when a marker is added or updated. This event can be used to customize markers. The variable that is initialized when working with markers is MARKERVAR.

`OnCustomizePopup`
Event called when a popup is added. This event can be used to customize popups. The variable that is initialized when working with popups is POPUPVAR.

`OnCustomizeOptions`
Event called when map options are changed such as the showing or hiding the zoom control or map type control. Add any options you want to change to the map in this event.

`OnCustomizePolyElement`
Event called when a polyelement is added (polyline, polygon, circle, rectangle). This event can be used to customize a polyelement. The variable that is initialized when working with popups is POLYELEMENTVAR.

`OnGetDefaultHTMLMessage`
Event called when the map is being loaded but no API key is set. This message is to inform users about that an API key is missing by default. This message can be changed through this event.

`OnCustomEvent`
Event called when a custom event is added and triggered from inside the map, following the rules to call the proper communication sequence. Below is a sample that demonstrates this.

To better demonstrate how the custom events are working, a couple of examples.

Sample 1) Initializing the default map with custom maptype control (Google Maps):

```
procedure TForm1.TMSFNCMaps1CustomizeMap(Sender: TObject;
  var ACustomizeMap: string);
var
  opt: string;
begin
  if TMSFNCMaps1.Service = msGoogleMaps then
  begin
    opt := '{' + LB +
```

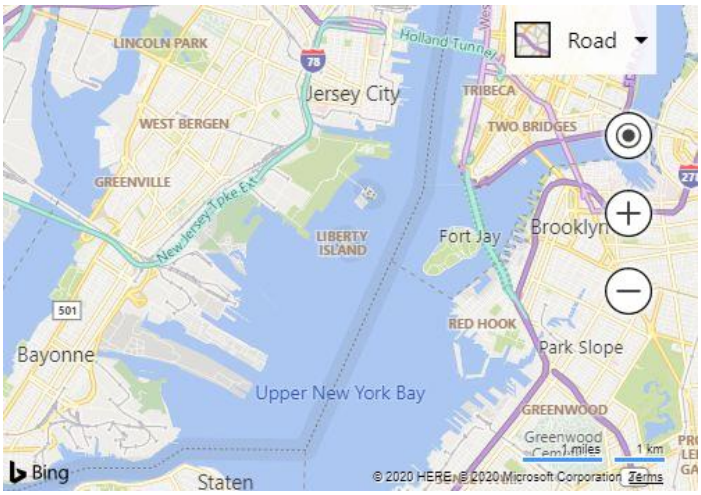
```
            '  style: window.google.maps.MapTypeControlStyle.DROPDOWN_MENU,'
+ LB +
            '  mapTypeIds: [''roadmap'', ''terrain'']' + LB +
            '}';
    ACustomizeMap := MAPVAR + '.set(''mapTypeControlOptions'', ' + opt +
')';
  end;
end;
```

Sample 2) Restricting map bounds to a specific area (Google Maps)

```
procedure TForm1.TMSFNCMaps1CustomizeMap(Sender: TObject;
  var ACustomizeMap: string);
var
  opt: string;
begin
  if TMSFNCMaps1.Service = msGoogleMaps then
  begin
    ACustomizeMap :=
      'var NEW_ZEALAND_BOUNDS = {' + LB +
      '    north: -34.36,' + LB +
      '    south: -47.35,' + LB +
      '    west: 166.28,' + LB +
      '    east: -175.81,' + LB +
      '  };' + LB + LB;

    opt := '{' + LB +
           '  restriction:{' + LB +
           '    latLngBounds: NEW_ZEALAND_BOUNDS,' + LB +
           '    strictBounds: false' + LB +
           '  }' + LB +
           '}';

    ACustomizeMap := ACustomizeMap + 'var AUCKLAND = {lat: -37.06, lng:
174.58};' + LB;
    ACustomizeMap := ACustomizeMap + MAPVAR + '.setCenter(AUCKLAND);' + LB;
    ACustomizeMap := ACustomizeMap + MAPVAR + '.setZoom(7);' + LB;
    ACustomizeMap := ACustomizeMap + MAPVAR + '.setOptions('+ opt +');';
  end;
end;
```

Sample 3) adding an extra event to Bing Maps

```
procedure TForm1.TMSFNCMaps1CustomEvent(Sender: TObject;
  AEventData: TTMSFNCMapsEventData);
begin
  if AEventData.EventName = 'MapRightClick' then
  begin
    TTMSFNCUtils.Log('map right click detected at ' +
AEventData.Coordinate.JSON);
  end;
end;

procedure TForm1.TMSFNCMaps1CustomizeMap(Sender: TObject;
  var ACustomizeMap: string);
begin
  if TMSFNCMaps1.Service = msBingMaps then
  begin
    ACustomizeMap :=
    '  window.Microsoft.Maps.Events.addHandler(' + MAPVAR + ',
''rightclick'', function(event){' + LB +
    '    ' + GETSENDEVENT + '(parseEvent(event, "MapRightClick"));' + LB +
    '  })';
  end;
end;
```




```
Debug Output: map right click detected at
{"$type":"TTMSFNCMapsCoordinate","Latitude":40.7012209558957,"Longitude":-
74.0589215556641} Process Project17.exe (10472)
```

Events

`OnZoomChanged;`
Event called when zoom has changed. An OnZoomChanged event can occur when zooming with the mouse, keyboard or with the map controls.

`OnMapTypeChanged;`
Event called when the map type has changed. An OnMapTypeChanged event can occur when the map type is changed via the map type control.

`OnMapClick;`
Event called when the map is clicked.

`OnMapDblClick;`
Event called when the map is double clicked.

`OnMapMouseUp;`
Event called when the mouse is up on the map. (Occurs after an OnMapMouseDown event).

`OnMapMouseDown;`
Event called when the mouse is down on the map.

`OnMapMouseMove;`
Event called when the mouse is moved inside the map.

`OnMapMouseEnter;`
Event called when the mouse enters the map.

`OnMapMouseLeave;`
Event called when the mouse leaves the map.

`OnMapMoveStart;`
Event called when a map move actions is started.

`OnMapMoveEnd;`
Event called when a map move actions has ended.

`OnMarkerClick;`
Event called when a marker is clicked.

`OnMarkerDblClick;`

Event called when a marker is double clicked.

`OnMarkerMouseUp;`
Event called when the mouse is up on a marker. (Occurs after an OnMarkerMouseUp event).

`OnMarkerMouseDown;`
Event called when the mouse is down on a marker.

`OnMarkerMouseEnter;`
Event called when the mouse enters a marker.

`OnMarkerMouseLeave;`
Event called when the mouse leaves a marker.

`OnPolyElementClick;`
Event called when a polyelement (polygon, polyline, circle, rectangle) is clicked.

`OnPolyElementDblClick;`

Event called when a polyelement is double clicked.

`OnPolyElementMouseUp;`
Event called when the mouse is up on a polyelement. (Occurs after an OnPolyElementMouseUp event).

`OnPolyElementMouseDown;`
Event called when the mouse is down on a polyelement.

`OnPolyElementMouseEnter;`
Event called when the mouse enters a polyelement.

`OnPolyElementMouseLeave;`
Event called when the mouse leaves a polyelement.

`OnMapInitialized;`
Event called when the map is initialized.

Note: Some events (like OnMarkerMouseUp, OnMarkerMouseDown, OnMarkerMouseEnter, OnMarkerMouseLeave …) are not supported when using the Apple MapKit mapping service because this service is mainly targeted at mobile use.

Common Types

During the development of your application with TTMSFNCMaps you'll encounter the following record types:

TTMSFNCMapsCoordinateRec
TTMSFNCMapsBoundsRec

They are types that allow easy conversion from the class type equivalents to pass through or retrieve data from the map. They are defined inside the *.TMSFNCCommonTypes.pas unit and they also come with a set of helper methods:

```
function CenterCoordinate: TTMSFNCMapsCoordinateRec;
```
method that returns a coordinate with latitude and longitude of 0.

```
function DefaultCoordinate: TTMSFNCMapsCoordinateRec;
```
method that returns a coordinate based on the default latitude and longitude (New York)

```
function EmptyCoordinate: TTMSFNCMapsCoordinateRec;
```
Method that returns an empty coordinate (which is the same as a center coordinate)

```
function CreateCoordinate(const ALatitude: Double; const ALongitude:
Double): TTMSFNCMapsCoordinateRec;
```
Method that creates a coordinate record based on a latitude and longitude value.

```
function CreateBounds(ACoordinates: TTMSFNCMapsCoordinateRecArray):
TTMSFNCMapsBoundsRec; overload;
```
Method that creates a bounds record based on an array of coordinates

```
function CreateBounds(ACoordinatesArray:
TTMSFNCMapsCoordinateRecArrayArray): TTMSFNCMapsBoundsRec; overload;
```
Method that creates a bounds record based on an array of an array of coordinates (multi-array). This is helpful when wanting to combine arrays of coordinates from different collections such as combining markers & polylines.

```
function CreateBounds(ACoordinatesArrayArray:
TTMSFNCMapsCoordinateRecArrayArrayArray): TTMSFNCMapsBoundsRec; overload;
```
Method that creates a bounds record based on an array of an array of an array of coordinates (multi-array of array). This is helpful when wanting to combine arrays of coordinates from different collections such as combining markers & polylines.

```
function CreateBounds(ANorthLatitude, AEastLongitude, ASouthLatitude,
AWestLongitude: Double): TTMSFNCMapsBoundsRec; overload;
```
Method that creates a bounds record based on an North latitude and longitude and a South latitude and longitude

```
function CreateCircle(ACenter: TTMSFNCMapsCoordinateRec; ARadius: Double):
TTMSFNCMapsCoordinateRecArray;
```
Method that creates a circle based on a center and a radius. It returns an array of coordinates that form a circle.

```
function BoundsCenter(ABounds: TTMSFNCMapsBoundsRec):
TTMSFNCMapsCoordinateRec;
```
Method that calculates the center of a bounds record, which is determined by a NorthEast coordinate and a SouthWest coordinate.

```
function BoundsRectangle(ABounds:  TTMSFNCMapsBoundsRec):
TTMSFNCMapsCoordinateRecArray;
```
Method that returns a rectangle based on a bounds record. It returns an array of coordinates that form a rectangle.

```
function MeasureDistance(ACoordinate1: TTMSFNCMapsCoordinateRec;
ACoordinate2: TTMSFNCMapsCoordinateRec): Double;
```
Measures the distance between 2 coordinates.

```
function CalculateCoordinate(ACoordinate: TTMSFNCMapsCoordinateRec;
ABearing: Double; ADistance: Double): TTMSFNCMapsCoordinateRec;
```
Calculates a new coordinate based on an existing coordinate, the bearing and the distance. (Please see chapter PolyElements for an example on using the bearing).

```
function CalculateBearing(ACoordinate1: TTMSFNCMapsCoordinateRec;
ACoordinate2: TTMSFNCMapsCoordinateRec): Double;
```
Method that calculates the bearing between 2 coordinates.

TTMSFNCGoogleMaps

TTMSFNCGoogleMaps is a descendant of TTMSFNCMaps and exposes a set of properties and methods to add functionality specific for the Google Maps API. Each topic below goes deeper into the already existing functionality in TTMSFNCMaps and adds more functionality on top.

Dragging Markers

After adding markers with the AddMarker function or directly through the Markers collection the basic properties that can be set in the abstract TTMSFNCMaps class are the Visible, Title and Coordinate properties. On top of that, the Google Maps API has the ability to add a drop animation, make the marker draggable, clickable. For testing purposes we have added code that demonstrates how to make a marker draggable.

```
procedure TForm1.AddDraggableMarker;
var
  m: TTMSFNCGoogleMapsMarker;
begin
  m :=
TTMSFNCGoogleMapsMarker(TMSFNCGoogleMaps1.AddMarker(DefaultCoordinate));
  m.Draggable := True;
end;
```

When holding the mouse/finger down on the marker, a cross appears beneath and you are able to freely drag around the marker. Releasing the mouse/finger will drop the marker to the new location and trigger the OnMarkerDragEnd event.

In the OnMarkerDragEnd event you have 2 choices. Either leave the original coordinate store inside the marker collection item, or synchronize. The synchronization will help in updating markers that are already added to the map. When setting the coordinate of the marker to the new location, each following update will not reset the marker to the original location, but to the new location. Leaving out the code will reset the marker to the original location when another property changes.

```
procedure TForm1.TMSFNCGoogleMaps1MarkerDragEnd(Sender: TObject;
  AEventData: TTMSFNCMapsEventData);
begin
  if Assigned(AEventData.Marker) then
    AEventData.Marker.Coordinate := AEventData.Coordinate;
end;
```

Editing Polyelements

Polyelements such as a polygon, polyline, rectangle and circle can be edited. To allow editing of a polyelement, add the polyelement to the map as demonstrated in the previous chapter of adding polyelements. Make sure to also specify the Editable property.

```
procedure TForm1.AddRectangleToMap;
var
  c: TTMSFNCGoogleMapsRectangle;
  b: TTMSFNCMapsBoundsRec;
  ne, sw: TTMSFNCMapsCoordinateRec;
begin
  TMSFNCGoogleMaps1.BeginUpdate;
  ne := CalculateCoordinate(DefaultCoordinate, 45, 100000);
  sw := CalculateCoordinate(DefaultCoordinate, 225, 100000);
  b := CreateBounds(ne.Latitude, ne.Longitude, sw.Latitude, sw.Longitude);
  c := TTMSFNCGoogleMapsRectangle(TMSFNCGoogleMaps1.AddRectangle(b));
  c.Editable := True;
  c.FillColor := gcOrange;
  c.FillOpacity := 0.5;
  c.StrokeColor := gcGreen;
  c.StrokeWidth := 4;
  TMSFNCGoogleMaps1.EndUpdate;
end;
```

When adding the rectangle to the map, you'll notice that handles appear. These handles can be used to change the rectangle. Whenever a change is detected the OnPolyElementEditEnd is called. As this polyelement is a rectangle, the data is stored in the CustomData property of the TTMSFNCMapsEventData object as it contains more than only a coordinate. The CustomData contains an array of coordinates that can be mapped to the bounds property of the rectangle as demonstrated in the code below. The CustomData is always JSON.

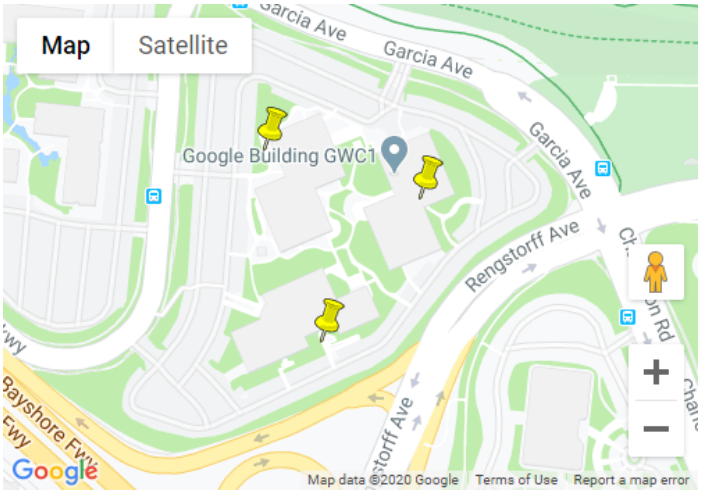
```
procedure TForm1.TMSFNCGoogleMaps1PolyElementEditEnd(Sender: TObject;
  AEventData: TTMSFNCMapsEventData);
begin
  if Assigned(AEventData.PolyElement) then
  begin
    if (AEventData.PolyElement is TTMSFNCGoogleMapsRectangle) then
      (AEventData.PolyElement as TTMSFNCGoogleMapsRectangle).Bounds.JSON :=
AEventData.CustomData;
  end;
end;
```

Loading KML Layers

Loading of KML layers is supported out of the box in Google Maps. Simply specify the URL of the KML layer and Google Maps will do the rest. Below is a sample and the output of adding a KML layer.

```
TMSFNCGoogleMaps1.AddKMLLayer('https://developers.google.com/maps/documenta
tion/javascript/examples/kml/westcampus.kml');
```

An optional parameter (true by default) is to automatically zoom to the bounds of the loaded KML layer. When executing the above code this is the result:

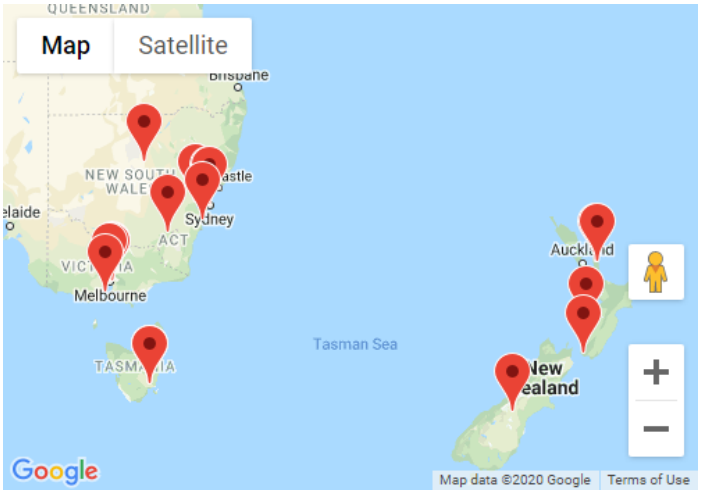


Clusters

Clustering is a nice feature of Google Maps that comes with a separate library (MarkerClustererPlus). The library can be used to cluster markers together. To get started adding clusters, first add a set of markers.

```
procedure TForm1.AddClusters;
var
 locations: array of TTMSFNCMapsCoordinateRec;
  I: Integer;
begin
  locations := [
    CreateCoordinate(-31.563910, 147.154312),
    CreateCoordinate(-33.718234, 150.363181),
    CreateCoordinate(-33.727111, 150.371124),
    CreateCoordinate(-33.848588, 151.209834),
    CreateCoordinate(-33.851702, 151.216968),
    CreateCoordinate(-34.671264, 150.863657),
```

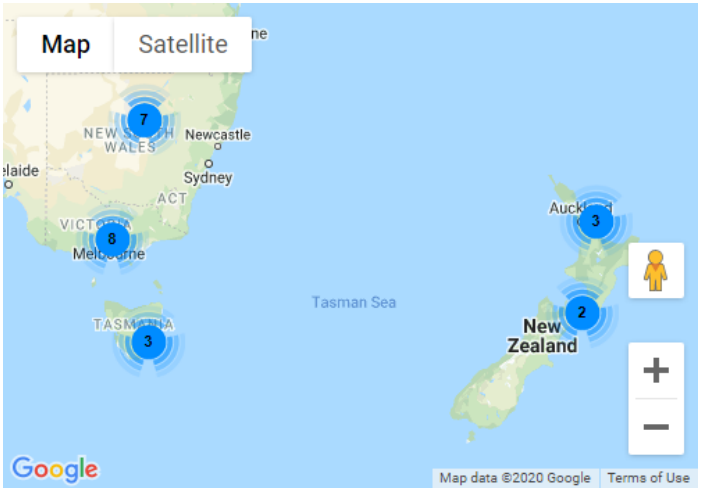
```
    CreateCoordinate(-35.304724, 148.662905),
    CreateCoordinate(-36.817685, 175.699196),
    CreateCoordinate(-36.828611, 175.790222),
    CreateCoordinate(-37.750000, 145.116667),
    CreateCoordinate(-37.759859, 145.128708),
    CreateCoordinate(-37.765015, 145.133858),
    CreateCoordinate(-37.770104, 145.143299),
    CreateCoordinate(-37.773700, 145.145187),
    CreateCoordinate(-37.774785, 145.137978),
    CreateCoordinate(-37.819616, 144.968119),
    CreateCoordinate(-38.330766, 144.695692),
    CreateCoordinate(-39.927193, 175.053218),
    CreateCoordinate(-41.330162, 174.865694),
    CreateCoordinate(-42.734358, 147.439506),
    CreateCoordinate(-42.734358, 147.501315),
    CreateCoordinate(-42.735258, 147.438000),
    CreateCoordinate(-43.999792, 170.463352)];


  TMSFNCGoogleMaps1.BeginUpdate;
  for I := 0 to Length(locations) - 1 do
  begin
    TMSFNCGoogleMaps1.AddMarker(locations[I]);
  end;
  TMSFNCGoogleMaps1.EndUpdate;

end;
```



The second step is to add a cluster and bind the markers to a cluster. To add a cluster, use the Clusters.Add method. Additionally, we also use a ZoomToBounds to visualize the clusters on the map.

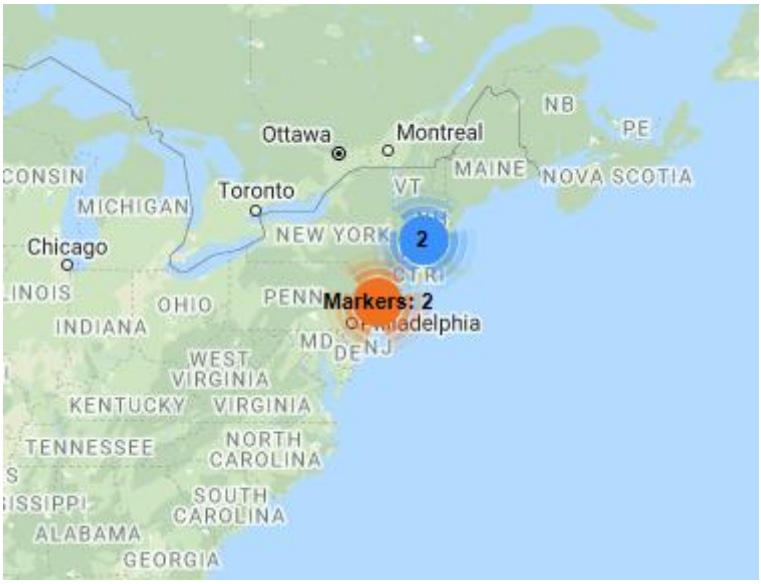
```
procedure TForm1.AddClusters;
var
  locations: array of TTMSFNCMapsCoordinateRec;
  I: Integer;
  m: TTMSFNCGoogleMapsMarker;
begin
  locations := [
    CreateCoordinate(-31.563910, 147.154312),
```

```
    CreateCoordinate(-33.718234, 150.363181),
    CreateCoordinate(-33.727111, 150.371124),
    CreateCoordinate(-33.848588, 151.209834),
    CreateCoordinate(-33.851702, 151.216968),
    CreateCoordinate(-34.671264, 150.863657),
    CreateCoordinate(-35.304724, 148.662905),
    CreateCoordinate(-36.817685, 175.699196),
    CreateCoordinate(-36.828611, 175.790222),
    CreateCoordinate(-37.750000, 145.116667),
    CreateCoordinate(-37.759859, 145.128708),
    CreateCoordinate(-37.765015, 145.133858),
    CreateCoordinate(-37.770104, 145.143299),
    CreateCoordinate(-37.773700, 145.145187),
    CreateCoordinate(-37.774785, 145.137978),
    CreateCoordinate(-37.819616, 144.968119),
    CreateCoordinate(-38.330766, 144.695692),
    CreateCoordinate(-39.927193, 175.053218),
    CreateCoordinate(-41.330162, 174.865694),
    CreateCoordinate(-42.734358, 147.439506),
    CreateCoordinate(-42.734358, 147.501315),
    CreateCoordinate(-42.735258, 147.438000),
    CreateCoordinate(-43.999792, 170.463352)];


  TMSFNCGoogleMaps1.BeginUpdate;
  TMSFNCGoogleMaps1.Clusters.Add;

  for I := 0 to Length(locations) - 1 do
  begin
    m :=
TTMSFNCGoogleMapsMarker(TMSFNCGoogleMaps1.AddMarker(locations[I]));
    m.Cluster := TMSFNCGoogleMaps1.Clusters[0];
  end;


TMSFNCGoogleMaps1.ZoomToBounds(TMSFNCGoogleMaps1.Markers.ToCoordinateArray)
;

  TMSFNCGoogleMaps1.EndUpdate;
end;
```



To optionally use a custom text instead of the number of markers on a cluster icon, the Text property can be used. Use %d to insert the marker count and the   HTML character to avoid word wrapping.

Example:
```
  m.Clusters[0].Text := 'Markers: %d';
```



To optionally use custom images for clusters, the ImagePath property can be used.
Make sure the provided path is pointing to a folder that contains 5 png image files with the following names: 1.png, 2.png, 3.png, 4.png and 5.png

Example:

```
TMSFNCMaps1.Clusters[0].ImagePath := 'file://' + StringReplace(ExtractFilePath(ParamStr(0)) + '/ClusterImages/', '\', '/', [rfReplaceAll]);;
```

The screensho shows 2 clusters that both contain 2 markers. One is displayed with the default image (blue) and the other one is displayed with a custom image (orange).

Events

OnClusterClick;
Event called when a cluster is clicked.

`OnClusterMouseEnter;`
Event called when the mouse enters a cluster.

`OnClusterMouseLeave;`
Event called when the mouse leaves a cluster.

OverlayViews

An overlayview is a layer that is placed on top of the map that contains HTML code. The overlayview can be displayed at a specific location or within specific bounds to make it adapt to the map's zoom level.

Additionally, when connected to a marker, the overlayview can function as a marker label.
To connect an overlay to a marker, use the markers' AddOverlayView and/or OverlayView property.

Properties

Bounds
Set the bounds NorthEast and SouthWest coordinates. Only used when Mode is set to omBounds.

BorderColor
Set the border color of the overlayview.

BackgroundColor
Set the backgroundcolor of the overlayview.

Clickable
Set if the OnOverlayViewClick event is triggered when the overlayview is clicked.

Coordinate
Set the location at which the overlayview is displayed. Only used when Mode is set to omCoordinate and no marker is connected.

CoordinatePosition
Set the position of the overlayview relative to the Coordinate. Available options are: TopLeft, TopCenter, TopRight, BottomLeft, BottomCenter, BottomRight, CenterLeft, CenterCenter, CenterRight, Custom. Only used when Mode is set to omCoordinate.

CoordinateOffsetTop
Set the top offset relative to the overlayview's position. Only used when Mode is set to omCoordinate.

CoordinateOffsetLeft
Set the left offset relative to the overlayview's position. Only used when Mode is set to omCoordinate.

Font
Set the font color, name, size and style.

Height
Set the height of the overlayview. If set to -1 the height will autosize to the content.

Mode
Sets if the overlayview is displayed within the specified bounds (Bounds property) or at a specific coordinate (Coordinate property). When a marker is connected this must be set to omCoordinate.

Padding
Sets the padding between the overlayview's border and content.

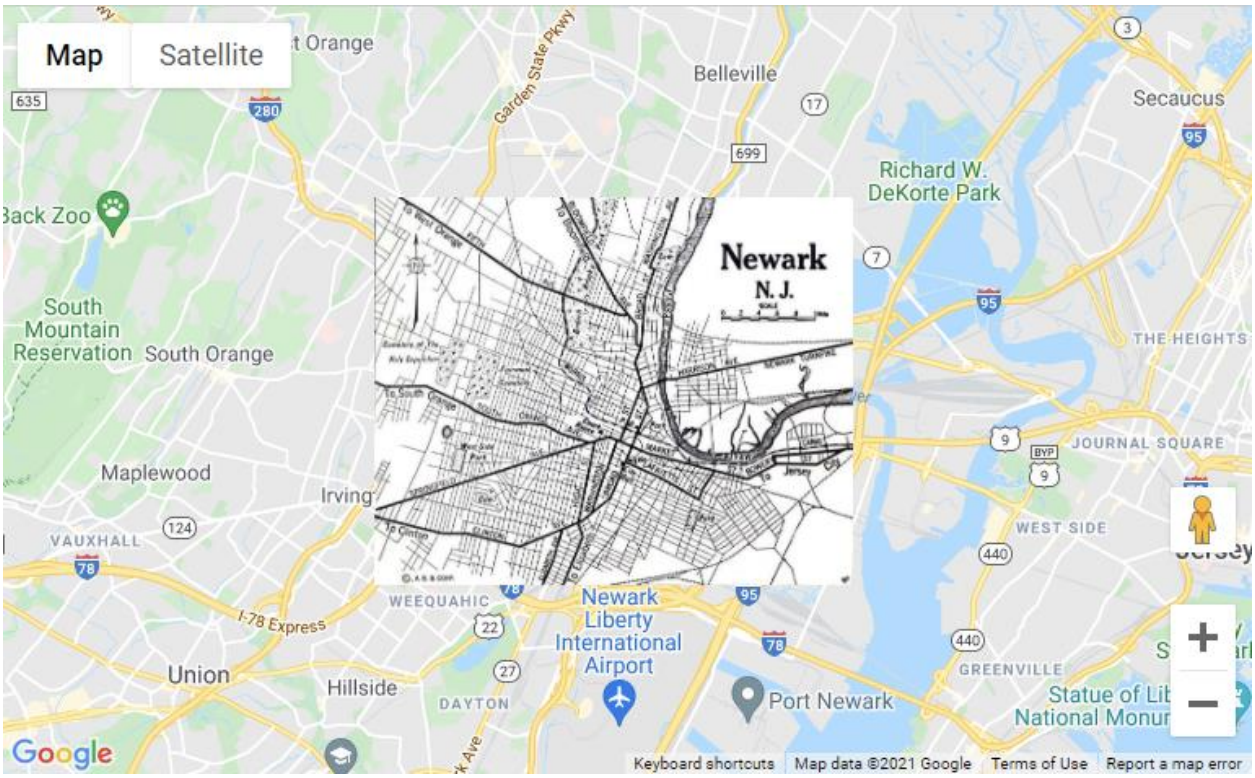
Visible
Show or hide the overlayview.

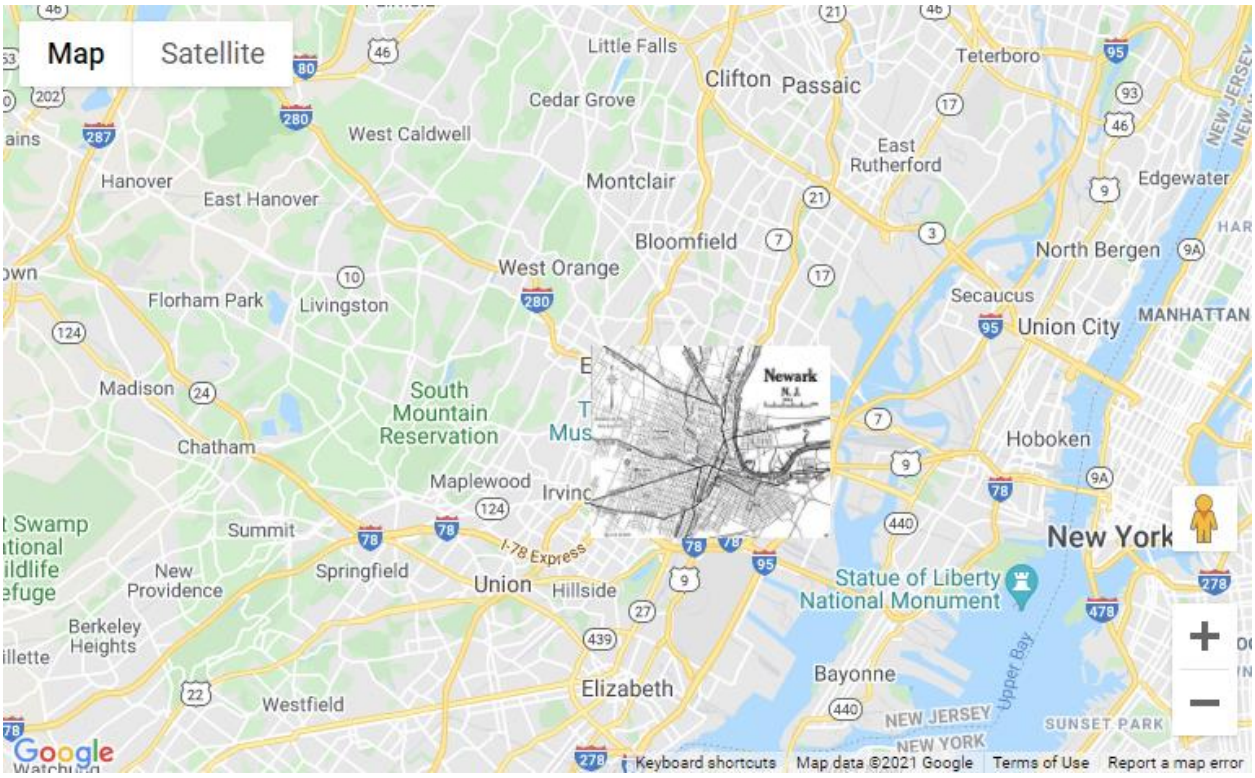
Width
Set the width of the overlayview. If set to -1 the width will autosize to the content.

Events

OnOverlayViewClick;
Event called when an overlayview is clicked.

- Sample using OverlayView Bounds. The image remains within the specified bounds regardless of map zoom level.






```
var
  ov: TTMSFNCGoogleMapsOverlayView;
begin
  TMSFNCGoogleMaps1.BeginUpdate;
```

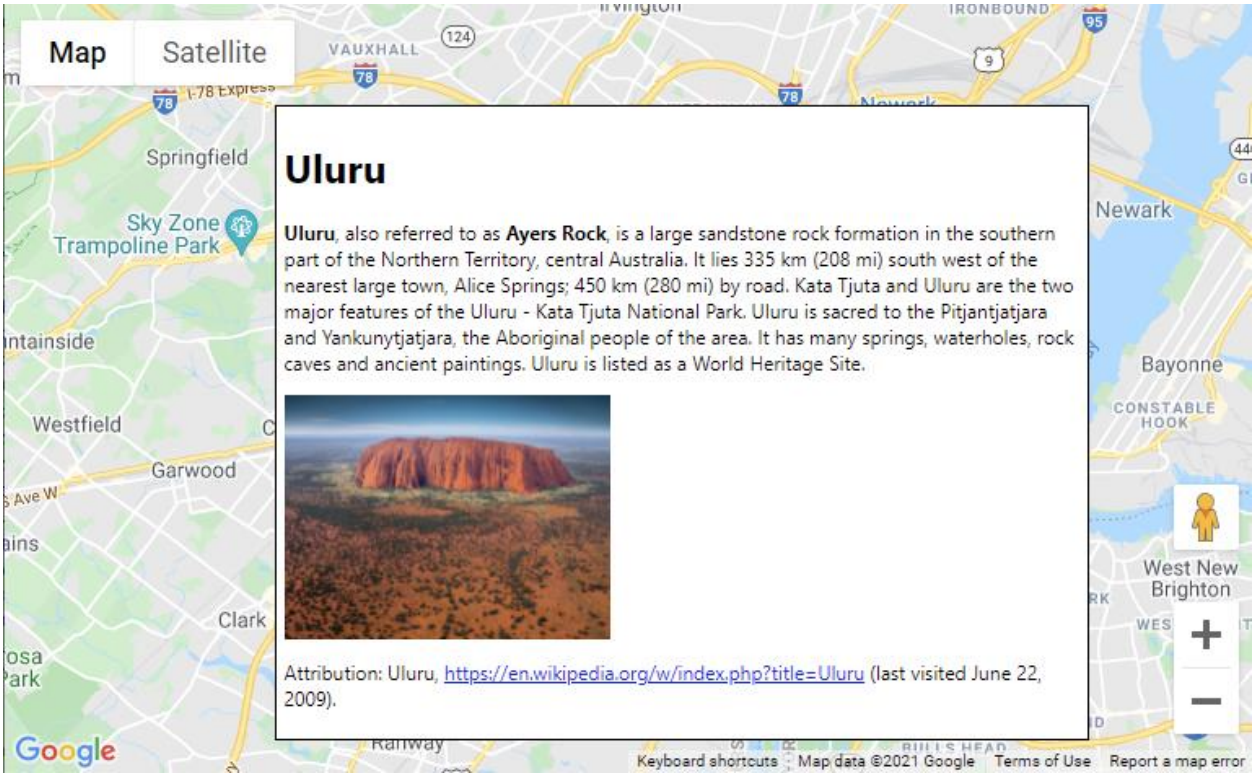
```
  ov := TMSFNCGoogleMaps1.AddOverlayView;
  ov.Text := '<img src="https://storage.googleapis.com/geo-devrel-public-
buckets/newark_nj_1922-661x516.jpeg"
style="position:absolute;width:100%;height:100%;">';
  ov.Bounds.NorthEast.Latitude := 40.773941;
  ov.Bounds.NorthEast.Longitude := -74.12544;
  ov.Bounds.SouthWest.Latitude := 40.712216;
  ov.Bounds.SouthWest.Longitude := -74.22655;
  ov.Mode := omBounds;
  ov.Padding := 0;
  ov.BackgroundColor := gcNull;
  ov.BorderColor := gcNull;

  TMSFNCGoogleMaps1.EndUpdate;
```

- Sample using an OverlayView connected to a Marker


```
var
  m: TTMSFNCGoogleMapsMarker;
  ov: TTMSFNCGoogleMapsOverlayView;
begin
  TMSFNCGoogleMaps1.BeginUpdate;

  ov := TMSFNCGoogleMaps1.AddOverlayView;
  ov.Text := 'Marker Label';

  m :=
TTMSFNCGoogleMapsMarker(TMSFNCGoogleMaps1.AddMarker(TMSFNCGoogleMaps1.Optio
ns.DefaultLatitude, TMSFNCGoogleMaps1.Options.DefaultLongitude));
```

```
  m.AddOverlayView('Marker Label');

  TMSFNCGoogleMaps1.EndUpdate;
```

- Sample using an OverlayView to display complex HTML at a specific location




```
var
  ov: TTMSFNCGoogleMapsOverlayView;
begin
  TMSFNCGoogleMaps1.BeginUpdate;

  ov := TMSFNCGoogleMaps1.AddOverlayView;
  ov.Mode := omCoordinate;
  ov.Coordinate.Latitude := 40.71;
  ov.Coordinate.Longitude := -74.22;
  ov.Width := 500;

  ov.Text := '' +
      '<h1 id="firstHeading" class="firstHeading">Uluru</h1>'+
      '<div id="bodyContent">'+
      '<p><b>Uluru</b>, also referred to as <b>Ayers Rock</b>, is a large '
+
      'sandstone rock formation in the southern part of the '+
      'Northern Territory, central Australia. It lies 335 km
(208 mi) '+
      'south west of the nearest large town, Alice Springs; 450 km '+
      '(280 mi) by road. Kata Tjuta and Uluru are the two major '+
      'features of the Uluru - Kata Tjuta National Park. Uluru is '+
      'sacred to the Pitjantjatjara and Yankunytjatjara, the '+
      'Aboriginal people of the area. It has many springs, waterholes, '+
```

```
      'rock caves and ancient paintings. Uluru is listed as a World '+
      'Heritage Site.</p>'+
      '<img src="https://www.tripsavvy.com/thmb/XHi4Ec428VJFk_bsvNtv6fx5-
bE=/2000x1500/filters:fill(auto,1)/GettyImages-675746097-
5a8b94a427364afc89f054f71ceda86f.jpg" width="200"></img>' +
      '<p>Attribution: Uluru, <a target="_blank"
href="https://en.wikipedia.org/w/index.php?title=Uluru&oldid=297882194">'+
      'https://en.wikipedia.org/w/index.php?title=Uluru</a> '+
      '(last visited June 22, 2009).</p>'+
      '</div>';

  TMSFNCGoogleMaps1.EndUpdate;
```

StreetView

Programmatically show or hide StreetView is possible in TTMSFNCGoogleMaps through the Options.StreetView.Enabled property. Extra properties Heading, Pitch and Zoom are also available.

Events

`OnStreetViewChange`
Event called each time the StreetView position changes.

The data is stored in the CustomData property of the TTMSFNCMapsEventData object as it contains more than only a coordinate. The CustomData contains the updated values for the Heading, Pitch and Zoom properties. The CustomData is always JSON.

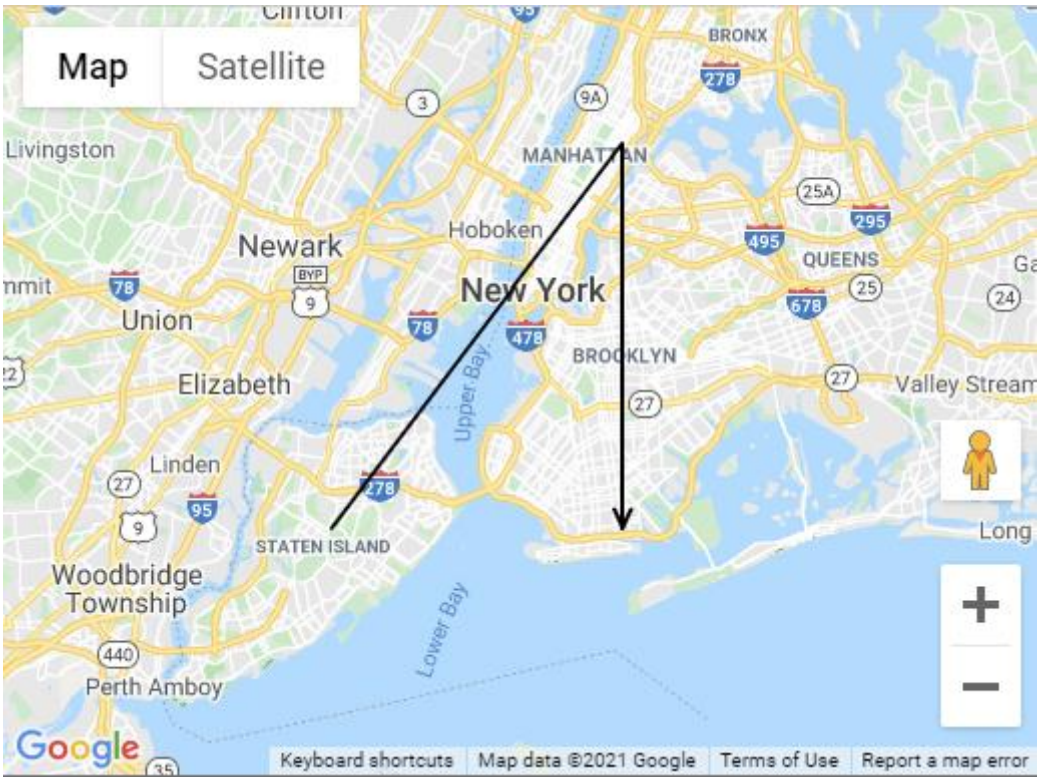
```
procedure TForm1.mStreetViewChange(Sender: TObject;
  AEventData: TTMSFNCMapsEventData);
begin
    TTMSFNCUtils.Log('StreetView: ' + AEventData.CustomData);
end;
```

Geodesic Polylines

The Polyline's Geodesic boolean property can be used to display a polyline as a geodesic line.

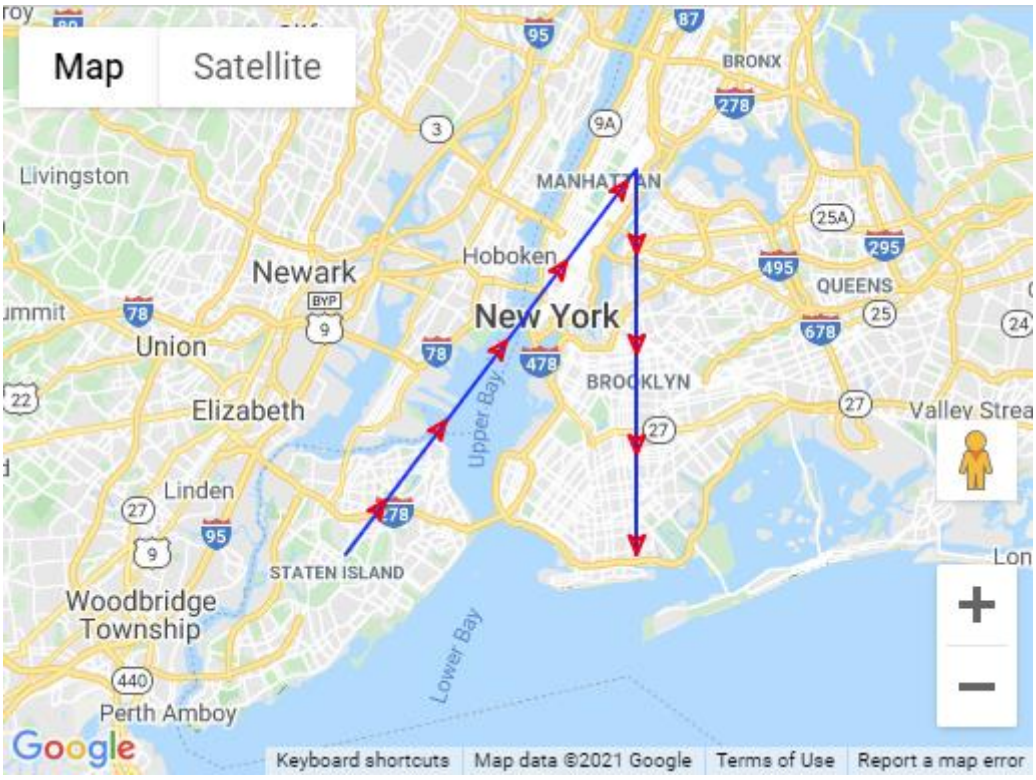
Polyline Symbols

The Polyline's Symbols property can be used to display symbols on a polyline.
By default a single arrow symbol is displayed at the end of the polyline with the same color, opacity and strokewidth as the polyline. Additional properties are available to configure the symbols.



This example shows a red closed arrow repeated every 50 pixels on a blue polyline.

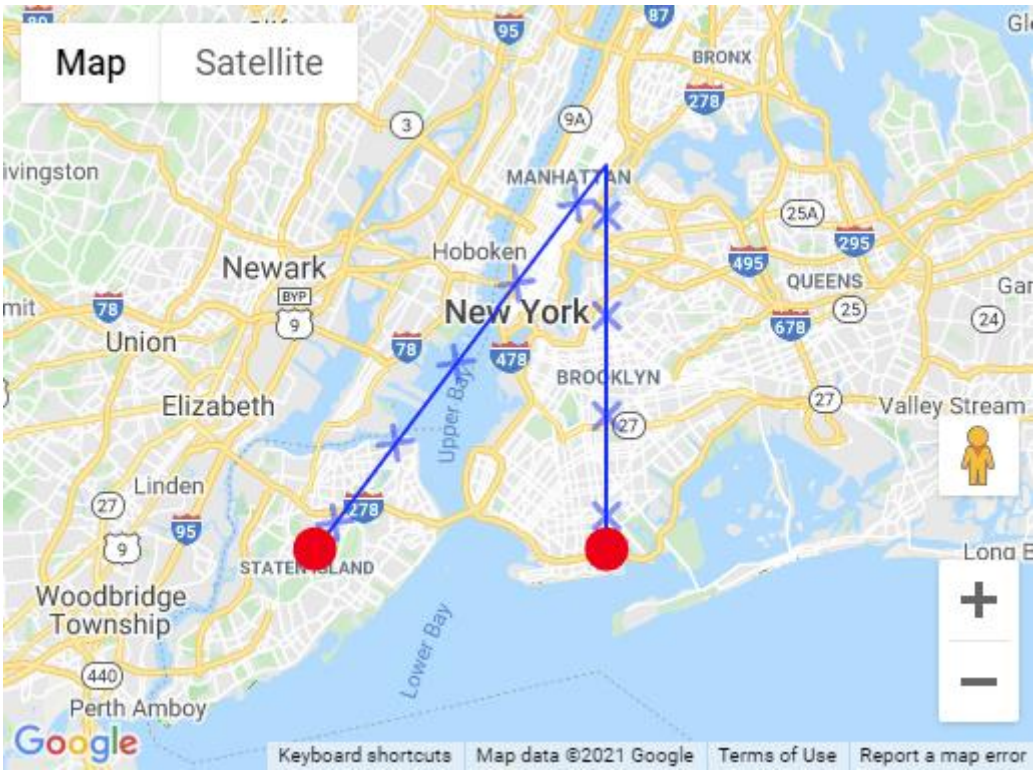
```
var
  ar: TTMSFNCMapsCoordinateRecArray;
  p: TTMSFNCGoogleMapsPolyline;
  ps: TTMSFNCGoogleMapsPolylineSymbol;
begin
  TMSFNCGoogleMaps1.BeginUpdate;
  SetLength(ar, 3);
  ar[0].Latitude := TMSFNCGoogleMaps1.Options.DefaultLatitude - 0.1;
  ar[0].Longitude := TMSFNCGoogleMaps1.Options.DefaultLongitude - 0.1;
  ar[1].Latitude := TMSFNCGoogleMaps1.Options.DefaultLatitude + 0.1;
  ar[1].Longitude := TMSFNCGoogleMaps1.Options.DefaultLongitude + 0.1;
  ar[2].Latitude := TMSFNCGoogleMaps1.Options.DefaultLatitude - 0.1;
  ar[2].Longitude := TMSFNCGoogleMaps1.Options.DefaultLongitude + 0.1;
  p := TMSFNCGoogleMaps1.AddPolyline(ar);
  p.StrokeColor := gcBlue;

  ps := p.Symbols.Add;
  ps.Path := spForwardClosedArrow;
  ps.StrokeColor := gcRed;
  ps.RepeatSymbol := 50;

  TMSFNCGoogleMaps1.EndUpdate;
end;
```



This example shows how to use multiple symbols simultaneously including a symbol defined with a custom SVG path.

```
var
  ar: TTMSFNCMapsCoordinateRecArray;
  p: TTMSFNCGoogleMapsPolyline;
  ps: TTMSFNCGoogleMapsPolylineSymbol;
begin
  TMSFNCGoogleMaps1.BeginUpdate;
  SetLength(ar, 3);
  ar[0].Latitude := TMSFNCGoogleMaps1.Options.DefaultLatitude - 0.1;
  ar[0].Longitude := TMSFNCGoogleMaps1.Options.DefaultLongitude - 0.1;
  ar[1].Latitude := TMSFNCGoogleMaps1.Options.DefaultLatitude + 0.1;
  ar[1].Longitude := TMSFNCGoogleMaps1.Options.DefaultLongitude + 0.1;
  ar[2].Latitude := TMSFNCGoogleMaps1.Options.DefaultLatitude - 0.1;
  ar[2].Longitude := TMSFNCGoogleMaps1.Options.DefaultLongitude + 0.1;
  p := TMSFNCGoogleMaps1.AddPolyline(ar);
  p.StrokeColor := gcBlue;

  ps := p.Symbols.Add;
  ps.Path := spCustom;
  ps.CustomPath := 'M -2,-2 2,2 M 2,-2 -2,2';
  ps.Scale := 3;
  ps.Offset := 50;
  ps.RepeatSymbol := 50;
  ps.StrokeOpacity := 0.5;

  ps := p.Symbols.Add;
  ps.Path := spCircle;
```

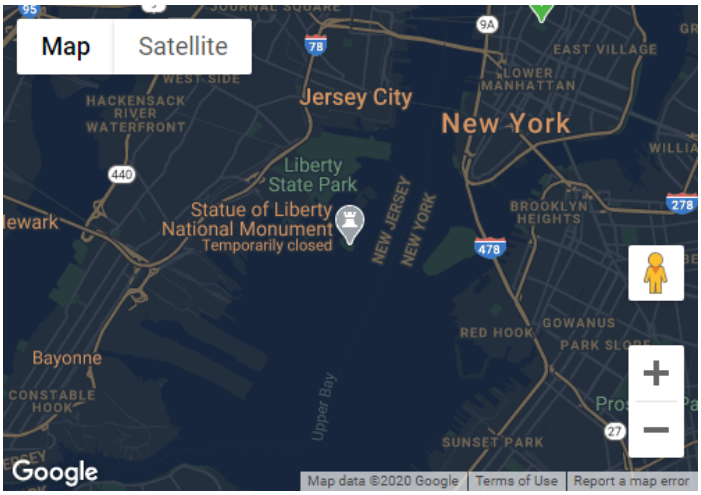
```
  ps.Offset := 0;
  ps.RepeatSymbol := 100;
  ps.RepeatSymbolUnits := unPercentage;
  ps.Scale := 10;
  ps.StrokeWidth := 2;
  ps.StrokeColor := gcRed;
  ps.FillColor := gcRed;
  ps.FillOpacity := 1;

  TMSFNCGoogleMaps1.EndUpdate;
end;
```



Map Style

When using TTMSFNCGoogleMaps, you are able to specify a style to change the visuals of the map. By default, the Options.MapStyle is empty. To specify a style, you need to assign JSON to the Options.MapStyle property. There is a predefined night style available that can be used. To create your own style, use the Map Style Creator at https://mapstyle.withgoogle.com/

```
TMSFNCGoogleMaps1.Options.MapStyle := MAPSTYLENIGHT;
```

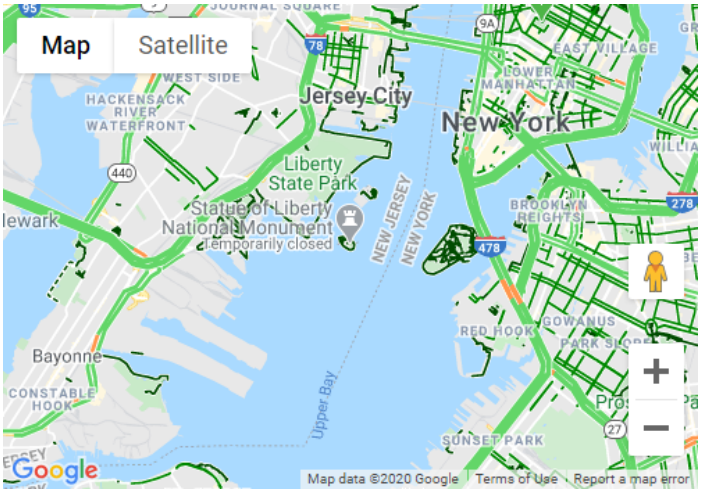


Note: Options.MapStyle is disabled when a Map ID is assigned (currently only required for Map Rotation and Tilting). The Map style should be defined in the Google Maps Console. For information on map styles: https://developers.google.com/maps/documentation/javascript/styling

Traffic & Bicycling

Enabling traffic and bicycling layers is as simple as using the code below.

```
TMSFNCGoogleMaps1.Options.ShowTraffic := True;
TMSFNCGoogleMaps1.Options.ShowBicycling := True;
```



Map Rotation & Tilting

When using TTMSFNCGoogleMaps, it is possible to specify the rotation and tilting of the map.

Note: A MapID is required when using Rotation or Tilting.

Detailed information on creating a Google Map ID:
https://developers.google.com/maps/documentation/get-map-id

Detailed information on Google Map ID functionality:
https://developers.google.com/maps/documentation/javascript/styling#creating-map-ids

Note: Options.MapStyle is disabled when a MapID is assigned. The Map style should be defined in the Google Maps Console. For information on map styles:
https://developers.google.com/maps/documentation/javascript/styling

Options.Heading
Programmatically change the rotation of the map in degrees.
By default the value is set to 0 with North facing up on the map.
For example, if MapRotation is set to 180, South will be facing up on the map.

Options.Tilt
Programmatically change the tilting of the map.

Example:

```
TMSFNCGoogleMaps1.Options.MapID := 'abc123';
TMSFNCGoogleMaps1.Options.Heading := 180;
TMSFNCGoogleMaps1.Options.Tilt := 5;
```

Integrated Directions

When using TTMSFNCGoogleMaps, it is possible to calculate step by step directions between two locations.

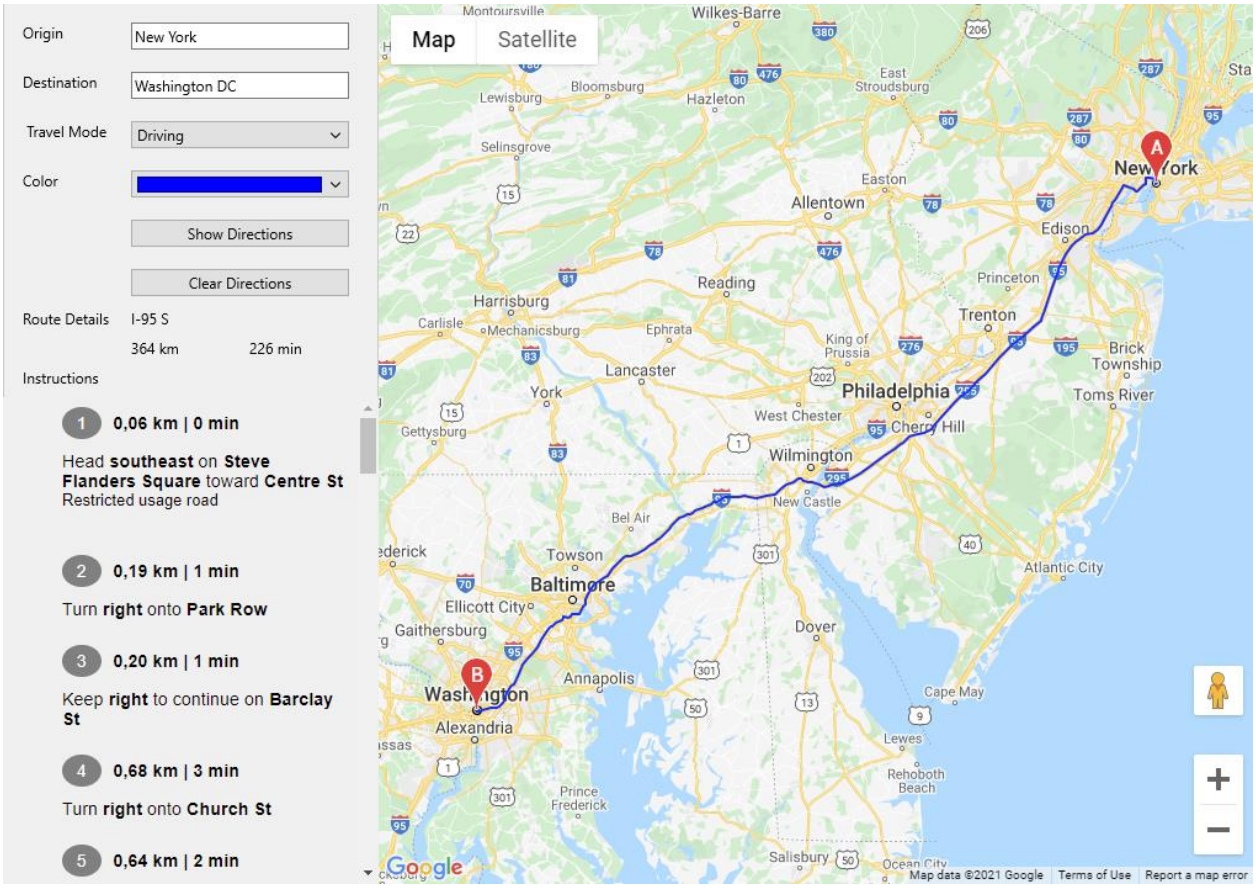
AddDirections
Calculates and displays directions between “Origin” and “Destination” location specified with a string address or a latitude and longitude coordinate.

Parameters:

AOrigin: Specifies the origin address location as a string
ADestination: Specifies the destination address location as a string
AOriginLatitude: Specifies the origin location latitude coordinate
AOriginLongitude: Specifies the origin location longitude coordinate
ADestinationLatitude: Specifies the destination location latitude coordinate
ADestinationLongitude: Specifies the destination location longitude coordinate
AShowMarkers: Indicates if markers are displayed at the origin and destination location
AShowPolyline: Indicates if a polyline indicating the route between origin and destination is displayed
AStrokeColor: Specifies the color of the polyline
AStrokeWidht: Specifies the width of the polyline
AStrokeOpacity: Specifies the opacity of the polyline
ATravelMode: Specifies which travel mode to use when calculating directions. Available options are Driving, Walking, Bicycling and PublicTransport
AvoidTolls: Indicates toll roads should be avoided when calculating directions
AWayPoints: Add waypoints to the route calculation
AOptimizeWayPoints: Indicates the waypoints order can be changed to optimize the route calculation (Azure, Google Maps, TomTom only)

Directions
Collection containing a list of previous requested directions.
Removing an item in the collection will automatically remove it from the map.

OnRetrievedDirectionsData
Event triggered when directions calculation was successful. A subset of the directions data is available in ADirectionsData. The full directions data is available in AEventData.CustomData as a JSON object.



TTMSFNCOpenLayers

TTMSFNCOpenLayers is a descendant of TTMSFNCMaps and exposes extra functionality specific for the OpenLayers API.

TileServer

When using TTMSFNCOpenLayers, you are able to specify a custom tileserver URL.

Options.TileServer
Specifies a custom tileserver URL used to display the map.

TileLayers

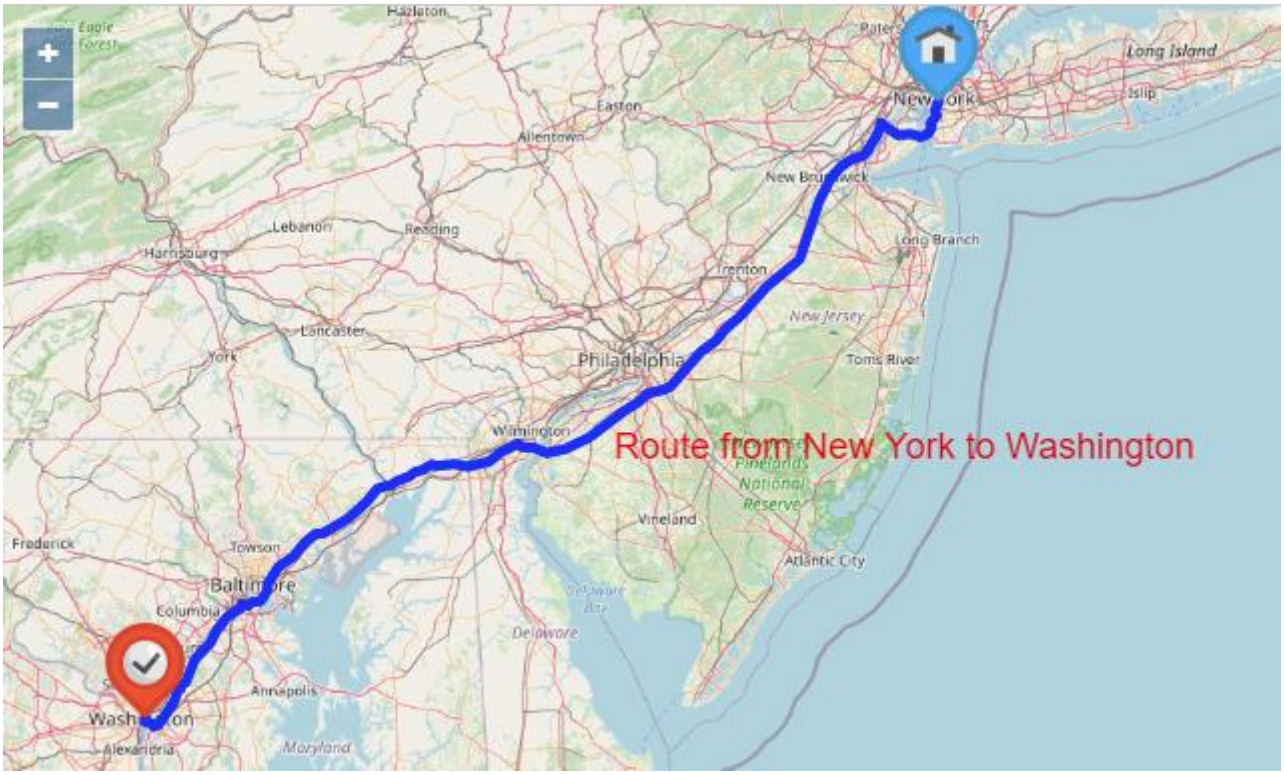
When using TTMSFNCOpenLayers, you are able to specify a list of tilelayers to display top of the map layer.

TileLayers
A collection of tilelayers with an URL value and an optional Opacity value.

Polyline & Polygon labels

When using TTMSFNCOpenLayers, you are able to specify a label text for polylines and polygons. The label text is displayed in the center of the polyline by default.

Label.Text
Sets the text of the label
Label.FontColor
Sets the font color of the label text
Label.FontSize
Sets the font size of the label text
Label.OffsetX
Sets the label's X offset in pixels
Label.OffsetY
Sets the label's Y offset in pixels



TTMSFNCMapKit

TTMSFNCMapKit is a descendant of TTMSFNCMaps and exposes extra functionality specific for the MapKit JS API.

Map Rotation

When using TTMSFNCMapKit, it is possible to specify the rotation of the map.

Options.MapRotation
Programmatically set the rotation of the map in degrees.
By default the value is set to 0 with North facing up on the map.
If, for example, MapRotation is set to 180, South will be facing up on the map.

OnMapRotationChanged
Event triggered when the map's rotation has changed. The new map rotation value is available in AEventData.CustomData.

Integrated Directions

When using TTMSFNCMapKit, it is possible to calculate step by step directions between two locations.

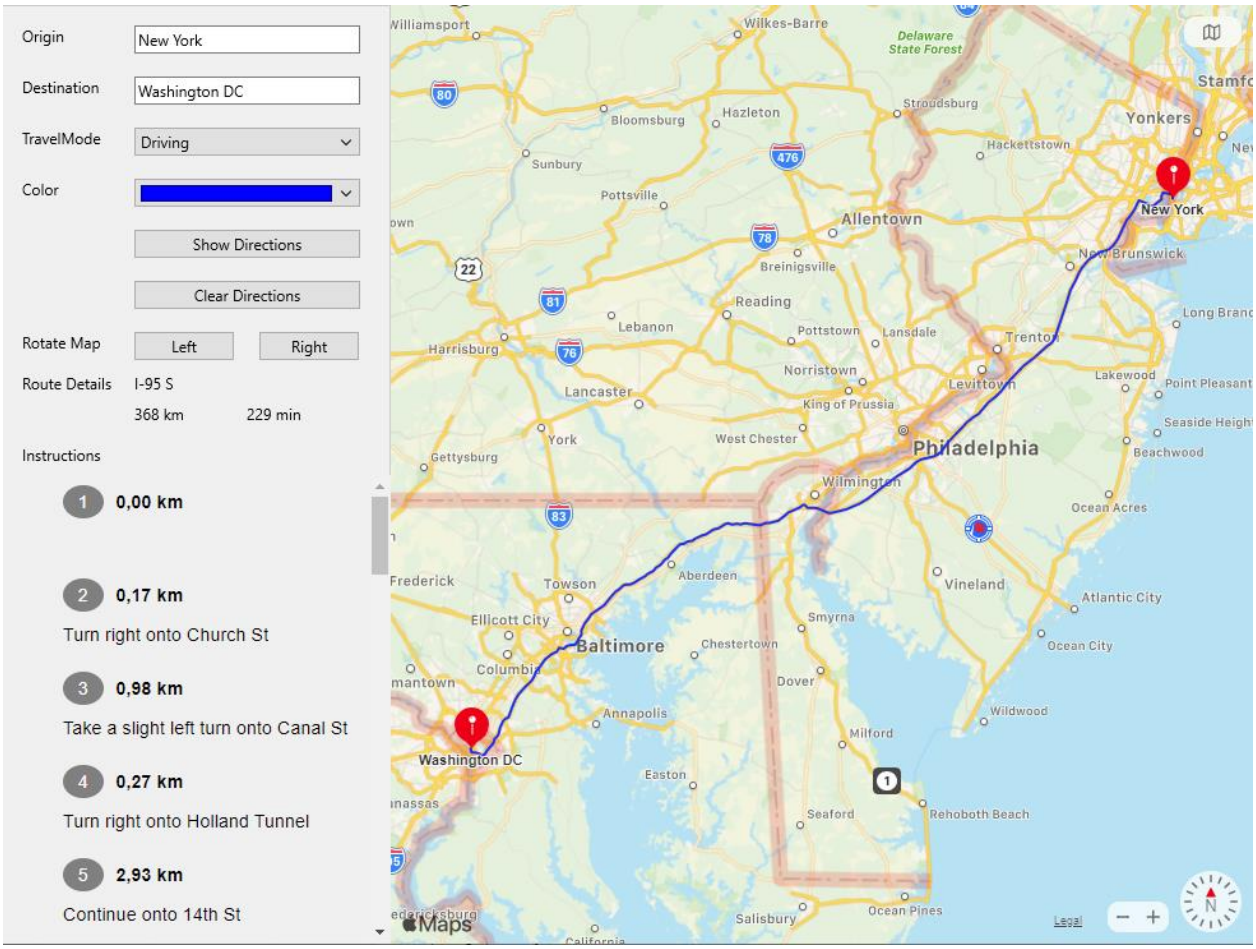
AddDirections
Calculates and displays directions between "Origin" and "Destination" location specified with a string address or a latitude and longitude coordinate.

Parameters:

AOrigin: Specifies the origin address location as a string
ADestination: Specifies the destination address location as a string
AOriginLatitude: Specifies the origin location latitude coordinate
AOriginLongitude: Specifies the origin location longitude coordinate
ADestinationLatitude: Specifies the destination location latitude coordinate
ADestinationLongitude: Specifies the destination location longitude coordinate
AShowMarkers: Indicates if markers are displayed at the origin and destination location
AShowPolyline: Indicates if a polyline indicating the route between origin and destination is displayed
AStrokeColor: Specifies the color of the polyline
AStrokeWidht: Specifies the width of the polyline
AStrokeOpacity: Specifies the opacity of the polyline
ATravelMode: Specifies which travel mode to use when calculating directions. Available options are Driving and Walking.

Directions
Collection containing a list of previous requested directions.
Removing an item in the collection will automatically remove it from the map.

OnRetrievedDirectionsData
Event triggered when directions calculation was successful. A subset of the directions data is available in ADirectionsData. The full directions data is available in AEventData.CustomData as a JSON object.



TTMSFNCMapBox

TTMSFNCMapBox is a descendant of TTMSFNCMaps and exposes extra functionality specific for the MapBox API.

Map Style

When using TTMSFNCMapBox, you are able to specify a style to change the visuals of the map.

Options.MapStyle
Programmatically set a specific map style.

TTMSFNCTomTom

TTMSFNCTomTom is a descendant of TTMSFNCMaps and exposes extra functionality specific for the TomTom Maps API.

Map Style

When using TTMSFNCTomTom, you are able to specify a style to change the visuals of the map.

Options.MapStyle
Programmatically set a specific map style.

TTMSFNCGeocoding

Usage

TTMSFNCGeocoding is a component to perform geocoding of an address or reverse geocoding of a coordinate by using an existing REST API service.

Currently available services are:

- Google
- Here
- Azure
- Bing
- TomTom
- MapBox
- OpenStreetMap Nominatim
- OpenRouteService
- GeoApify

Authorization information

API Key (Except OpenStreetMap Nominatim)

Organisation

Properties

APIKey
Sets the API Key for the service

GeocodingRequests
Holds a list of geocoding/reversegeocoding results

Service
Indicates which service to use to retrieve geocoding/reversegeocoding information

Methods

```
GetGeocoding(AAddress: string; ACallback:
TTMSFNCGeocodingGetGeocodingCallBack = nil; AID: string = ''; ADataPointer:
Pointer = nil);
```
Retrieves a coordinate based on the provided AAddress

```
GetReverseGeocoding(ACoordinates: TTMSFNCMapsCoordinateRec; ACallback:
TTMSFNCGeocodingGetGeocodingCallBack = nil; AID: string = ''; ADataPointer:
Pointer = nil);
```
Retrieves an address based on the provided ACoordinates

Events

```
OnGetGeocoding
```

Triggered after calling GetGeocoding. Contains the results of the geocoding action. The ARequest.Status and ARequest.ErrorMessage values can be used to review the status and error message (if any) returned by the selected service. In case of multiple Geocoding calls, the ARequest.ID value can be used to determine which call the results are associated with.
The data returned from the request can be found in the ARequest.Items collection.

`OnGetReverseGeocoding`
Triggered after calling GetReverseGeocoding. Contains the results of the reversegeocoding action. The ARequest.Status and ARequest.ErrorMessage values can be used to review the status and error message (if any) returned by the selected service. In case of multiple GetReverseGeocoding calls, the ARequest.ID value can be used to determine which call the results are associated with.
The data returned from the request can be found in the ARequest.Items collection.

`OnGetGeocodingResult`
Triggered after calling GetGeocoding. Contains the results of the geocoding action.
The data returned from the request can be found in the AResult.Items collection.

`OnGetReverseGeocodingResult`
Triggered after calling GetReverseGeocoding. Contains the results of the reversegeocoding action.
The data returned from the request can be found in the AResult.Items collection.

Sample

- Sample using OnGetGeocoding event

```
public
  TTMSFNCGeocoding1: TTMSFNCGeocoding;
  procedure TTMSFNCGeocoding1GetGeocoding(Sender: TObject; const ARequest:
TTMSFNCGeocodingRequest; const ARequestResult:
TTMSFNCCloudBaseRequestResult);

procedure TForm1.FormCreate(Sender: TObject);
begin
  TTMSFNCGeocoding1:= TTMSFNCGeocoding.Create(Self);
  TTMSFNCGeocoding1.OnGetGeocoding := TTMSFNCGeocoding1GetGeocoding;
  TTMSFNCGeocoding1.APIKey := 'abc123';
  TTMSFNCGeocoding1.Service := gsGoogle;
  TTMSFNCGeocoding1.GetGeocoding('New York', nil, 'origin');
  TTMSFNCGeocoding1.GetGeocoding('Philadelphia', nil, 'destination');
end;

procedure TForm1.TTMSFNCGeocoding1GetGeocoding(Sender: TObject; const
ARequest: TTMSFNCGeocodingRequest;
  const ARequestResult: TTMSFNCCloudBaseRequestResult);
var
  I: Integer;
  it: TTMSFNCGeocodingItem;
  FStartAddress, FEndAddress: TTMSFNCMapsCoordinateRec;
begin
  for I := 0 to ARequest.Items.Count - 1 do
  begin
    it := ARequest.Items[I];

    if ARequest.ID = 'origin' then
```

```
    begin
      FStartAddress := it.Coordinate.ToRec;
      TMSFNCMaps1.SetCenterCoordinate(FStartAddress);
      TMSFNCMaps1.AddMarker(FStartAddress);
    end;

    if ARequest.ID = 'destination' then
    begin
      FEndAddress := it.Coordinate.ToRec;
      TMSFNCMaps1.AddMarker(FEndAddress);
    end;
  end;
end;
```

- Sample using GetGeocoding with callback

```
public
  TTMSFNCGeocoding1: TTMSFNCGeocoding;

procedure TForm1.FormCreate(Sender: TObject);
var
  FStartAddress, FEndAddress: TTMSFNCMapsCoordinateRec;
begin
  TTMSFNCGeocoding1 := TTMSFNCGeocoding.Create(Self);
  TTMSFNCGeocoding1.APIKey := 'abc123';
  TTMSFNCGeocoding1.Service := gsGoogle;

  TTMSFNCGeocoding1.GetGeocoding('New York',
  procedure(const ARequest: TTMSFNCGeocodingRequest; const ARequestResult:
TTMSFNCCloudBaseRequestResult)
  begin
    FStartAddress := ARequest.Items[0].Coordinate.ToRec;
  end
  );

  TTMSFNCGeocoding1.GetGeocoding('Philadelphia',
  procedure(const ARequest: TTMSFNCGeocodingRequest; const ARequestResult:
TTMSFNCCloudBaseRequestResult)
  begin
    FEndAddress := ARequest.Items[0].Coordinate.ToRec;
  end
  );
end;
```

- Sample using GetReverseGeocoding with OnGetReverseGeocodingResult event

```
procedure TForm1.TMSFNCMaps1MapClick(Sender: TObject;
  AEventData: TTMSFNCMapsEventData);
begin
  TMSFNCGeocoding1.GetReverseGeocoding(AEventData.Coordinate.ToRec);
end;

procedure TForm1.TMSFNCGeocoding1GetReverseGeocodingResult(Sender: TObject;
```

```
  const AResult: TTMSFNCGeocodingRequest);
begin
  if AResult.Items.Count > 0 then
  begin
    TTMSFNCUtils.Log(AResult.Items[0].Street);
    TTMSFNCUtils.Log(AResult.Items[0].City);
    TTMSFNCUtils.Log(AResult.Items[0].PostalCode);
    TTMSFNCUtils.Log(AResult.Items[0].Region);
    TTMSFNCUtils.Log(AResult.Items[0].Country);
  end;
end;
```

TTMSFNCDirections

Usage

TTMSFNCDirections is a component for retrieving directions, route information and step by step instructions for origin and destination coordinates by using an existing REST API service. The component can be used in combination with TTMSFNCGeocoding to retrieve directions between an origin and destination address.

Currently available services are:

- Google
- Here
- Azure
- Bing
- TomTom
- MapBox
- OpenRouteService
- GeoApify

Note: Google Directions and GeoApify are not supported in TMS WEB Core applications due to technical limitations of the service.

Authorization information

API Key

Organisation

Properties

APIKey
Sets the API Key for the service

DirectionsRequests
Holds a list of directions results

Service
Indicates which service to use to retrieve directions

Methods

```
GetDirections(AOrigin, ADestination: TTMSFNCMapsCoordinateRec; ACallback:
TTMSFNCDirectionsGetDirectionsCallback = nil; AID: string = '';
ADataPointer: Pointer = nil; AAlternatives: Boolean = False; ATravelMode:
TTMSFNCDirectionsTravelMode = tmDriving; AWayPoints:
TTMSFNCMapsCoordinateRecArray = nil; AOptimizeWayPoints: Boolean = False;
ALocale: string = ''; AMode: TTMSFNCMapsLocaleMode = mlmDefault;
AAvoidTolls: Boolean = False);
```

Retrieves directions based on the provided AOrigin and ADestination coordinates.

Optional parameters:

AAlternatives: indicates if alternative routes are included in the results.

Note: OpenRouteService only supports alternative routes if the route distance is lower than 100 km.

ATravelMode: indicates which travel mode to use for the directions.
Available modes:

- tmDriving
- tmWalking
- tmBicycling (only available for Azure, Google, Here, MapBox, TomTom, OpenRouteService)
- tmPublicTransport (only available for Bing, Google, Here)
- tmTruck (only available for Azure, Here, TomTom, OpenRouteService)

If a TravelMode is selected that is unavailable for the currently active service, the option will revert to the default mode (tmDriving). The TravelMode that was used is returned in the OnGetDirections event.

AWayPoints: optionally provide a list of waypoints

AOptimizeWayPoints: indicates if waypoints are included in the original or optimized order (only available for Google, Azure, TomTom)

ALocale: indicates the locale of the step by step instructions

AAvoidTolls: avoids using toll roads/bridges in directions
(only available for Azure, Bing, Google, Here, MapBox)

Events

`OnGetDirections`
Triggered after calling GetDirections. Contains the results of the directions request. The ARequest.Status and ARequest.ErrorMessage values can be used to review the status and error message (if any) returned by the selected service. In case of multiple GetDirections calls, the ARequest.ID value can be used to determine which call the results are associated with.
The TravelMode that was used to determine the directions is returned in the ARequestResult.TravelMode value.

Sample

- Sample using OnGetDirections event

```
public
  TTMSFNCDirections1: TTMSFNCDirections;
  procedure TMSFNCDirections1GetDirections(Sender: TObject; const ARequest:
TTMSFNCDirectionsRequest; const ARequestResult:
TTMSFNCCloudBaseRequestResult);

procedure TForm1.FormCreate(Sender: TObject);
var
  FStartAddress, FEndAddress: TTMSFNCMapsCoordinateRec;
begin
  TTMSFNCDirections1:= TTMSFNCDirections.Create(Self);
  TTMSFNCDirections1.OnGetDirections := TMSFNCDirections1GetDirections;
  TTMSFNCDirections1.APIKey := 'abc123';
```

```
  TTMSFNCDirections1.Service := dsGoogle;
  TTMSFNCDirections1.GetDirections(FStartAddress, FEndAddress, nil,
'myroute', nil, False, tmDriving, nil, False, 'en-EN', mlmCountry);
end;

procedure TForm1.TMSFNCDirections1GetDirections(Sender: TObject; const
ARequest: TTMSFNCDirectionsRequest;
  const ARequestResult: TTMSFNCCloudBaseRequestResult);
var
  its: TTMSFNCDirectionsItems;
  it: TTMSFNCDirectionsItem;
  I: Integer;
begin
  TMSFNCMaps1.BeginUpdate;
  its := ARequest.Items;
  for I := 0 to its.Count - 1 do
  begin
    it := its[I];
    if ARequest.ID = 'myroute' then
    begin
      if it.Coordinates.Count > 0 then
        TMSFNCMaps1.AddPolyline(it.Coordinates.ToArray).StrokeColor :=
gcRed;
    end;
  end;
  TMSFNCMaps1.EndUpdate;
end;
```

TTMSFNCLocation

Usage

TTMSFNCLocation is a component for retrieving the current geographical location by using an existing REST API service.

Currently available services are:

- Google
- IPStack

Authorization information

API Key

Organisation

Properties

APIKey
Sets the API Key for the service

LocationRequests
Holds a list of location results

Service
Indicates which service to use to retrieve the location

Connection
Indicates if an HTTP or HTTPS connection should be used when the Service is set to IPStack. Note that the free tier for the IPStack service only supports HTTP.

Methods

```
GetLocation(ACallback: TTMSFNCLocationGetLocationCallBack = nil; AID:
string = '');
```
Retrieves the current geographical location.

Events

`OnGetLocation`
Triggered after calling GetLocation. Contains the results of the location request. In case of multiple GetLocation calls, the ARequest.ID value can be used to determine which call the results are associated with.

Sample

- Sample using OnGetLocation event

```
public
  TTMSFNCLocation1: TTMSFNCLocation;
  procedure TTMSFNCLocation1GetLocation(Sender: TObject; const ARequest:
TTMSFNCLocationRequest; const ARequestResult:
TTMSFNCCloudBaseRequestResult);

procedure TForm1.FormCreate(Sender: TObject);
begin
  TTMSFNCLocation1 := TTMSFNCLocation.Create(Self);
  TTMSFNCLocation1.OnGetLocation := TTMSFNCLocation1GetLocation;
  TTMSFNCLocation1.APIKey := 'abc123';
  TTMSFNCLocation1.Service := lsGoogle;
  TTMSFNCLocation1.GetLocation;
end;

procedure TForm1.TTMSFNCLocation1GetLocation(Sender: TObject;
  const ARequest: TTMSFNCLocationRequest;
  const ARequestResult: TTMSFNCCloudBaseRequestResult);
begin
  TMSFNCMaps1.SetCenterCoordinate(ARequest.Coordinate.ToRec);
  TMSFNCMaps1.AddMarker(ARequest.Coordinate.ToRec);
end;
```

TTMSFNCElevation

Usage

TTMSFNCElevation is a component for retrieving the elevation for a specified geographical location by using an existing REST API service.

Currently available services are:

- Google
- MapQuest
- AirMap (no API Key required)

Authorization information

API Key

Organisation

Properties

APIKey
Sets the API Key for the service

ElevationData
Holds a list of elevation results. The data remains available between GetElevation calls until ClearElevationData is called.

Service
Indicates which service to use to retrieve the elevation

Methods

```
GetElevation(ACoordinate: TTMSNFCMapsCoordinateRec; ACallback:
TTMSFNCLocationGetLocationCallBack = nil);
```
Retrieves the elevation for a geographical location.

```
GetElevation(ACoordinates: TTMSNFCMapsCoordinateRecArray; ACallback:
TTMSFNCLocationGetLocationCallBack = nil);
```
Retrieves the elevation for a list of geographical locations.

```
FindElevation(var ACoordinate: TTMSFNCMapsCoordinateRec): Boolean;
```
Returns true if the provided Coordinate is found in the ElevationData list, false otherwise.
If true, the elevation data is added to the TTMSFNCMapsCoordinateRec.

```
ClearElevationData;
```
Removes all data from ElevationData.

Events

OnGetElevation
Triggered after calling GetElevation. The returned data is contained in ElevationData together with the data from any previous requests.
Data from the current request is also available in ARequest.Items.
The response includes the elevation for each point, in meters.
Only when using the Google elevation service: the resolution value (the maximum distance between data points from which the elevation was interpolated, in meters) is included as well.

Sample

- Sample using OnGetElevation event

```
  public
    co: TTMSFNCMapsCoordinateRec;
    TMSFNCElevation1: TTMSFNCElevation;
    procedure TTMSFNCElevation1GetElevation(Sender: TObject; const
ARequest: TTMSFNCElevationRequest; const ARequestResult:
TTMSFNCCloudBaseRequestResult);

procedure TForm1.Button1Click(Sender: TObject);
begin
  co.Latitude := 50.89096;
  co.Longitude := 3.05186;

  TMSFNCElevation1 := TTMSFNCElevation.Create(Self);
  TMSFNCElevation1.OnGetElevation := TTMSFNCElevation1GetElevation;
  TMSFNCElevation1.APIKey := 'abc123';
  TMSFNCElevation1.Service := esGoogle;
  TMSFNCElevation1.GetElevation(co);
end;

procedure TForm1.TTMSFNCElevation1GetElevation(Sender: TObject;
  const ARequest: TTMSFNCElevationRequest;
  const ARequestResult: TTMSFNCCloudBaseRequestResult);
begin
  if TMSFNCElevation1.FindElevation(co) then
    TTMSFNCUtils.Log('found elevation: ' + FloatToStr(co.Elevation))
  else
    TTMSFNCUtils.Log('no elevation found: ' + ARequest.Status)
end;
```

TTMSFNCTollCost

Usage

TTMSFNCTollCost is a component for retrieving route toll cost information for origin and destination coordinates by using an existing REST API service. The component can be used in combination with TTMSFNCGeocoding to retrieve toll cost between an origin and destination address.

Currently available services are:

- Here
- PTV xServer

Authorization information

API Key (Here)
UserName + Token + ServiceEndPoint (PTV xServer)

Organisation

Properties

APIKey
Sets the API Key for the service (Here only)

UserName
Sets the username for the service (PTV xServer only)

Token
Sets the token for the service (PTV xServer only)

ServiceEndPoint
Sets a custom endpoint for the service (PTV xServer only). If left empty the default EU test environment endpoint is used.

TollCostRequests
Holds a list of toll cost results

Notes:
- Prices returned by Here include VAT while prices returned by PTV xServer do not include VAT
- Currency returned by Here is taken from the GetTollCost ACurrency parameter; PTV xServer returns the local currency relative to the origin and destination location.
- Steps[].Distance is the actual step distance in meters for Here and the distance in meters from the starting point for PTV xServer
- Steps[].Duration is the actual step duration in seconds for Here and the duration from the starting point for PTV xServer

Service
Indicates which service to use to retrieve toll cost information

Methods

```
GetTollCost(AOrigin, ADestination: TTMSFNCMapsCoordinateRec; ACallback:
TTMSFNCDirectionsGetDirectionsCallback = nil;
      AID: string = ''; ADataPointer: Pointer = nil; ATravelMode:
TTMSFNCTollCostTravelMode = tmDriving; AWayPoints:
TTMSFNCMapsCoordinateRecArray = nil; ACurrency: TTMSFNCTollCostCurrency =
tcEUR; ALocale: string = ''; AMode: TTMSFNCMapsLocaleMode = mlmDefault);
```

Retrieves toll cost information based on the provided AOrigin and ADestination coordinates.

Optional parameters:

ATravelMode: indicates which travel mode to use for the directions.
Available modes:

- tmCar
- tmDeliveryVan
- tmLightTruck
- tmHeavyTruck

AWayPoints: sets an array of waypoint coordinates
ACurrency: sets the currency. Only when using Here, PTV xServer automatically returns the toll cost in the local currency.
ALocale: indicates the locale of the step by step instructions

Events

`OnGetTollCost`
Triggered after calling GetTollCost. Contains the results of the toll cost request. The ARequest.Status and ARequest.ErrorMessage values can be used to review the status and error message (if any) returned by the selected service. In case of multiple GetTollCost calls, the ARequest.ID value can be used to determine which call the results are associated with.

Sample

- Sample using OnGetTollCost event

```
public
  TMSFNCTollCost1: TTMSFNCTollCost;
  procedure TMSFNCTollCost1GetTollCost(Sender: TObject;
    const ARequest: TTMSFNCTollCostRequest;
    const ARequestResult: TTMSFNCCloudBaseRequestResult);

procedure TForm1.FormCreate(Sender: TObject);
var
  FStartAddress, FEndAddress: TTMSFNCMapsCoordinateRec;
begin
  TMSFNCTollCost1 := TTMSFNCDirections.Create(Self);
  TMSFNCTollCost1.OnGetTollCost := TMSFNCDirections1GetTollCost;
  TMSFNCTollCost1.APIKey := 'abc123';
  TMSFNCTollCost1.GetTollCost(FStartAddress, FEndAddress, nil, 'myroute',
nil, tmDriving, nil, tcEUR, 'en-EN');
end;
```

```
procedure TForm1.TMSFNCTollCost1GetTollCost(Sender: TObject; const
ARequest: TTMSFNCDirectionsRequest;
  const ARequestResult: TTMSFNCCloudBaseRequestResult);
var
  its: TTMSFNCTollCostItems;
  it: TTMSFNCTollCostItem;
  I: Integer;
begin
  TMSFNCMaps1.BeginUpdate;
  its := ARequest.Items;
  for I := 0 to its.Count - 1 do
  begin
    it := its[I];
    if ARequest.ID = 'myroute' then
    begin
      lbTotalCost.Text := FloatToStr(it.TotalCost);
      if it.Coordinates.Count > 0 then
        TMSFNCMaps1.AddPolyline(it.Coordinates.ToArray);
    end;
  end;
  TMSFNCMaps1.EndUpdate;
end;
```

TTMSFNCStaticMap

Usage

TTMSFNCStaticMap is a component for retrieving an URL to a static map image for a specified geographical location by using an existing API service.

Note: The TTMSFNCMapsImage component can be used to display the static map image URL and/or to save the image to a local file.

Currently available services are:

- Azure
- Bing
- Google
- Here
- MapBox
- TomTom

Authorization information

API Key

Organisation

Properties

APIKey
Sets the API Key for the service

Service
Indicates which service to use to retrieve the static map image URL

Methods

```
GetStaticMap(ACoordinate: TTMSFNCMapsCoordinateRec; AWidth: Single = 640;
AHeight: Single = 480; AZoom: Single = 10; AShowMarker: Boolean = True;
AMapType: TTMSFNCStaticMapType = mtDefault): string;

GetStaticMap(AAddress: string; AWidth: Single = 640; AHeight: Single = 480;
AZoom: Single = 10; AShowMarker: Boolean = True; AMapType:
TTMSFNCStaticMapType = mtDefault): string;
```

Returns a static map image URL based on the provided parameter values.

Notes:
- ACoordinate and AAdress are mutually exclusive parameters
- AAddress is only available for Google and Here
- The TTMSFNCGeocoding component can be used to convert a given address to a coordinate

ACoordinate: Indicates the center coordinate of the map
AAddress: Indicates the center address of the map
AWidth: Indicates the Width of the image
AHeight: Indicates the Height of the image
AZoom: Indicates the zoom level of the map
AShowMarker: Indicates if a marker is displayed at the ACoordinate or AAddress (Not available for TomTom)
AMapType: Indicates the map Type. Available map types: Default, Satellite, Hybrid, Terrain, Light and Dark. (Not all map types are available for all map services)

The minimum and maximum map image size in pixels as indicated by the respective services:

	Min Height	Max Height	Min Width	Max Width
Azure	/	8192	/	8192
Bing	80	2000	80	1500
Google	/	640	/	640
Here	/	2048	/	2048
MapBox	1	1280	1	1280
TomTom	1	8192	1	8192

Sample

- Sample using a combination of TTMSFNCStaticMap, TTMSFNCMapsImage and TTMSFNCGeocoding

```
procedure TForm1.GetCenterCoordinate;
begin
  TMSFNCGeocoding1.GeocodingRequests.Clear;
  TMSFNCGeocoding1.GetGeocoding('New York');
end;

procedure TForm1.TMSFNCGeocoding1GetGeocoding(Sender: TObject;
  const ARequest: TTMSFNCGeocodingRequest;
  const ARequestResult: TTMSFNCCloudBaseRequestResult);
begin
  if TMSFNCGeocoding1.GeocodingRequests.Count > 0 then
  begin
    TMSFNCMapsImage1.URL :=
TMSFNCStaticMap1.GetStaticMap(TMSFNCGeocoding1.GeocodingRequests[0].Items[0
].Coordinate.ToRec, TMSFNCMapsImage1.Width, TMSFNCMapsImage1.Height);
  end;
end;
```

TTMSFNCMapsImage

Usage

TTMSFNCMapsImage is an Image component that can automatically retrieve an image from an URL and display it on the form. It supports BMP, PNG, GIF, ICO, JPEG formats.

Organisation

Properties

URL
Sets the URL where to retrieve the image from

Samples

- Save the remote image referenced by the URL property value to a local image file

```
TMSFNCMapsImage1.GetBitmap.SaveToFile('c:\image.png');
```

TTMSFNCPlaces

Usage

TTMSFNCPlaces is a component for retrieving autocomplete suggestions for a search query by using an existing REST API service. An optional location can be included to bias results based on a geographic coordinate.

Currently available services are:

- Google
- Here
- Azure
- Bing
- GeoApify

Note: Google Places is not supported in TMS WEB Core applications due to technical limitations of the service.

Authorization information

API Key

Organisation

Properties

APIKey
Sets the API Key for the service

PlacesRequests
Holds a list of search results.
Note:
Not all properties are returned by all available services. Some properties are only returned by methods from TTMSFNCGooglePlaces.

Service
Indicates which service to use

Methods

`GetAutoComplete`
Retrieves autocomplete suggestions based on the ASearch string query.

ASearch: string keyword
ALocation: optional geographic coordinate. Search results will be biased on this location.
ALocale: indicates the locale of the search results

Events

OnGetAutoComplete
Triggered after calling GetAutoComplete. Contains the results of the autocomplete request in ARequest.Items.
The ARequest.Status and ARequest.ErrorMessage values can be used to review the status and error message (if any) returned by the selected service. In case of multiple GetAutoComplete calls, the ARequest.ID value can be used to determine which call the results are associated with.

Sample

- Sample using OnGetAutoComplete event

```
public
  TTMSFNCPlaces1: TTMSFNCPlaces;
  procedure TMSFNCPlaces1GetAutoComplete(Sender: TObject;
  const ARequest: TTMSFNCPlacesRequest;
  const ARequestResult: TTMSFNCCloudBaseRequestResult);

procedure TForm1.FormCreate(Sender: TObject);
begin
  TTMSFNCPlaces1:= TTMSFNCPlaces.Create(Self);
  TTMSFNCPlaces1.OnGetAutoComplete := TMSFNCPlaces1GetAutoComplete;
  TTMSFNCPlaces1.APIKey := 'abc123';
  TTMSFNCPlaces1.Service := dsGoogle;
  TTMSFNCPlaces1.GetAutoComplete('New York');
end;

procedure TForm1.TMSFNCPlaces1GetAutoComplete(Sender: TObject; const
ARequest: TTMSFNCPlacesRequest;
  const ARequestResult: TTMSFNCCloudBaseRequestResult);
var
  I: Integer;
begin
  for I := 0 to ARequest.Items.Count - 1 do
  begin
    ListBox1.Items.Add(ARequest.Items[I].Address);
  end;
end;
```

TTMSFNCGooglePlaces

Usage

TTMSFNCGooglePlaces is an extended version of TTMSFNCPlaces that adds specific functionality for Google Places only.

Note: Google Places is not supported in TMS WEB Core applications due to technical limitations of the service.

Authorization information

API Key

Organisation

Properties

APIKey
Sets the API Key for the service

PlacesRequests
Holds a list of search results
Note:
Not all properties are returned by all methods.

Methods

`GetAutoComplete`
Retrieves places suggestions based on a partial search query.
ASearch: string keyword
ALocation: optional geographic coordinate. Search results will be biased on this location.
ALocale: indicates the locale of the search results

`SearchByText`
Search for places based on a string query and a location
ASearch: string keyword
ALocation: geographic coordinate

`SearchNearby`
Search for places near to a specified location
ALocation: geographic coordinate

`SearchNearbyNextPage`
Get the more results from a previous SearchNearby call.
The SearchNearby call returns a limited amount of results. The SearchNearbyNextPage can be used to retrieve more results, if any, based on the last SearchNearby call.

`GetPlaceDetail`
Get detailed data, including images where available, for a specified place or place id.
APlace: a TTMSFNCPlacesItem to get details for
APlaceID: a place id to get details for

(Both parameters are mutually exclusive)

Events

`OnGetAutoComplete`
Triggered after calling GetAutoComplete.

`OnSearchByText`
Triggered after calling SearchByText.

`OnSearchNearby`
Triggered after calling SearchNearby or SearchNearbyNextPage.

`OnGetPlaceDetail`
Triggered after calling GetPlaceDetail.

Note:
Contains the results of the request in ARequest.Items.
The ARequest.Status and ARequest.ErrorMessage values can be used to review the status and error message (if any) returned by the selected service. In case the method is executed multiple times, the ARequest.ID value can be used to determine which call the results are associated with.

TTMSFNCTimeZone

Usage

TTMSFNCTimeZone is a component for retrieving time zone information for a specified geographical location by using an existing API service.

Currently available services are:

- Azure
- Bing
- Google
- Here

Authorization information

API Key

Organisation

Properties

APIKey
Sets the API Key for the service

Service
Indicates which service to use

TimeZoneRequests
A collection containing the results of the GetTimeZone calls.

TimeZone: The ID or name of the time zone
Description: The description of the time zone
Offset: Time offset relative to UTC
DSTOffset: Daylight Savings Time offset relative to UTC

Note
The time zone information content depends on the selected service.
Not all services return a description and/or a DST offset. While Google returns the time offset in seconds, other services return a formatted time string.

Methods

```
GetTimeZone(ACoordinate: TTMSFNCMapsCoordinateRec; ADateTime: TDateTime =
0; ALocale: string = ''; AMode: TTMSFNCMapsLocaleMode = mlmDefault;
ACallback: TTMSFNCTimeZoneGetTimeZoneCallBack = nil; AID: string = '';
ADataPointer: Pointer = nil);
```

Initiates a TimeZone request with the selected service based on the provided geographical location. The OnGetTimeZone event is triggered after the request has finished and the result data is contained in the TimeZoneRequests collection.

Notes
- ACoordinate and AAdress are mutually exclusive parameters
- AAddress is only available for Here and Bing services
- The TTMSFNCGeocoding component can be used to convert a given address to a coordinate

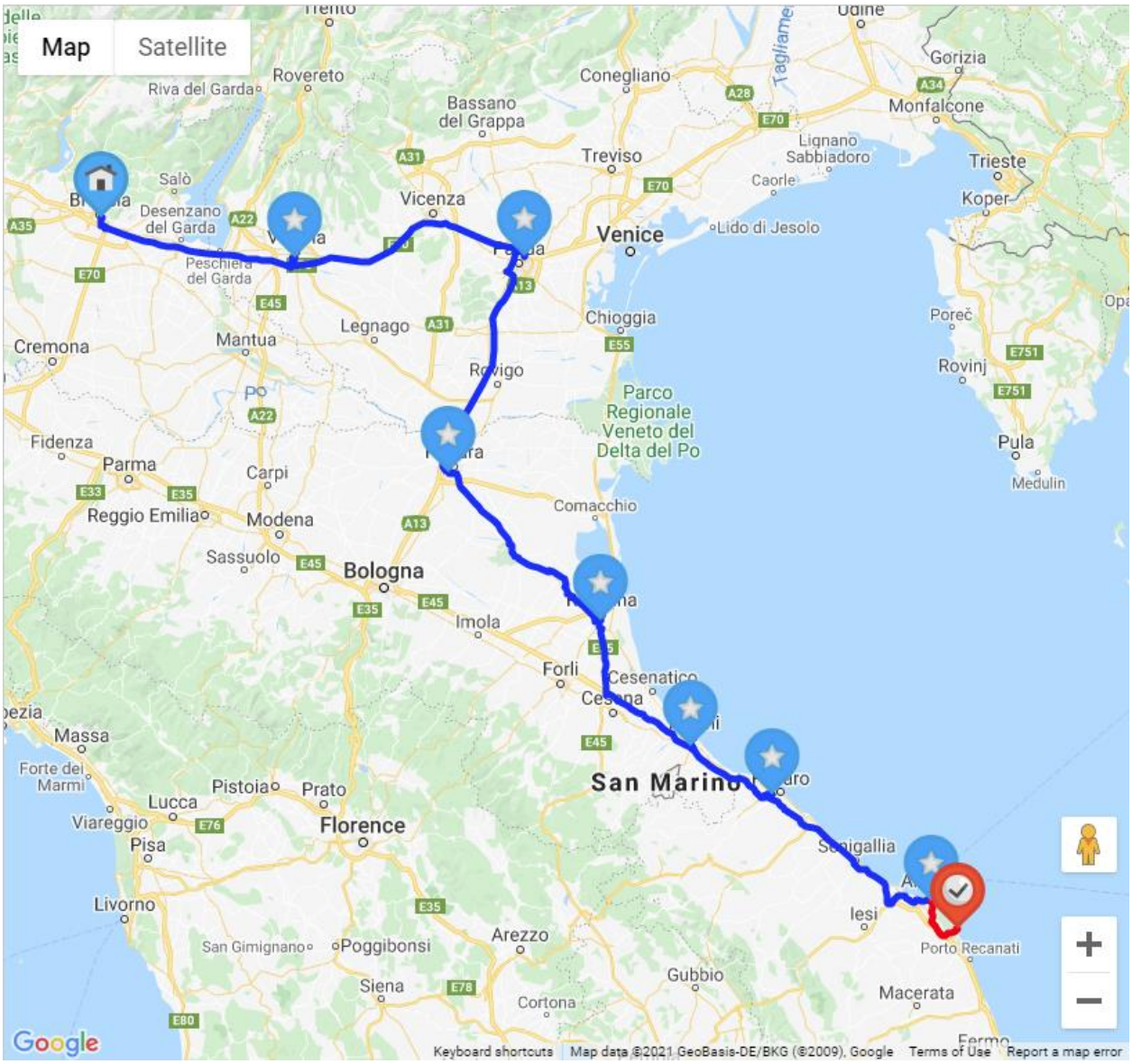
ACoordinate: Indicates the geographical location to request the time zone for
AAddress: Indicates the address to request the time zone for
ADateTime: Indicates a specific date and time to calculate the time zone for. If no value is specified the current date and time are used (Google and Bing only)
ALocale: Indicates the locale used for the time zone information (Google only)

Sample

- Sample using a combination of TTMSFNCTimeZone and TTMSFNCGeocoding

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  TMSFNCGeocoding1.GetGeocoding('New York');
end;

procedure TForm1.TMSFNCGeocoding1GetGeocoding(Sender: TObject;
  const ARequest: TTMSFNCGeocodingRequest;
  const ARequestResult: TTMSFNCCloudBaseRequestResult);
begin
  if ARequest.Items.Count > 0 then
    TMSFNCTimeZone1.GetTimeZone(ARequest.Items[0].Coordinate.toRec);
end;

procedure TForm1.TMSFNCTimeZone1GetTimeZone(Sender: TObject;
  const ARequest: TTMSFNCTimeZoneRequest;
  const ARequestResult: TTMSFNCCloudBaseRequestResult);
begin
  if ARequest.Items.Count > 0 then
    TTMSFNCUtils.Log('TimeZone: ' + ARequest.Items[0].TimeZone);
end;
```

TTMSFNCRouteCalculator



Usage

TTMSFNCRouteCalculator is a component for constructing and editing custom routes by using an existing geocoding & directions API service.

Currently available services are:

- Azure
- Bing
- Google
- Here
- MapBox

- OpenRouteService
- GeoApify

Note: Google Directions is not supported in TMS WEB Core applications due to technical limitations of the service.

The TTMSFNCRouteCalculator can be connected to a TTMSFNCMaps to automatically construct, display and edit routes by interacting with the map.

Note
Supported mapping services usable in combination with TTMSFNCRouteCalculator are:

- Google
- Here
- OpenLayers

Authorization information

API Key

Organisation

Properties

APIKey
Sets the API Key for the service

Service
Indicates which service to use

Active
Indicates that the TTMSFNCRouteCalculator can be used by interacting with the connected TTMSFNCMaps

Options
- AvoidTolls: Avoid toll roads when calculating routes
- IncludeAlternativeRoutes: Include alternative routes when calculating routes.

  Note:
  The alternative routes are returned with the OnGetRouteDirections event, they are not included in the Routes collection.

- TravelMode: Set the travel mode used when calculating routes
- Polyline: Set how the route is displayed when connected to TTMSFNCMaps. The marker value can be an URL to an image file or a base64 encoded image. If no value is provided a default image is used instead.
  - StartMarker: Set a custom value for the route's start location marker
  - EndMarker: Set a custom value for the route's end location marker
  - WayPointMarker: Set a custom value for the route's waypoint marker
  - AddWayPointMarker: Set a custom value for the marker displayed while adding a waypoint
  - SelectedWayPointMarker: Set a custom value for a marker in selected state

  - StrokeColor: Set the color of the route's polylines
  - StrokeWidth: Set the width the route's polylines
  - StrokeOpacity: Set the opacity of the route's polylines
  - SelectedStrokeColor: Set the color of a selected polyline
- HistoryEnabled: Enable the history manager to be able to use undo/redo functionality

Routes
A collection of currently available routes

HistoryManager
Manages the history of the current route.
Use HistoryManager.Undo to reverse the last change and HistoryManager.Redo to restore the last change that was reversed by Undo.

Methods

Note
Methods that rely on HTTP requests to a REST API service are executed asynchronously. When the request is finished, an event (see below) is triggered. Alternatively you can use an anonymous method in combination with the ACallback parameter.

`CalculateRoute`
Provide the start and end addres or start and end coordinate to create a new route.

- `CalculateRoute(AStartAddress, AEndAddress: string; ACallback: TTMSFNCRouteCalculatorCalculateRouteCallback = nil; AID: string = ''; ADataPointer: Pointer = nil);`

- `CalculateRoute(AStartCoordinate, AEndCoordinate: TTMSFNCMapsCoordinateRec; ACallback: TTMSFNCRouteCalculatorCalculateRouteCallback = nil; AWayPoints: TTMSFNCMapsCoordinateRecArray = nil; AID: string = ''; ADataPointer: Pointer = nil);`

`AddRouteSegment`
Add an extra segment to an existing route by providing a new end address or end coordinate.

- `AddRouteSegment(ARoute: TTMSFNCRouteCalculatorRoute; AEndAddress: string; ACallback: TTMSFNCRouteCalculatorCalculateRouteCallback = nil; AID: string = ''; ADataPointer: Pointer = nil);`

- `AddRouteSegment(ARoute: TTMSFNCRouteCalculatorRoute; AEndCoordinate: TTMSFNCMapsCoordinateRec; ACallback: TTMSFNCRouteCalculatorCalculateRouteCallback = nil; AID: string = ''; ADataPointer: Pointer = nil);`

`AddWayPointsToSegment`
Add one or more new waypoints to an existing segment of a route.

- `AddWayPointsToSegment(ASegment: TTMSFNCRouteCalculatorSegment; AWayPoints: TTMSFNCMapsCoordinateRecArray; ACallback: TTMSFNCRouteCalculatorCalculateRouteCallback = nil; AID: string = ''; ADataPointer: Pointer = nil);`

UpdateRouteSegment
Update the specified segment of an existing route by providing a new start and end address or start and end coordinate.

- `UpdateRouteSegment(ASegment: TTMSFNCRouteCalculatorSegment; AStartAddress, AEndAddress: string; ACallback: TTMSFNCRouteCalculatorCalculateRouteCallback = nil; AID: string = ''; ADataPointer: Pointer = nil);`

- `UpdateRouteSegment(ASegment: TTMSFNCRouteCalculatorSegment; AStartCoordinate, AEndCoordinate: TTMSFNCMapsCoordinateRec; ACallback: TTMSFNCRouteCalculatorCalculateRouteCallback = nil; AID: string = ''; ADataPointer: Pointer = nil);`

UpdateRouteSegmentStartAddress/Coordinate
Update the specified segment of an existing route by providing a new start address or start coordinate.

- `UpdateRouteSegmentStartAddress(ASegment: TTMSFNCRouteCalculatorSegment; AStartAddress: string; ACallback: TTMSFNCRouteCalculatorCalculateRouteCallback = nil; AID: string = ''; ADataPointer: Pointer = nil);`

- `UpdateRouteSegmentStartCoordinate(ASegment: TTMSFNCRouteCalculatorSegment; AStartCoordinate: TTMSFNCMapsCoordinateRec; ACallback: TTMSFNCRouteCalculatorCalculateRouteCallback = nil; AID: string = ''; ADataPointer: Pointer = nil);`

UpdateRouteSegmentEndAddress/Coordinate
Update the specified segment of an existing route by providing a new end address or end coordinate.

- `UpdateRouteSegmentEndAddress(ASegment: TTMSFNCRouteCalculatorSegment; AEndAddress: string; ACallback: TTMSFNCRouteCalculatorCalculateRouteCallback = nil; AID: string = ''; ADataPointer: Pointer = nil);`

- `UpdateRouteSegmentEndCoordinate(ASegment: TTMSFNCRouteCalculatorSegment; AEndCoordinate: TTMSFNCMapsCoordinateRec; ACallback: TTMSFNCRouteCalculatorCalculateRouteCallback = nil; AID: string = ''; ADataPointer: Pointer = nil);`

DeleteRoute
Delete an existing route.

- `DeleteRoute(ARoute: TTMSFNCRouteCalculatorRoute);`

DeleteRouteSegment
Delete the specified segment from an existing route.

- `DeleteRouteSegment(ASegment: TTMSFNCRouteCalculatorSegment; ACallback: TTMSFNCRouteCalculatorCalculateRouteCallback = nil);`

GetGeocoding / GetReverseGeocoding
Convert the provided address to a coordinate or vice versa.

- `GetGeocoding(AAddress: string; ACallback: TTMSFNCGeocodingGetGeocodingCallBack = nil; AID: string = ''; ADataPointer: Pointer = nil);` virtual;

- `GetReverseGeocoding(ACoordinates: TTMSFNCMapsCoordinateRec; ACallback: TTMSFNCGeocodingGetGeocodingCallBack = nil; AID: string = ''; ADataPointer: Pointer = nil);`

GetDirections
Get directions data for the provided origin and destination coordinates.

- `GetDirections(AOrigin, ADestination: TTMSFNCMapsCoordinateRec; ACallback: TTMSFNCDirectionsGetDirectionsCallBack = nil; AWayPoints: TTMSFNCMapsCoordinateRecArray = nil; AID: string = ''; ADataPointer: Pointer = nil);`

GetCurrentLocation
Get the current physical location of the device. With AMode set to lmDevice, use a TLocationSensor to determine the location. With AMode set to lmService, use the Google REST API service to determine the location.
Note: lmService is currently limited to the Google service only.

- `GetCurrentLocation(AMode: TTMSFNCRouteCalculatorLocationMode = lmDevice);`

LoadGPXFromStream/File/Text
Import a route from an existing GPX data stream, file or text.

- `LoadGPXFromStream(const AStream: TStream; ADoGeocoding: Boolean = False): TTMSFNCMapsGPXRec;`
- `LoadGPXFromFile(AFileName: string; ADoGeocoding: Boolean = False): TTMSFNCMapsGPXRec;`
- `LoadGPXFromText(AText: string; ADoGeocoding: Boolean = False): TTMSFNCMapsGPXRec;`

SaveToGPXStream/File/Text
Export an exsiting route to GPX data stream, file or text. Optionally provide extra metadata.

- `SaveToGPXStream(ARoute: TTMSFNCRouteCalculatorRoute; AStream: TStream; AMetaData: TTMSFNCMapsGPXMetaData);`
- `SaveToGPXFile(ARoute: TTMSFNCRouteCalculatorRoute; AFileName: string; AMetaData: TTMSFNCMapsGPXMetaData);`
- `SaveToGPXText(ARoute: TTMSFNCRouteCalculatorRoute; AMetaData: TTMSFNCMapsGPXMetaData): string;`

SaveRoutesToStream/Text/File
Save an existing route to a stream, text or file.

- `SaveRoutesToStream(AStream: TStream);`
- `SaveRoutesToText: string;`
- `SaveRoutesToFile(AFileName: string);`

LoadRoutesFromStream/Text/File
Load a route from a stream, text or file.

- `LoadRoutesFromStream(AStream: TStream);`
- `LoadRoutesFromText(AText: string);`
- `LoadRoutesFromFile(AFileName: string);`

Events

OnGetGeocoding
Triggered after calling GetGeocoding

OnGetDirections
Triggered after calling GetDirections

OnGetRouteGeocoding
Triggered after geocoding is used in a call to CalculateRoute

OnGetRouteDirections
Triggered after directions are used in a call to CalculateRoute

OnCalculateRoute
Triggered after a call to CalculateRoute, AddRouteSegment, AddRouteWayPoints, UpdateRouteSegment and DeleteRouteSegment

OnCreateGPXTrack
Triggered during GPX import each time a track (route) is added

OnCreateGPXSegment
Triggered during GPX import each time a route segment is added

OnLoadGPXRouteComplete
Triggered after GPX import is complete

OnGetLocation
Triggered after a call to GetCurrentLocation

Interaction with TTMSFNCMaps

Note
Currently supported mapping services usable in combination with TTMSFNCRouteCalculator are:

- Google
- Here
- OpenLayers

Getting started

Use the TMSFNCMaps.RouteCalculator property to assign a TTMSFNCRouteCalculator component.
Set TMSFNCRouteCalculator.Active to True to enable map interaction.
Set TMSFNCRouteCalculator.Options.EnableHistoryManager to True to enable undo/redo functionality.

Map interaction usage

The first click on the map sets the starting location of the route.
Every additional click on the map adds a new waypoint.
Drag markers to change waypoint locations.
Drag polylines to insert new waypoints.
To remove waypoints, select a marker or polyline and press the Del key.

Programmatic usage

Methods

```
RouteCalculatorPlotRoute(ARoute: TTMSFNCRouteCalculatorRoute);
```
Display the provided TTMSFNCRouteCalculatorRoute on the map. If the route is already displayed on the map it will be updated to reflect the latest route settings.

`RouteCalculatorPlotRoutes`
Display all routes from the assigned RouteCalculator on the map. If a route is already displayed on the map it will be updated to reflect the latest route settings.

`RouteCalculatorClearRoutes`
Remove all routes from the assgiend RouteCalculator from the map.

```
RouteCalculatorDeletePolyline(APolyline: TTMSFNCMapsPolyline; AUpdateRoute:
Boolean = True);
```
Remove the RouteCalcaultor route segment associated with the provided polyline from the route and update the map if AUpdateRoute is True.

```
RouteCalculatorDeleteMarker(AMarker: TTMSFNCMapsMarker; AUpdateRoute:
Boolean = True);
```
Remove the RouteCalculator route waypoint associated with the provided marker from the route and update the map if AUpdateRoute is True.

```
RouteCalculatorDeleteSelectedPolyline(AUpdateRoute: Boolean = True);
```
Remove the RouteCalcaultor route segment associated with the currently selected polyline from the route and update the map if AUpdateRoute is True.

```
RouteCalculatorDeleteSelectedMarker(AUpdateRoute: Boolean = True);
```
Remove the RouteCalculator route waypoint associated with the currently selected marker from the route and update the map if AUpdateRoute is True.

Events

`OnRouteCalculatorWayPointAdded`
Triggered when a click on the map occurs and a new waypoint is added to the RouteCalculator Route or when an existing polyline is dragged to insert a new waypoint.

OnRouteCalculatorWayPointUpdated
Triggered when a waypoint marker is dragged to a new location and an existing RouteCalculator waypoint is updated.

`OnRouteCalculatorPolylineAdded`
Triggered when a click on the map occurs and a new segment is added to the RouteCalculator Route or when an existing polyline is dragged to insert a new waypoint.

`OnRouteCalculatorPolylineUpdated`
Triggered when a waypoint marker is dragged to a new location and an existing RouteCalculator segment is updated.

`OnRouteCalculatorBeforeDeletePolyline`
Triggered when an existing route segment (represented by a polyline on the map) is about to get deleted.

`OnRouteCalculatorAfterDeletePolyline`
Triggered after an existing route segment (represented by a polyline on the map) has been deleted.

`OnRouteCalculatorBeforeDeleteMarker`
Triggered when an existing route waypoint (represented by a marker on the map) is about to get deleted.

OnRouteCalculatorAfterDeleteMarker
Triggered when an existing route waypoint (represented by a marker on the map) has been deleted.

Properties

RouteCalculatorSelectedMarker
Get the currently selected waypoint (represented by a marker on the map) if any.

RouteCalculatorSelectedPolyline
Get the currently selected segment (represented by a polyline on the map) if any.

Sample

- Set up TTMSFNCRouteCalculator and TTMSFNCMaps to create routes with map interaction.

```
procedure TForm1.FormCreate(Sender: TObject);
var
  APIKey: string;
begin
  APIKey := 'abc';

  TMSFNCRouteCalculator1.Active := True;
  TMSFNCRouteCalculator1.Options.HistoryEnabled := True;
  TMSFNCRouteCalculator1.Service := csGoogle;
  TMSFNCRouteCalculator1.APIKey := APIKey;

  TMSFNCMaps1.RouteCalculator := TMSFNCRouteCalculator1;
  TMSFNCMaps1.Service := msGoogleMaps;
  TMSFNCMaps1.APIKey := APIKey;
end;
```

Threading

When calling synchronous methods or functions such as XYToLatLon or LatLonToXY inside an event handler, you will get the following message after a short delay:

Running a message loop synchronously in an event handler in Webview can cause reentrancy issue. Please refer to https://docs.microsoft.com/en-us/microsoft-edge/webview2/concepts/threading-model#re-entrancy for more information about threading model in WebView2 and how to enable native code debugging for this scenario.

TTMSFNCMaps based on TTMSFNCWebBrowser in Windows is currently not thread safe, to fix this issue please apply the following workaround:

```
procedure TMapForm.TMSFNCMaps1MapClick(Sender: TObject;
  AEventData: TTMSFNCMapsEventData);
var
  t: ITask;
  c: TTMSFNCMapsCoordinateRec;
  p: TTMSFNCMapsAnchorPointRec;
begin
  c := AEventData.Coordinate.ToRec;

  t := TTask.Create(
  procedure
  begin
    TThread.Synchronize(TThread.Current,
    procedure
    begin
      p := TMSFNCMaps1.LatLonToXY(c.Latitude, c.Longitude);
      TTMSFNCUtils.Log('XY: ' + FloatToStr(p.X) + ' x ' + FloatTostr(p.Y));
    end);
  end
  );
  t.Start;
end;
```

TMS FNC Maps Book

TMS FNC Maps bundles all our expertise with using mapping services, with writing cross-platform and cross-framework Delphi code as well as building components for TMS WEB Core web client applications. From the beginning, we made the radical decision to base this work on the new operating system browser Edge Chromium from Microsoft for the Windows platform (where we already use Safari for iOS and macOS and Chrome for Android). From the very beginning of this development, our chief evangelist Dr. Holger Flick was involved and could play with internal versions and give feedback and steer the development.

If you want to integrate mapping services' functionality such as from Google Maps, Bing Maps, Openstreet Maps, Here maps, TomTom maps, Azure maps in a Windows VCL application, a cross platform FireMonkey application or a TMS WEB Core web client application, TMS FNC Maps addresses it all and the new book covers developing with TMS FNC Maps in great detail over close to 400 pages.

Content

- Get to know the FNC framework, FNC Core, and FNC Maps in a nutshell.
- Learn about FNC Maps in detail: You will find dozens of hands-on examples from installation to modern multi-tier applications with large databases, web services, web, and desktop clients.
- Annotate maps and visualize data using markers with clustering, lines, shapes, pop-up windows and routes.
- Build applications that can handle any geographical data and interact with other Geographic Information Services (GIS). Load GPX tracks from GPS devices, GeoJSON shapes, and Google Earth-specific KML layers.
- Include other mapping services to get directions with turn-by-turn instructions, to geocode addresses, and to determine your geolocation.
- Create reports with a customizable template based on data from maps and databases.
- Allow users to freely customize maps using events or interactive markers and shapes.

The examples in the book make use of:

TMS FlexCel
TMS FNC Core
TMS VCL UI Pack

TMS WEB Core
TMS Cryptography Pack
TMS XData

How to order

Amazon USA
Amazon UK
Amazon Germany

Terms of use

With the purchase of TMS FNC Maps, you are entitled to our consulting and support services to integrate the supported services in Delphi applications and with this consulting and support comes the full source code needed to do this integration. As TMS FNC Maps uses the supported services you are bound to the terms of these services that can be found at:

Google Maps: https://developers.google.com/terms
Microsoft Bing Maps: https://www.microsoft.com/en-us/maps/product
Microsoft Azure Maps: https://azure.microsoft.com/en-in/support/legal/
Openlayers: https://openlayers.org/
TomTom: https://developer.tomtom.com/terms-and-conditions
Here Maps: https://developer.here.com/terms-and-conditions
MapBox: https://www.mapbox.com/legal/service-terms
Apple Mapkit: https://developer.apple.com/terms/
Airmap: https://www.airmap.com/terms-service/
MapQuest: https://developer.mapquest.com/legal

TMS software is not responsible for the use of TMS FNC Maps components. The purchase of TMS FNC Maps does not include any license fee that you might possibly be required to pay to the supported services. It will depend on your type of usage of these services whether a license fee needs to be paid.
It is the sole responsibility of the user or company providing the application that integrates the terms and conditions of the supported services. TMS software does not take any responsibility nor indemnifies any party violating the terms and conditions of the supported services.
We cannot guarantee that a 3rd party service will approve or allow the use of the services used in TMS FNC Maps components in your application(s) now or at any time in the future. This is up to the 3rd party service provider to decide and not under our control. In case the 3rd party service provider makes changes to the conditions for using the service, the API itself or anything else that causes an incompatibility with our component implementations, tmssoftware.com will do its best to accommodate the changes but cannot be forced in any way or within any timeframe to do so and might be technically limited to do so.

Limited warranty

TMS software cannot guarantee the current or future operation and uptime of the supported services.
TMS software offers the consulting and support for TMS FNC Maps in good faith that the supported services are reliable and future-proof.
In no case, TMS software shall offer refunds or any other compensation in case the supported services terms/operation changes or stops.