

Table des matières

ECMap	1
Purchase - Contact	4
Use License	6
Installation	9
TECMap	19
Positioning	25
Map Type	32
Controls	44
Location	46
Places	53
Import / Export	61
Overlays	68
Markers	84
Mobile	94
InfoWindow	100
Labels	105
Roads	108
Layers	122
Pois	132
Groups	138
StreetView	143
EarthView	149
Panoramio	164
DistanceMatrix	170
TECNativeMap	174
Offline mode	199
Vector Tiles	204
Multi view	214
Remote tile server	216
Location	224

Import/Export	239
High resolution	256
Shapes	260
TECShapeMarker	288
TECShapePoi	298
TECShapeLine	303
TECShapePolygone	320
TECShapeInfoWindow	323
Animation	328
Roads	339
Layers	353
Android	377
iOS	381
Autres composants	382
TECCesiumMap	383
TECMapAdressEdit	403
TECStaticMap	405
TECVelib	407
Programming	410
Interactive creating a route	411
Display StreetView in an InfoWindow	426
Display a Panoramio Layer in Bing maps, Leaflet, MapQuest, CloudMade and Google Maps	429
Display a MiniMap	437
Table of Figures	443

ECMap is a suite of components for Delphi, Since version 7, that supports the apis [Google Maps 3](#) , [Google Earth](#), [Bing Maps](#), [CloudMade](#) , [OpenMapQuest](#) et [Leaflet](#)



Fig. 1 2 components, 5 Apis, 3 display engines !

TECMap IE & Chromium

With [TECMap](#) You can use 2 engines display, Internet Explorer and [Google Chromium](#) for the latter you must first install the component [Delphi Chromium Embedded](#)

You then have access to apis web Google maps and Earth, Bing Maps, CloudMade, OpenMapQuest and Leaflet.

With [TECCesiumMap](#) integrate the [globe 3D Cesium](#) in your applications !



TECNativeMap 100% Delphi VCL and Firemonkey

TECNativeMap uses a **100% delphi** rendering, **0% WebBrowser & Javascript** so faster and lighter in memory

Support [offline mode](#), [mappilary](#), [multiple views](#), [screenshot](#) areas even off screen (maximum 8000 x 8000), [animations](#), [KML](#) format ...

Available in VCL (D7 to 10.3 Rio) and Firemonkey (Xe3 to 10.3 Rio) (**Windows , Mac OS X, iOS and Android**)

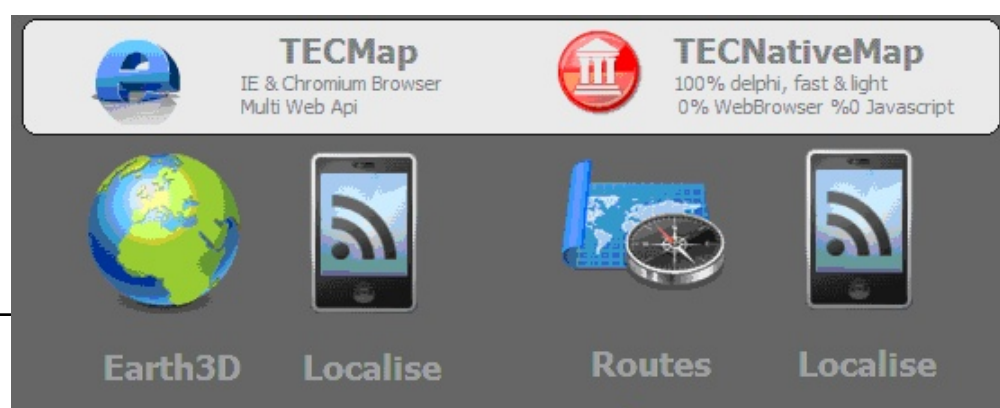
Test a demonstration for Android

You will be able to optimally manage your maps and geographical data using the language you know best, Delphi !

The **sources of the components and demonstrations are delivered** with a chm help and PDF, you can consult the manual directly on the site and get an idea of the capabilities of the component by downloading the [10 demonstrations](#) .

Free *with your license components* [TECCesiumMap](#), [TECMapAdressEdit](#) , [TECStaticMap](#) and [TECVelib](#)

A [trial version](#) of the components [TECMap](#) and [TECNativeMap](#) is available [on request](#)



Identification

ECMap and this site is edited by **Christopher ESCOT-SEP**, entrepreneur registered in Saint-Lô under number **509 737 532** whose registered office is **231 rue des fauvelles, 50000 Saint-Lo , FRANCE**

Documentation

You can download this document in [PDF](#) and [CHM](#) format

Buy

You can pay directly online from the website by clicking the **Buy now** button, check to my name or transfer (make your request by email to get my IBAN)

Full suite TECMap



1 developer licence

500 euros (vat 0%)

The license includes **free updates**, the **entire source code** of the component (**TECMap**, **TECNativeMap**, **TECCesiumMap**, **TECMapAdressEdit**, **TECStaticMap**, **TECVelib**) and **demonstrations**, free support by email for the use of the component

TECNativeMap



1 developer

220 euros (vat 0%)

The license includes **free updates**, the **entire source code** of the component **TECNativeMap** and **demonstrations**, free support by email for the use of the component

Contact Us

infos@helpandweb.com

Feel free to ask a trial version of TECMap and TECNativeMap.


TECMap license allows you to integrate it into your applications and distribute them freely without royalty, you can modify the source to suit your needs.


You may not redistribute the component, source or binary to produce a component from TECMap and distribute, including free.


The license to use the component TECMap is not a license to use the apis and map data used, they are respectively owned by Google, CloudMade, MapQuest Microsoft and OpenMapStreet.


Before use you must **read and accept** their terms of use

- [Terms of use apis Google Maps / Google Earth](#)
- [Terms of use of the API CloudMade](#)
- [Terms of use of MapQuest Nominatim](#)
- [Condition of use of MapQuest Elevation](#)
- [Terms of use of MapQuest map OpenStreet](#)
- [Terms of Use OpenMapStreet](#)

[MapQuest Open Elevation](#)  is used for altitude research with API [CloudMade](#) , [OpenMapQuest](#) and [Leaflet](#)

[MapQuest Nominatim](#)  is used to retrieve an address from latitude and longitude with [CloudMade](#) and [Leaflet](#).

[MapQuest Xapi API Service](#)  is used under CloudMade, OpenMapQuest and Leaflet the equity method of [Places](#) of Google.

Map [MapQuest](#) , maps [Mapnik](#), [Osmarender](#) and [CycleMap](#), Can be used with Google, CloudMade and Leaflet

Registration

TECMap uses a default key for Bing, OpenMapQuest, CloudMade and Leaflet, you **must obtain** your free key by registering on their site.

[Get a key for CloudMade / Leaflet](#)

[Get a key for OpenMapQuest](#)

Enter your key in the property **CloudMadeKey**, **MapQuestKey**, **BingKey** and **GoogleMapsKey** of TECMap.

The Google Maps JavaScript API v3 does not require an API key to function correctly

Google Maps API Premier

If you have subscribed to Google Maps API Premier you got a "Customer ID" style **gme-xxx**, you must fill in the **ClientID** property of TECMap to use it.

[contact google to find out how to subscribe to this service](#)

Install EMap

- Simply open the **EMap.dpk** file in the Source directory
- Right-click **EMap.bpl** in the project manager and click **install**
- Add the Source directory in the **list of libraries - Win32** (Menu Tools - Options).

You can install the component that will be on **ECMaps** tab of the Component palette

Install TECNativeMap

Repeat the procedure above but with the files **ECNativeMAP.dpk** (Vcl) and **FMX.ECNativeMAP.dpk** (Firemonkey)

You do not need to install ECNativeMAP.dpk with EMap because it is included in, however you must install FMX.ECNativeMAP.dpk

Activate Chromium

To use the Google Chromium browser instead of Internet Explorer you must first install the component [Delphi Chromium Embedded CEF1](#) developed by [Henri Gourvest](#).

ECMap also supports Chromium Embedded CEF3

Always take the latest versions present on SVN

Edit the **Delphi_Versions.inc** file included with ECMap and replace **{\$DEFINE CHROMIUM}** by **{\$DEFINE CHROMIUM}**

If you are using a version of Chromium CEF1 <= 275 you must replace **{\$DEFINE CEF275_UP}** by **{\$DEFINE CEF275_UP}**

To use CEF3 replace **{\$DEFINE CEF3}** par **{\$DEFINE CEF3}**

You can now install the ECMap component, use **TECMap.DisplayBrowser** property values **dbIE** or **dbChromium**

You will need to distribute binaries of chromium with your executable, they are in bin\win32.

In the .dpr of your project you can specify the path to the binary

```
program demomap;
```

```
uses
```

```
  ceflib,sysutils,
```

```
  Forms,
```

```
  UMainDemoMap in 'UMainDemoMap.pas' ;
```

```
  {$R *.RES} }
```

```
begin
```

```
CefLibrary := expandfilename('.\bin\win32\libcef.dll');
```

```
Application.Initialize;
```

```
Application.MainFormOnTaskbar := True;
```

```
Application.CreateForm(TFormDemoECMap, FormDemoECMap);
```

```
Application.Run;
```

end.

Here you will have the subdirectory bin\win32 at the same level as your executable

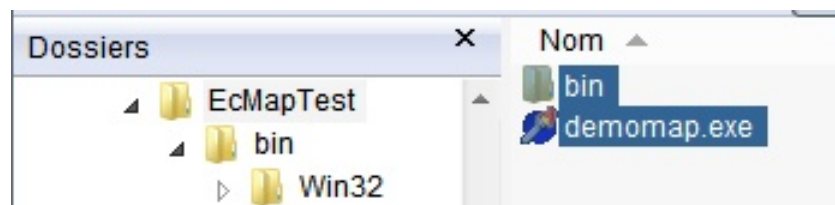


Fig. 4 Delivery of a project using chromium

CEF3 lets you use multiple processes, by default this is not the case, single process mode is recommended only for debugging not for delivery.

To enable multi-process mode you must add these 2 lines after `CefLibrary := ...` in your `.dpr`

```
CefSingleProcess := False;
```

```
if not CefLoadLibDefault then Exit;
```

Demonstrations

The component comes with 8 demonstration programs that show you how to exploit **DemoLocalise** you how to use the functions of [geolocation](#) , getting to a point from its address, find an address from its geographical position.

You will also see the use of a [InfoWindow](#) , the selection of the [API used](#) (or Google CloudMade) and the [type of map](#) displayed.

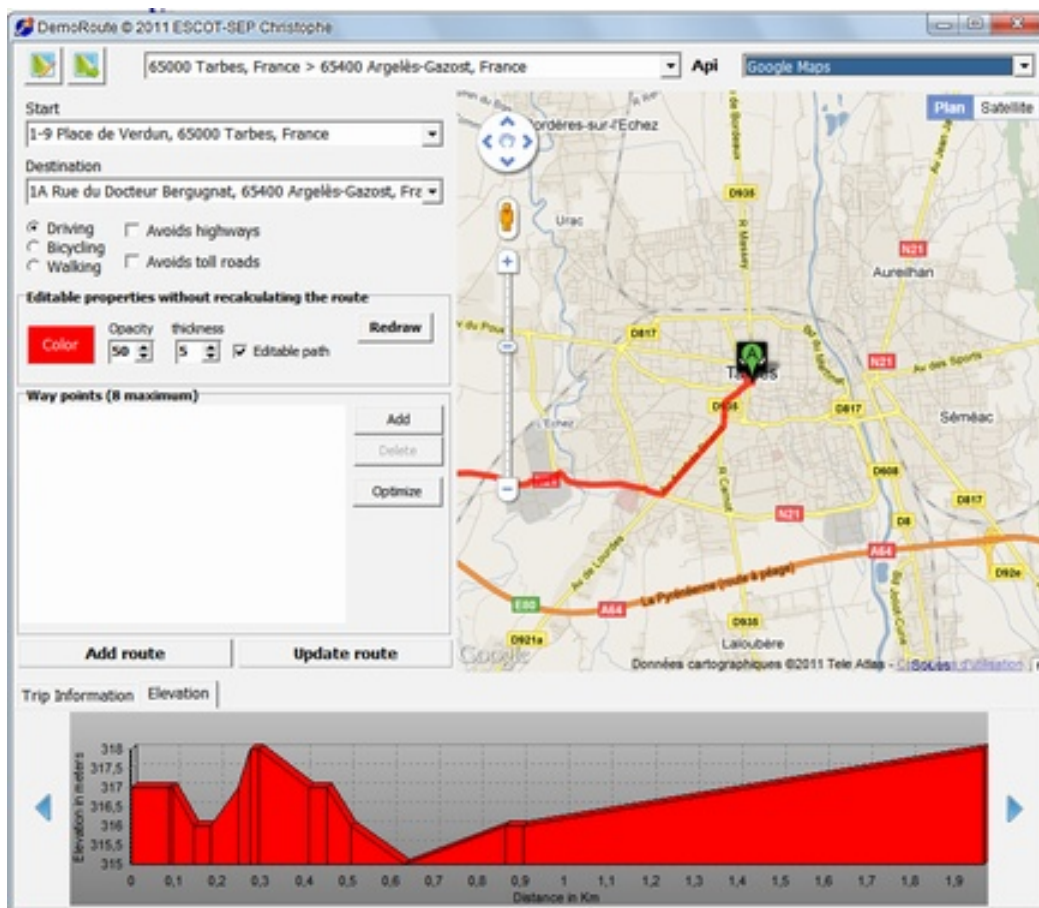


Fig. 5 DemoRoute

DemoRoute focuses on the management of [roads](#) , you will also see how to get [the altitude of a point set](#)

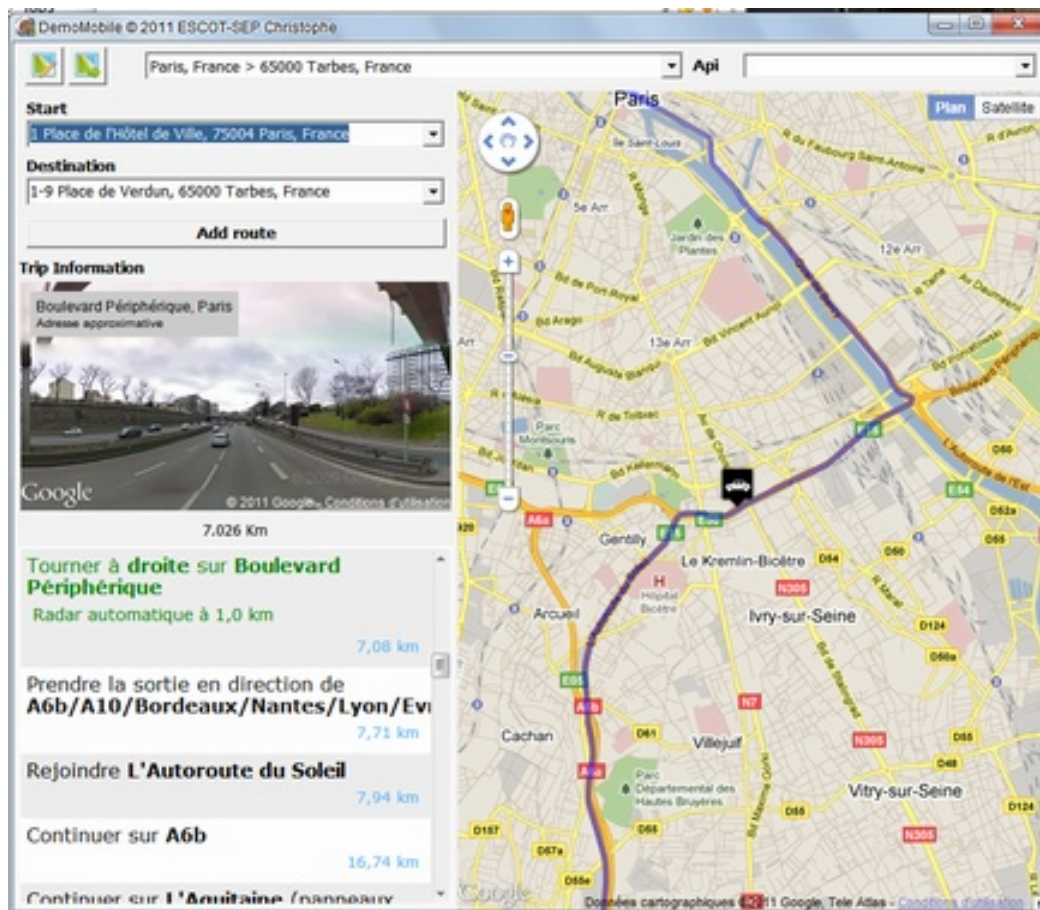


Fig. 6 DemoMobile

DEMOmobile for automatic management of movable on a track coupled to a monitoring of **StreetView**

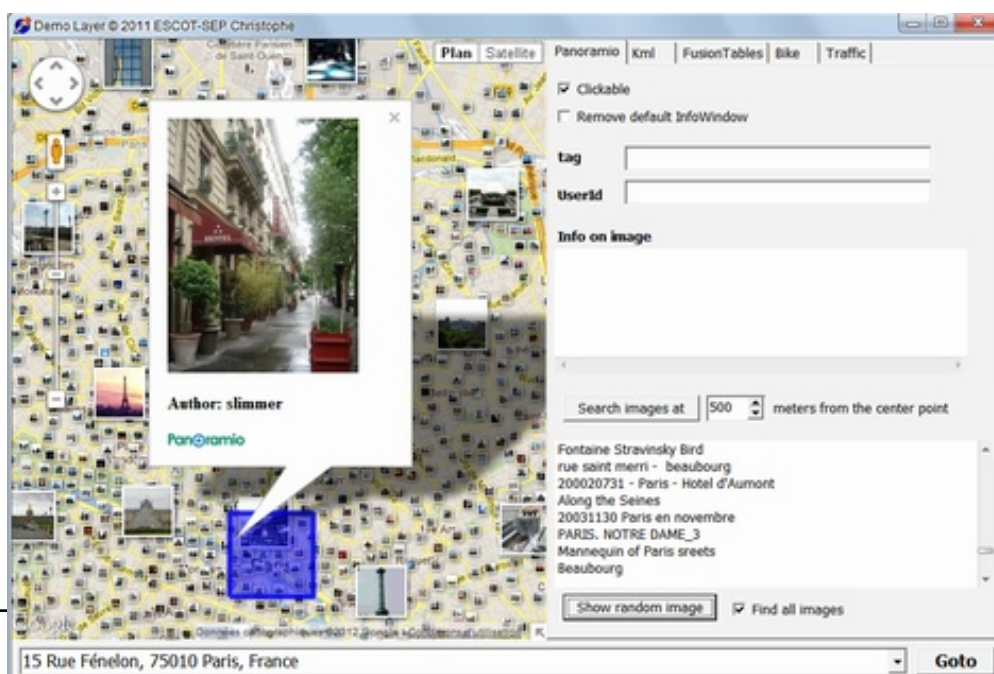


Demo3D uses [EarthView](#) , shows you how to display a [3D model](#) and make it follow a route



Fig. 8 DemoOverlays

DemoOverlays you discover the dynamic management of [overlays](#)
DemoLayer lets you see how to use [Panoramio](#) and other types of [layers](#)



DemoMatrix in Delphi is an adaptation of the [example javascript from google](#) and shows you how to use the [service DistanceMatrix](#)

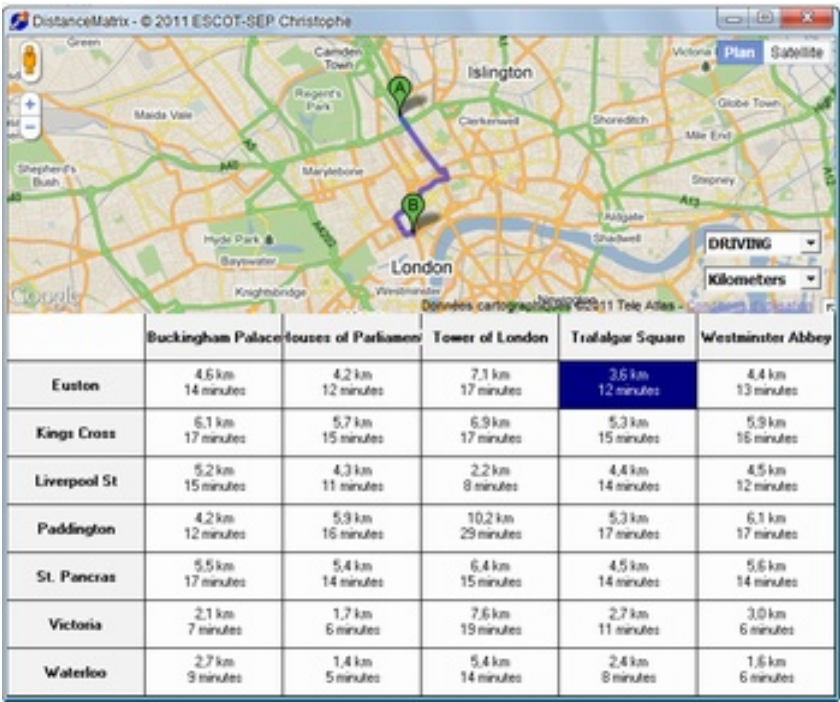


Fig. 10 Demonstrate the use of DistanceMatrix

DemoAdressEdit allows you to test the [TECMapAdressEdit](#) and [TECStaticMap](#) components



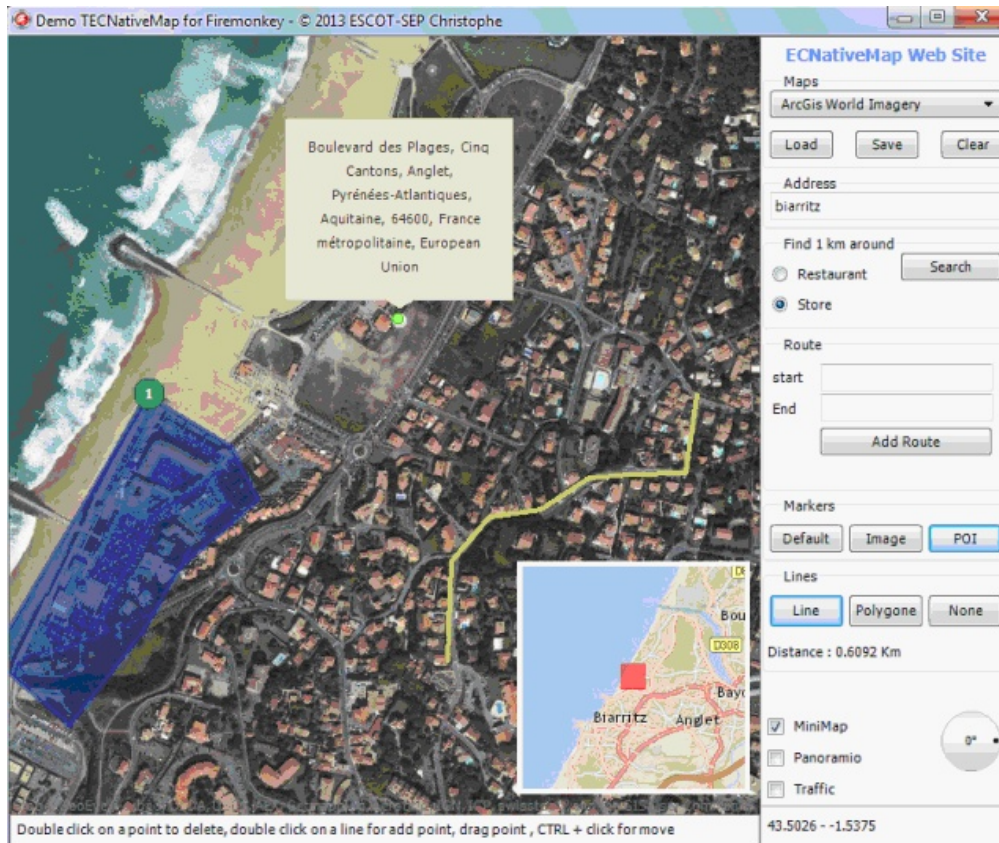


Fig. 12 Firemonkey Demo

Sources

The full source is supplied to the purchase of the license, but a test version is available for free and unconditional on [request](#).

The trial version is limited to the following points

- only Google maps is available
- This is an earlier uncorrected !

- No support for Chromium
- Google Earth is not available in full, no KML
- The component should be used only to test the product and distribution is not allowed
- An information box is open to launch

Trial TECNativeMap

Only the Bpl and dcu for Delphi Windows 32-bit VCL/FireMonkey

We will discover the methods, properties and events related to all that appeal to the geographical position of a point.

Placement on map

You can determine the coordinates of the center of the map through the **Latitude** and **Longitude** properties, they are **double** and **read / write**. You can directly edit the details through the procedure **setCenter(const dlatitude,dlongitude:double)**

```
// Delphi map component ECMap
var lat,lng:double;
begin
    // get center of map
    lat := map.Latitude;
    lng := map.Longitude;
    // move center of map
    map.setCenter(lat+0.001,lng);

end;
```

The procedure **PanTo(const dLatitude,dLongitude:double)** also offers the possibility to change the positioning.

The movement will be smoother if the move does not exceed half the height or width of the card.

Mouse position

The latitude and longitude of the point under the mouse cursor are returned by properties **MouseLatitude** and **MouseLongitude**. You can

tune into the event **OnMapMouseMove** to know your position in real time

Coping with change of position

When the center of the map is moved, either by code or directly to the mouse, the event **OnMapMove(sender: TObject;const dLatitude,dLongitude:double)** is triggered. **Sender** ECMap represents the component that has been modified, and **dLatitude** and **dLongitude** the new coordinates, which are also available through **Latitude** and **Longitude**.

When the card begins to move **OnMapDragStart** event is triggered, then **OnMapDrag** during displacement and **OnMapDragEnd** at the end .

You can also respond when moving the mouse by logging on **OnMapMouseMove**

Altitude

You get the altitude of the center of the map through the **Altitude** property, it is **double** and accessible *only by reading* .

Function

GetAltitudeAtLatLng(const dLatitude,dLongitude:double):double gives you the elevation of any point.

```
// Delphi map component ECMap
// altitude of map's center
map.Altitude;
// altitude at latitude 48, longitude 0.7
map.GetAltitudeAtLatLng(48,0.7);
```

Polylines have a special method for the [calculation of the altitude of all their points](#) .

Area displayed

You can determine the coordinates of the northeast (upper right) and point southwest (lower left) of the area displayed by the component properties through ECMap **NorthEastLatitude**, **NorthEastLongitude**, **SouthWestLatitude** **SouthWestLongitude** accessible *only by reading* .

The procedure

PanToBounds(const dLatlo,dLnglo,dLathi,dLnghi:double) you can change the view from the new coordinates of the low point (South West) and high (North East).

The procedure

fitBounds(const dLatlo,dLnglo,dLathi,dLnghi:double) allows you to adjust the view to the coordinates passed.

fitBounds works also with Google Earth

Function **ContainsLatLng(var dLatitude,dLongitude:double):boolean** tells you if the item in dLatitude, dLongitude is in the visible portion of the card.

As soon as the view changes **OnChangeMapBounds** event (**sender: TObject**) is triggered.

The **ScreenShot** property returns a TBitmap containing the map image

```
// Delphi map component TECNativeMap

// save map to bmp file
map.Screenshot.SaveToFile('c:\mymap.bmp');
```

Zoom

The **Zoom** property allows you to control the definition of your card, it is of type integer and is **read / write**

*Using a Google map type you have access also to the property and **maxZoom minZoom** that allow you to determine the limits of the zoom*

The change triggers the zoom event

OnChangeMapZoom(sender: TObject)

Distance

Function

DistanceFrom(const dLatitudeStart, dLongitudeStart, dLatitudeEnd, dLongitudeEnd)

Angle with respect to North

Function

HeadingFrom(const dLatitudeStart, dLongitudeStart, dLatitudeEnd, dLongitudeEnd)
 from point **dLatitudeStart, dLongitudeStart** to point **dLatitudeEnd, dLongitudeEnd**

We will discover the methods, properties and events related to all that appeal to the geographical position of a point.

Placement on map

You can determine the coordinates of the center of the map through the **Latitude** and **Longitude** properties, they are **double** and **read / write**. You can directly edit the details through the procedure **setCenter(const dlatitude,dlongitude:double)**

```
program demomap;

uses

  ceflib,sysutils,
  Forms,
  UMainDemoMap in 'UMainDemoMap.pas' ;

{$R *.RES}

begin

  CefLibrary := expandfilename('..\bin\win32\libcef.dll');

  Application.Initialize;
  Application.MainFormOnTaskbar := True;
  Application.CreateForm(TFormDemoECMap, FormDemoECMap);
  Application.Run;

end.
```

The procedure **PanTo(const dLatitude,dLongitude:double)** also offers the possibility to change the positioning.

The movement will be smoother if the move does not exceed half the height or width of the card.

Mouse position

The latitude and longitude of the point under the mouse cursor are returned by properties **MouseLatitude** and **MouseLongitude**. You can tune into the event **OnMapMouseMove** to know your position in real time.

Coping with change of position

When the center of the map is moved, either by code or directly to the mouse, the event **OnMapMove(sender: TObject;const dLatitude,dLongitude:double)** is triggered. **Sender** EMap represents the component that has been modified, and **dLatitude** **dLongitude** the new coordinates, which are also available through **Latitude** and **Longitude**.

When the card begins to move **OnMapDragStart** event is triggered, then **OnMapDrag** during displacement and **OnMapDragEnd** at the end .

You can also respond when moving the mouse by logging on **OnMapMouseMove**

Altitude

You get the altitude of the center of the map through the **Altitude** property, it is **double** and accessible *only by reading* .

Function

GetAltitudeAtLatLng(const dLatitude,dLongitude:double):double gives you the elevation of any point.

```
program demomap;  
  
uses  
  
  ceflib,sysutils,  
  Forms,  
  UMainDemoMap in 'UMainDemoMap.pas' ;
```

```

{$R *.RES
}

begin

  CefLibrary := expandfilename('.\bin\win32\libcef.dll');

  Application.Initialize;
  Application.MainFormOnTaskbar := True;
  Application.CreateForm(TFormDemoEMap, FormDemoEMap);
  Application.Run;

end.

```

Polylines have a special method for the [calculation of the altitude of all their points](#) .

Area displayed

You can determine the coordinates of the northeast (upper right) and point southwest (lower left) of the area displayed by the component properties through EMap **NorthEastLatitude, NorthEastLongitude, SouthWestLatitude SouthWestLongitude** accessible *only by reading* .

The procedure

PanToBounds(const dLatlo,dLnglo,dLathi,dLnghi:double) you can change the view from the new coordinates of the low point (South West) and high (North East).

The procedure

fitBounds(const dLatlo,dLnglo,dLathi,dLnghi:double) allows you to adjust the view to the coordinates passed.

fitBounds works also with Google Earth

Function `ContainsLatLng(var dLatitude,dLongitude:double):boolean`

tells you if the item in dLatitude, dLongitude is in the visible portion of the card.

As soon as the view changes **OnChangeMapBounds** event (**sender: TObject**) is triggered.

The **ScreenShot** property returns a TBitmap containing the map image

program demomap;

uses

ceflib,sysutils,

Forms,

UMainDemoMap in 'UMainDemoMap.pas' ;

{ \$R *.RES }

begin

CefLibrary := expandfilename('.\bin\win32\libcef.dll');

Application.Initialize;

Application.MainFormOnTaskbar := True;

Application.CreateForm(TFormDemoECMap, FormDemoECMap);

Application.Run;

end.

Zoom

The **Zoom** property allows you to control the definition of your card, it is of type integer and is **read / write**

*Using a Google map type you have access also to the property and **maxZoom minZoom** that allow you to determine the limits of the zoom*

The change triggers the zoom event

OnChangeMapZoom(sender: TObject)

Url

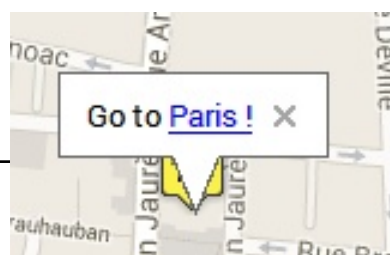
The **Url** property returns / accepts a string in the format '**#zoom/Latitude/Longitude**'

```
// zoom 18 latitude 48.856527 longitude 2.352104
// welcome to Paris !
map.Url := '#18/48.856527/2.352104';
```

Assigning a value to this property raises the event **OnBeforeUrl**(sender : TObject; var Url:string) that can allow you to change the url, and even to cancel the change by returning an empty string.

You can pass a such link in a [InfoWindow](#)

```
// welcome to Paris !
map.Shapes.InfoWindows[0].Content := 'Go to <a
href="#18/48.856527/2.352104">Paris !</a>';
```



If you pass a 'classical' url it will open in your default browser.

Distance

Function

DistanceFrom(*isClassical*, *dLatitudeStart*, *dLongitudeStart*, *dLatitudeEnd*, *dLongitudeEnd*)

Angle with respect to North

Function

HeadingFrom(*isClassical*, *dLatitudeStart*, *dLongitudeStart*, *dLatitudeEnd*, *dLongitudeEnd*)
Returns the angle in degrees from point *dLatitudeStart*, *dLongitudeStart* to point *dLatitudeEnd*, *dLongitudeEnd*

API

TECMap allows you to use either [the Google Maps API](#) , [the API CloudMade](#) , [the API Leaflet](#) and [the API OpenMapQuest](#) to display your cards, you make the switch by setting the property **Mapapi** to **apiGoogle** or **apiCloudMade** or **apiLeaflet** or **apiOpenMapQuest**

The **SSL** property lets you use a secure connection for the moment only the Google API provides this functionality.

ApiVersion property of type string allows you to specify the version of the API that you wish to use, it only works for Google, if you do not tell is the latest version to be used.

```
// Delphi map component EMap

// find the distance in km by road between Tarbes and
// Lourdes via Aureilhan
dKm := map.DistanceRouteByAdress('Tarbes',
'Aureilhan|Lourdes');

Adresses := TStringList.create;
try

    Adresses.add('Tarbes');
    Adresses.add('Aureilhan');
    Adresses.add('Séméac');
    Adresses.add('Lourdes');

    // create route Tarbes/Lourdes via Aureilhan and Séméac
    map.AddRouteByAdress(Adresses);

finally
    Adresses.free;
end;
```

This can be useful in case of bug in some versions.

The API change triggers events **OnBeforeChangeMapApi** and **OnAfterChangeMapApi**.

When the card is ready **OnLoad** event is fired, you can also tune into **OnBeforeLoad** if you want to change the connection url to the API and add code to connect javascript libraries

You can recharge a card and its contents by calling the **Reload**, useful to force a release of memory, and **OnAfterReload** **OnBeforeReload** events are available.

You do not have to worry about backing up the card, it will be fully recharged, excluding the results of [a search places](#)

Change properties Mapapi, apiVersion and SSL triggers a reload.

[StreetView](#) with memory leaks, until google fixes them, reload is call automatically every 1000 views.

TECMap will hold closer to the apis, please read their documentation.

In addition to the [license that I invite you to read](#) , the main differences are the lack of SSL, view satellite, [3D view](#) of [Panoramio](#) , the [DistanceMatrix](#) and [StreetView](#) in the API CloudMade.

The differences do not cause error in the component TECMap, using an unsupported property has simply no effect, it is possible that future developments following apis deficiencies are corrected.

You can change the API at any time without loss of data, excluding the results of a search for places.

Card Type

OpenMapQuest supports classic map, view satellite and hybrid views.



g. 14 OpenMapQuest

CloudMade and Leaflet support classic map.



You have also maps OpenMapStreet, Mapnik, Osmarender, CycleMap and (classic, satellite and hybrid)



Fig. 16 Carte OpenMapStreet Mapnik





g. 18 OpenMapStreet CycleMap



With the Google API you have the classical map, the satellite view with or without captions, a vie



g. 20 Google : Type Roadmap





g. 22 Google : Type TERRAIN

You change your view through the property

MapTypeId as possible

mtROADMAP

mtSATELLITE

mtHYBRID,

mtTERRAIN, mtMAPNIK, mtOSMARENDER, mtMapQuest and mtCYCLEMAP

For satellite images (mtSATELLITE, mtHYBRID) Google Map begins to develop at 45 ° views, you

Tilt

This is not implemented everywhere, [google](https://www.google.com/maps/)

When changing the type of card it is by code or directly on the map in detail, the event

OnChangeMapTypeId (Sender: TObject)

~~Earth View~~ Google API you have also seen

Styles

The **Styles** property allows you to reset the display of the map.

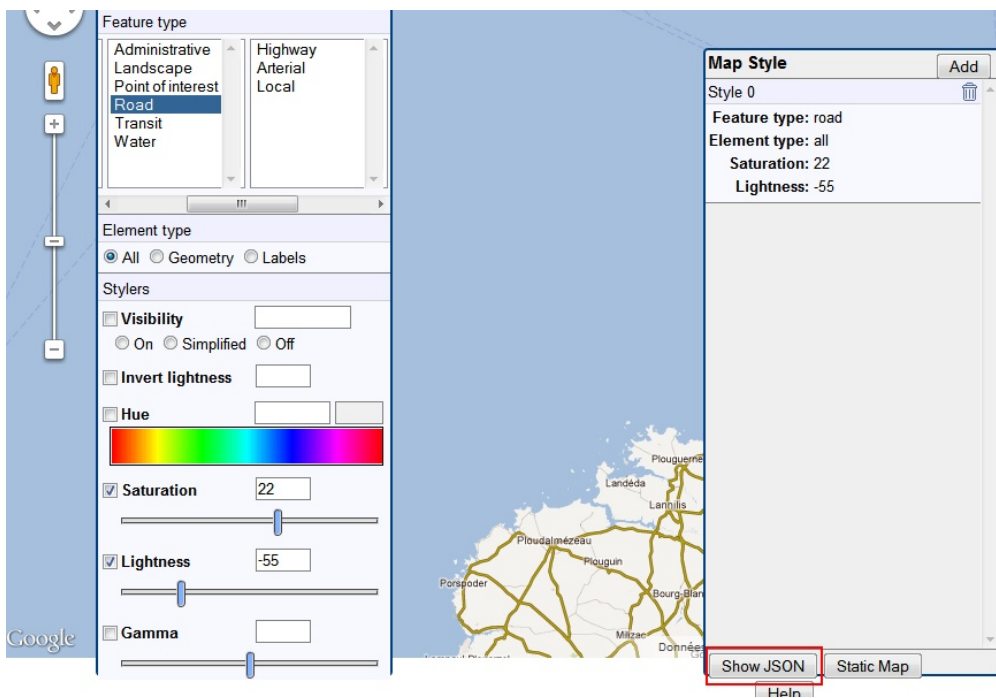
Visit [this link](#) for more information on the syntax of the styles.

Google puts at your disposal

You simply assign to

map.Styles

JSON.



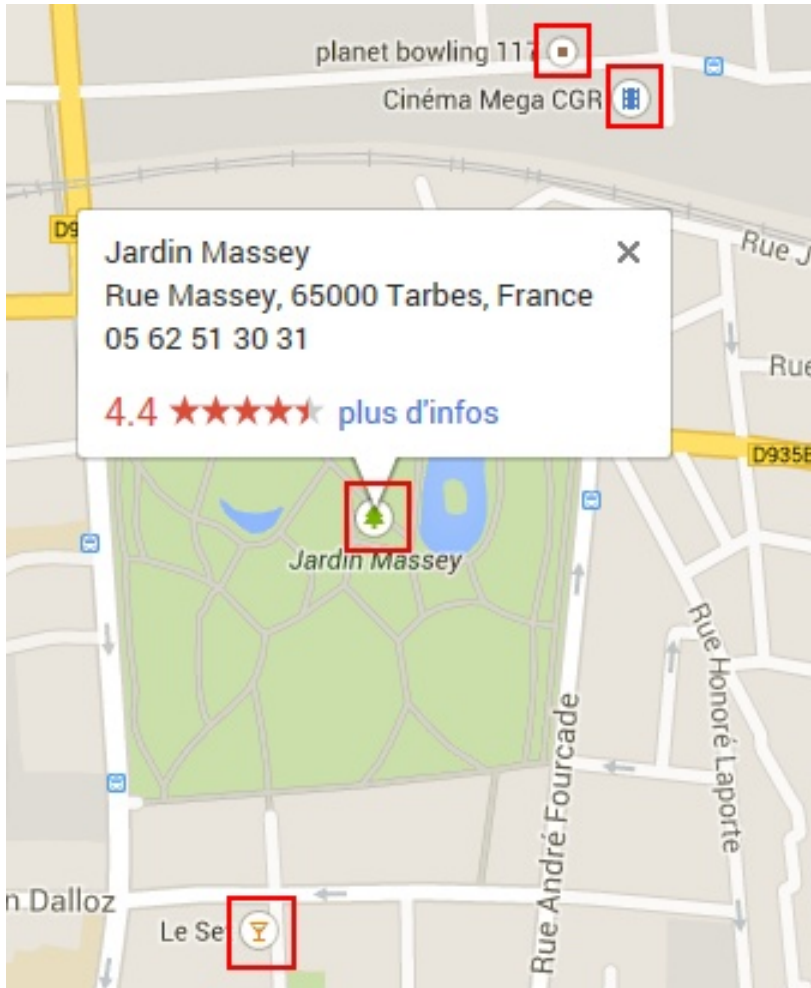
g. 23 Get the styles to the JSON format

Google Maps API v3 Styled Maps JSON

```
[
  {
    featureType: "road",
    stylers: [
      { saturation: 22 },
      { lightness: -55 }
    ]
  }
]
```

Remove Points of interest Google Maps

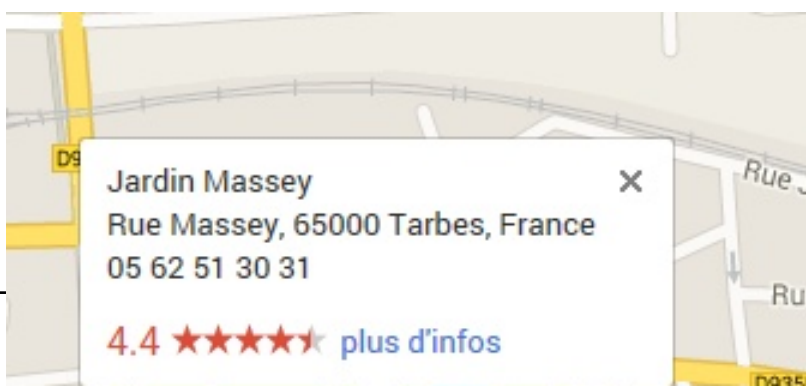
You can use styles to remove all or some poi on your card



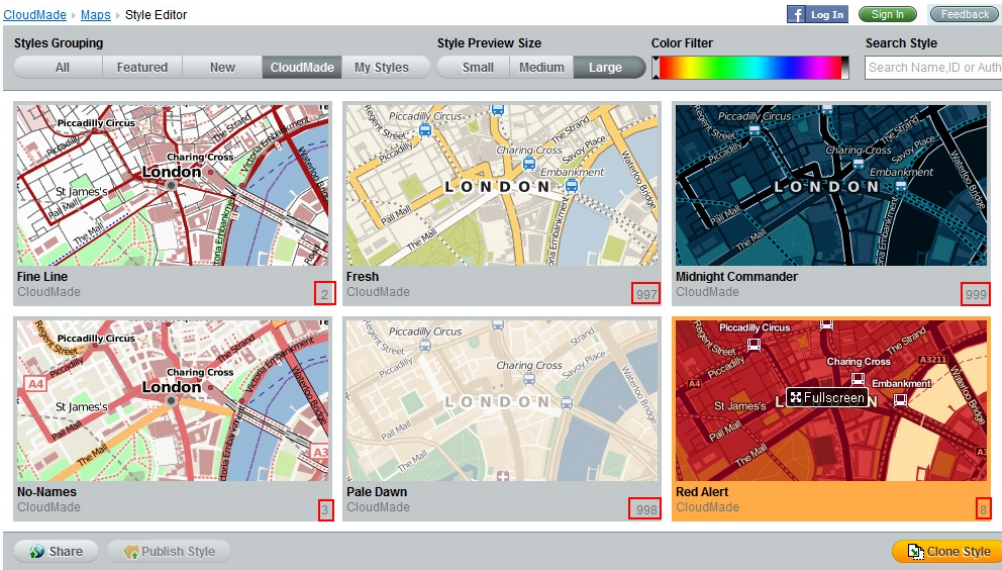
g. 25 POI

This example shows you how to delete all poi except the poi.park

```
map.styles = [
  {featureType: "poi",elementType: "all",stylers:[{visibility: "off" } ]},
  {featureType: "poi.park",elementType: "all",stylers:[{visibility: "on" }, ] } ]'
```



Voir la documentation de Google pour toutes les valeurs featureType
Under CloudMade the
Style library contains an identifier, you can go visit



g. 27 CloudMade style editor

2
3
8
997
998
999
How to use styles, example of use
map.Styles:=
'999'

OpenMapQuest does not support styles

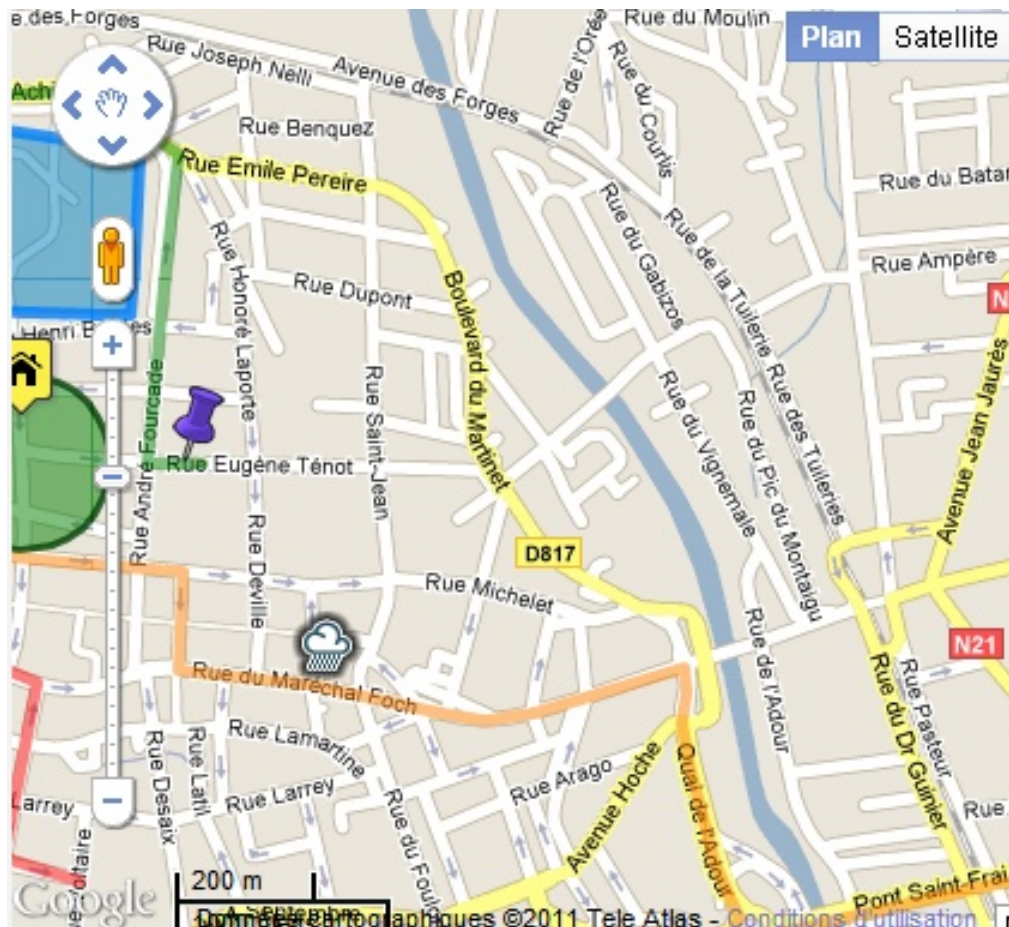


Fig. 28 Google maps controls

With maps CloudMade and OpenMapQuest only zoom control and

You can suppress the display of all controls by flipping the
EnabledUIControl
 property to **false**

```
// Delphi map component EMap
// Hide UI controls
map.EnabledUIControl := false;
// Hide navigation bar,
// NavigationControl
// MapTypeControl
// ZoomControl
// ScaleControl
```

NavigationPosition, MapTypeControlPosition, ZoomPosition, ScalePosition following values:

- cpTopLeft
- cpTopCenter
- cpTopRight,
- cpRightTop
- cpRightCenter
- cpRightBottom
- cpBottomRight
- cpBottomCenter
- cpBottomLeft
- cpLeftBottom
- cpLeftCenter
- cpLeftTop

Under CloudMade only positions

cpTopLeft
 cpTopRight
 cpBottomLeft
 cpBottomRight

Menu

TECMap property has a **menu** property that allows you to associate a **TPopupMenu** when you right click on the map.

The event is still triggered **OnMapRightClick**ed, it comes just after the opening of TPopupMenu

Property **Address** gives the address of the card, it is of type **string** and is **read / write**

Address format

To find an address you can use a free format such as "address, city, ZIP".

CloudMade and OpenMapQuest use [Nominatim Search Service](#)

Limitation of geolocation

Converting coordinates / address opens a connection to the server of your card provider, Google is setting up a quota system to avoid too much demand in the shortest time. If you exceed this limit event **OnJavaScriptError** (**Object;const sError:string**) is raised with sError = 'OVER_QUERY_LIMIT'

ECMap implements a caching system which means that as the coordinates do not change the connection on the server takes place only for the initial consultation, so you can perform multiple read and not worry about the quota.

The system cache is valid for all objects for their geolocation, such as Markers

Geolocation

To get the address of a specific point you have the function **GetAddressFromLatLng(dLatitude,dLongitude:double):string**; You can get the coordinates of an address with the function **GetLatLngFromAddress(const sAdress:string;var dLatitude,dLongitude:double):boolean**;

By assigning a value to the property **address**, you'll change the position of the center of your card to match it to the address

This will trigger the event **OnAddress(sender: TObject;const sAddresses:string;var Index:integer)**

GetLatLngFromAddress *also triggers* **OnAdress**

sAddresses contain all addresses, separated by #13#10 may correspond with your request.

Index contain the index of the selected address, default 0, you can change it in this event.

Output from this event the center of the map corresponds with the address selected

Property **Adresses** type **TGeoCoderReponses** contains all the answers returned by the server.

Example of use

```
// Delphi map component EMap

// calcul altitude for all point of polyline 0
// you can pass nil if you don't show a progressbar
map.Shapes.Lines[0].GetAltitudes(doGetAltitude);
...
```

```

    { *
      event fired by getAltitudes

      @param Sender TECShapeLine
      @param Total number of altitude's point calculated
      @cancel flag for abort calcul
    }
  procedure
    TFDemoRoute.doOnGetAltitude(Sender: TECShapeLine;const
    Total:integer;var cancel:boolean);
  begin
    ProgressAltitude.Position := total;
    // cancel if press button
    cancel                      := btAbortAlt.tag = -1;
  end;

```

Geolocation asynchronous

GetAdressFromLatLng, GetLatLngFromAdress et Adress := 'paris' await the results of the query, you have at your disposal two procedure asynchronous (nonblocking) equivalent.

```

procedure GeoLocationFromLatLng
(const dLatitude,dLongitude:double);

```

*under CloudMade this procedure is blocking but
not GeoLocation*

```

procedure GeoLocation(const sAdress:string);

```

Both procedures trigger event **OnGeoLocation** (sender:Tobject;const dLatitude,dLongitude:double;const sAdresse:string);

dLatitude,dLongitude et **sAdresse** correspond to the first response available like for functions synchronous **Addresses** contains all the results.

You can use the component TComboBox to contain both the query and the result thereof, the demo shows you how DemoLocalise

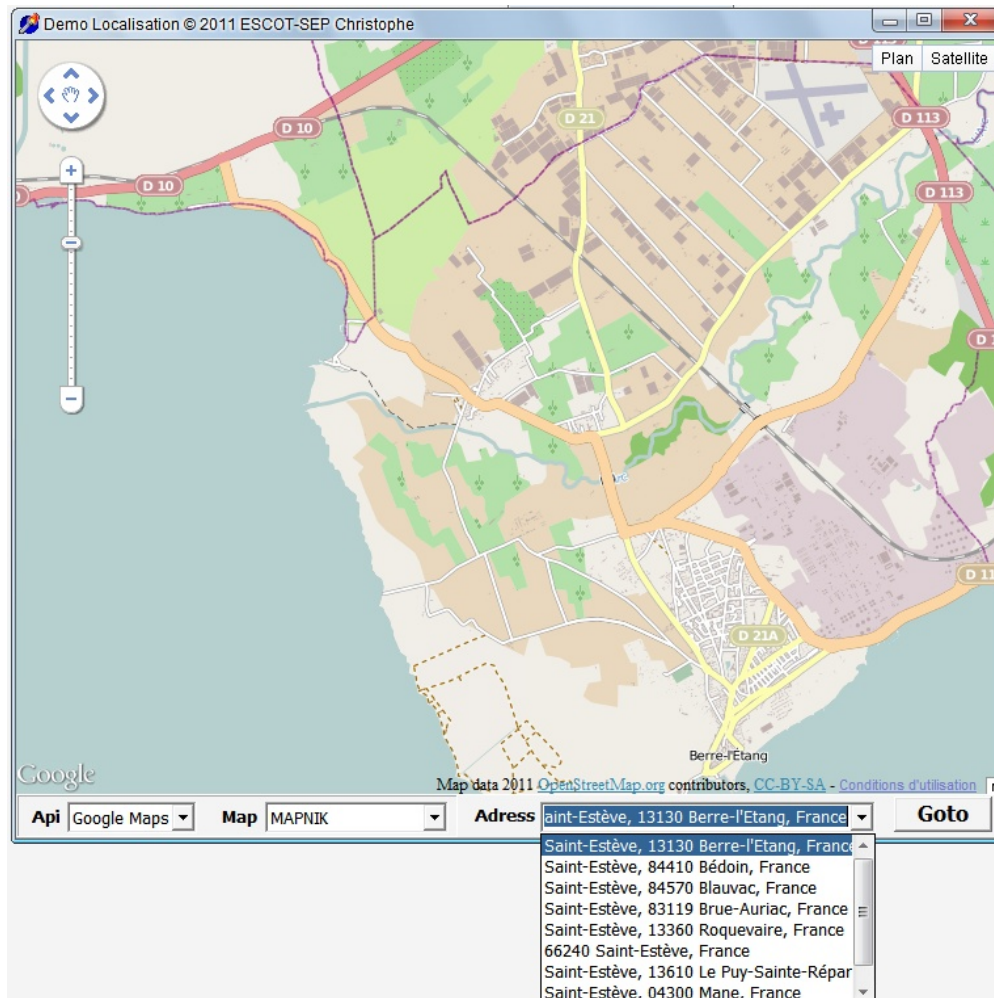


Fig. 29 Using a TComboBox to manage the geolocation

PropertiesTGeoCoderReponses

property Addresses:string

The list of addresses separated by any #13#10

property Adresse[index:integer] : string

The address number Index

property Error [index:integer] : string

The possible error

property Latitude[index:integer] : double

Latitude corresponding to the address Index

property Longitude[index:integer] : double

Longitude corresponding to the address Index

property LocationType[index:integer]: string

The accuracy of the answer (only for google)

History answers

Each geolocation will change the contents of the property **Addresses**, you can store it in a list spécialisée, accessible from the property **ListAdresses** type **TListGeoCoderReponses**, for future use, is the right time in the event **OnAdress**

PropertiesTListGeoCoderReponses

function Add(const geoCode:TGeoCoderReponses):integer;

Adds value in the list TGeoCoderReponses

function count:integer;

Number of stored TGeoCoderReponses

procedure clear;

Clears all values

procedure Delete(const index:integer);

Removes the index value index

property GeoCoderReponses[index:integer]:TGeoCoderReponses;

Read / write access to values by their index

Using ListAddresses

The typical usage of this list is where you must manage a starting address and arrival, demos and DEMOMobile DemoRoutes shows you a way to.

A ComboBox is used for departure and arrival, during validation of an address by **RETURN** Geolocation is request.

In the event **OnAdress** is stored in the ComboBox addresses in clear, placed in **ListAddresses** full data returned by the query, and stores the index into the tag property of the combobox.

We can then obtain the coordinates of any address placed in the ComboBox.

```
// Delphi map component EMap
var lat,lng:double;
begin
  // get center of map
  lat := map.Latitude;
```

```
lng := map.Longitude;  
    // move center of map  
map.setCenter(lat+0.001,lng);  
  
end;
```

Use the Open Map Quest services with Google maps

By switching the property **UseOpenMapQuestServices** to true you use OpenMapQuest for all that concerns the geolocation, the altitude and [places](#)

Property **Places** type **TECPlaces** lets you search specific locations in a given area, such as finding restaurants in a radius of 500 meters.

With the Google API you'll use the [service Places](#) with other you use [xapi MapQuest API Service](#) that uses data of [OpenStreetMap](#), this causes a small difference in the syntax of the research we detail thereafter.

You can force the use of Xapi under Google by setting the property **UseOpenMapQuestServices**

The **XapiServer** property allows you to use another server Xapi than MapQuest

```
// Delphi map component EMap
// use overpass-api.de

map.XapiServer := 'http://www.overpass-api.de/api/xapi?';
```

Google Places is limited to 20 results per query

TECPlaces

procedure **Search**(Tags:string);

Launch a search, tag contains the query, the syntax varies depending on the Google API or CloudMade

[List of tags available in Google](#)

[Documentation for xapi CloudMade](#)

```
// Delphi map component EMap
```

```

var Tags:string;
begin
  map.Places.Latitude := map.latitude;
  map.Places.Longitude:= map.Longitude;
  // syntaxe change with api
  case map.MapAPI of
    apigoogle : begin
      if ckStore.checked then
        Tags := 'store'
      else
        if ckRestaurant.checked then
          Tags := 'restaurant'
        else
          if ckDoctor.checked then
            Tags := 'doctor';
          end;
        end;

      else begin

        if ckStore.checked then
          Tags := 'node[shop=*]'
        else
          if ckRestaurant.checked then
            Tags := 'node[amenity=restaurant]'
          else
            if ckDoctor.checked then
              Tags := 'node[amenity=doctors]';
            end;
          end;
        end;

        // 500 meters
        map.Places.Radius := 500;
        // run search
        map.Places.Search(Tags);

```

When the search is completed, the event **OnPlacesSearch** is triggered

Each launch of Search erases the results of a previous search

procedure **TextSearch**(const TextQuery: string);

Launch of a textual search, only available with the api language Google search

When the search is complete, the event **OnPlacesSearch** is raised

property **AutoComplete** : boolean

Displays a field of research auto complete, only available with api Google

When a selection is complete, the **OnPlaceAutoComplete** event is raised

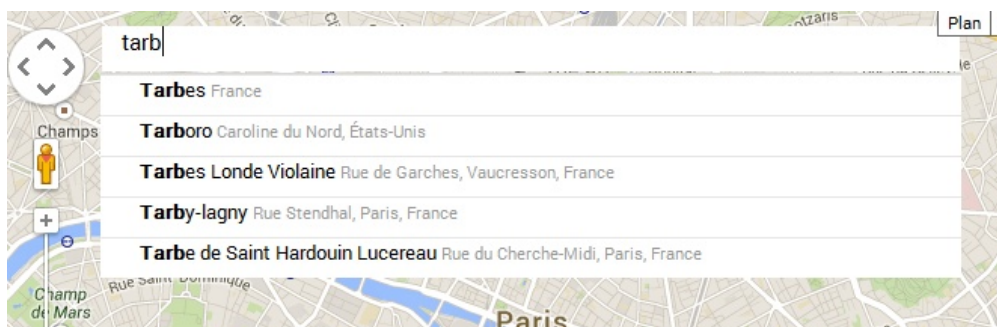


Fig. 30 Place AutoComplete

If you want to adjust the size of the search box to the width of the card, connect you on the **OnResize** event and add this line

```
map.Javascript('input_autocomplete.style.width="'+inttostr(map.Width-160)+'px'
```

property **Adress** : string read FAdress write FAdress;

Property **read / write** indicates that the focus of the search area, it takes precedence over the properties **Latitude** and **Longitude**

property **Status** : string;

Read-only property that returns a string indicating the status of research, 'OK' if all went well

The status is available in the event **OnPlacesSearch**

property **ItemDetail** : integer;

Read-only property that contains the index of the result which we seek further details

property **Latitude** : double;

Property **read / write** that indicates the latitude of the midpoint of the search area, this invalidates the contents of the Property **Address**

property **Longitude**: double;

Property **read / write** that indicates the longitude of the midpoint of the search area, this invalidates the contents of the Property **Address**

property Radius : integer;

Property **read / write** indicating the search radius **in meters**

property **useOSM** : boolean;

Use data from OpenStreetMap with the api Google

property **maxResult** : integer;

limits the number of results for a search in OpenStreetMap, ignored with **Google Places**

property **Searching**: boolean;

Read-only property that indicates whether a search is underway

property **Results**:TECPlaceResults;

List of results, the event **OnPlacesSearch** is triggered when results are available

TECPlacesResults

This class manages the list of results returned by **Search**

procedure **Clear**;

Erases all results

function **Count**:integer;

Returns the number of results matching the query

procedure **Delete**(const index:integer);

Clears the results of which we pass the index

property **Result**[index:integer]:TECPlaceResult

Table that allows access to the results, it is the default property so you can access it directly by **map.Places.Result** [index] instead of **map.Places.Results.Result** [index]

TECPlaceResult

Managing a class match for a search

procedure **getDetails**;

Performs a query on the result for further details

 *Not available CloudMade*

When details are available the event **OnPlacesDetail** is triggered .

property **Result**[const key:string]:string;

Returns the key value which is passed as parameter

property **Detail**[const Key:string]:string;

Returns the key value which is passed as a parameter for details

property **NameResult**[const index:integer]:string;

Returns the key of a result in terms of its index

property **NameDetail**[const index:integer]:string;

Returns the key of a detail according to its index

property **CountResult**:integer;

Returns the number of elements (key = value) of a result

property **CountDetail**:integer read getCountDetail;

Returns the number of elements (key = value) of a detail

property **Latitude** : double;

Latitude result

property **Longitude**: double;

Longitude result

property **RawResult** : string;

Returns all the elements of the result as a string consisting of **key = value** lines

property **RawDetail** : string;

Returns all the elements of detail as a string consisting of **key = value** lines

```
// Delphi map component EMap

// Event OnPlacesSearch
procedure TFDemoLocalise.mapPlacesSearch(Sender: TObject);
var i:integer;
    iMarker : integer;
    s,icon,types,names: string;
begin

    if map.Places.Status<>'OK' then exit;

    for i:=0 to map.Places.Results.count-1 do
    begin

        types:= map.Places.Results[i].result['types'];
        names:= map.Places.Results[i].result['name'];

        // add a marker for all valid result
        if (names<>'') then
        begin

            iMarker

:= map.AddMarker(map.Places.Results[i].latitude,map.Places.Results[i].longitu

            if iMarker>-1 then
            begin
                // select icon for types
                if pos('restaurant',types)>0 then
                    icon :=
'http://google-maps-icons.googlecode.com/files/restaurant.png'
                else
                    if pos('doctor',types)>0 then
                        icon :=
'http://google-maps-icons.googlecode.com/files/doctor.png'
                    else
                        icon :=
'http://google-maps-icons.googlecode.com/files/supermarket.png'

                map.Markers[iMarker].icon := icon;
```

```

        map.Markers[iMarker].tag      := FStartPlace+i;
        map.Markers[iMarker].infoWindow
:= map.InfoWindows.Add(names);
        map.InfoWindows[map.Markers[iMarker].infoWindow].Anchor
:= iMarker;

    end;

end;

end;

end;

```

Demonstration

the program **DemoLocalise** shows you how to manage **Places**

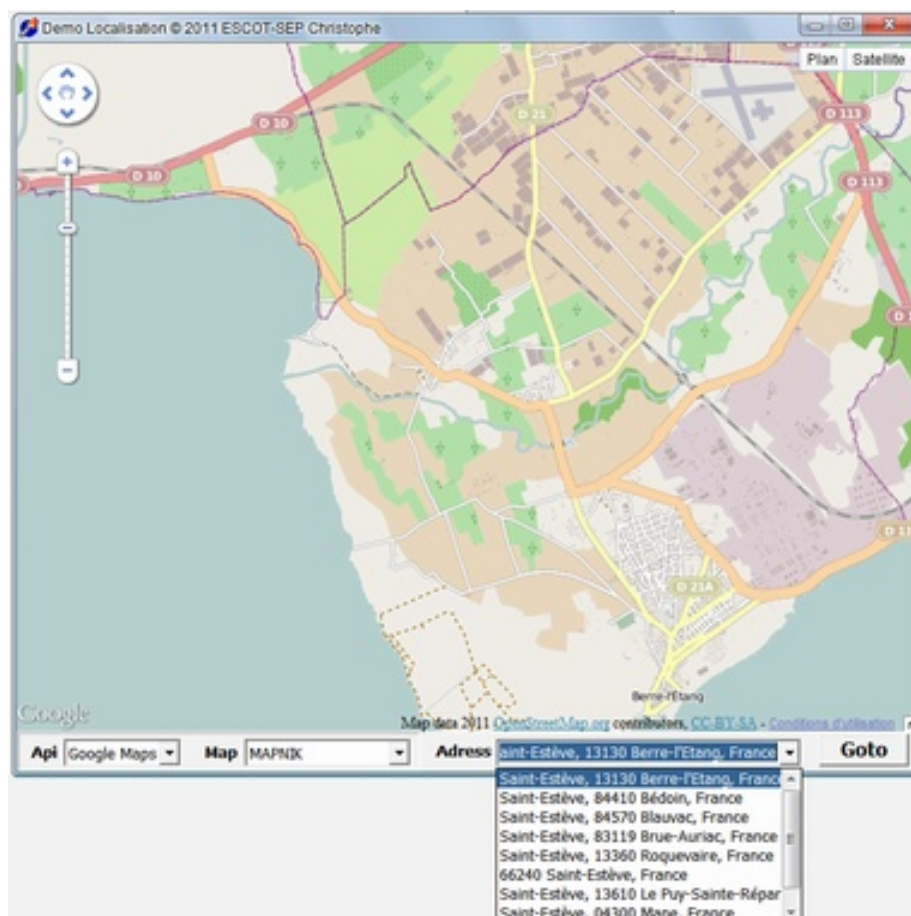


Fig. 31 DemoLocalise

TECMap can save and reload all its data in a simple text file, it includes not only the component's settings and views but also map data [overlays](#)

Alternatively, you can import/export your data in [GPX](#), [KML](#) and [GeoJSON](#) formats but only geographical data are exploited.

Sauvegarde/Restauration

function **SaveToFile**(const filename:string):boolean;

Saves the map to a text file.

Use the .gpx , .kml and .json to specify a format, otherwise it is the internal format of ECMap that will be used

function **LoadFromFile**(const filename:string):boolean;

Load the map with a text file

Use the .gpx , .kml and .json to specify a format, otherwise it is the internal format of ECMap that will be used

LoadFromFile can download files on internet

property **toGPX** : string;

Property **read/write** which gives access to the data of the card in GPX format.

This property is used by **SaveToFile**, **LoadFromFile**.

property **toTxt** : string;

Property **read/write** which gives access to the data of the card in a text format.

This property is used by **SaveToFile**, **LoadFromFile**.

property **ToKml** : string;

Property in **read / write** which returns the [overlays](#) (all except the [Labels](#)) in [Kml format](#)

[EarthView](#) allows import/export KML in a larger format

property **ToGeoJSON** : string;

Property in **read / write** which returns / import [overlays](#) (just the markers, the polylines, polygons, circles and rectangles) to [GeoJSON format](#)

The affection of a value to **toTxt**, **ToKml**, **ToGpx**, **ToGeoJSON** or the loading of a file with **LoadFromFile** causes events

OnBeforeChangeToTxt and **OnAfterChangeToTxt**.

OnLoadOverlay is also raised after the addition of each overlay (marker, polyline, polygon,...) that may allow you to adjust properties while loading data.

You can stop the load by switching **StopLoadOverlay** to true

```
// Delphi map component EMap
//
procedure TFormDemoEMap.mapLoadOverlay(sender: TObject;
const Index: Integer; const OverlayType: TOverlayType);
begin
    case OverlayType of
        ovMarker : begin
            // override data marker
            map.markers[Index].name :=
'your data';
        end;
```



```

        ovLine    : begin
            // override data Polyline
            map.Polylines[index].Color := clBlue;
            map.Polylines[index].Update;
        end;

        ...

    end;

    // stop if too many markers
    map.StopLoadOverlay := map.Markers.Count>3000;
end;

```

Backup / Restore

function **SaveToFile**(const filename:string):boolean;

Saves the map as a text file

function **LoadFromFile**(const filename:string):boolean;

Load the card with a text file

function **SaveToKMLFile**(const filename:string):boolean;

Saves overlays to the [kml format](#)

property **toTxt** : string;

Property **read / write** which provides access to map data in text format.

This property is used by **SaveToFile**, **LoadFromFile**.

property **ToKml** : string;

Read-only property that returns [overlays](#) (all but [Labels](#)) in [kml format](#)

This property is used by **SaveToKmlFile**

[EarthView](#) allows [import / export KML](#)

property **ToGeoJSON** : string;

Property in **read / write** which provides access to [overlays](#) (just the markers, polylines,

polygons, circles and rectangles) in [GeoJSON format](#)

The affection of a value to `toTxt` causes and events **OnBeforeChangeToTxt** and **OnAfterChangeToTxt** the first place just before the change in fair value and the second after.

Setup import / export

When you import / export in text format, by default all the data are, whether the card settings (such as API, position etc.) or overlays ([markers](#) , circles, [roads](#) etc.).

Just as when importing the old overlays are replaced by new.

Property **ToTxtType** offers some control.

```
// Delphi map component EMap

// valeur par défaut,
// importe/exporte tout (map+overlay)
// remplace l'ancien contenu par le nouveau
map.toTxtType := [ttaMap,ttaOverlays,ttaReplace];

// importe/exporte tout (map+overlay)
// ajoute les nouveaux overlays à ceux déjà existant
map.toTxtType := [ttaMap,ttaOverlays];

// importe/exporte uniquement les paramètres de la carte
map.toTxtType := [ttaMap];

// importe/exporte uniquement les overlays
map.toTxtType := [ttaOverlays];
```

text format

each part is divided into section, data are named and need not be in any

particular order.

Overlays for each line corresponds to a data element (a marker, a circle etc.)

Example map

[map]

draggable=true

draggablecursor=CrossHair

draggingcursor=

enableuicontrol=true

maptypecontrol=false

navigationcontrol=true

scalecontrol=false

zoomcontrol=true

mobilesenabled=true

latitude=43.2318637451068

longitude=0.0860298080261312

maptypeid=ROADMAP

maptypecontrolstyle=DEFAULT

maptypecontrolposition=TOP_RIGHT

navigationstyle=DEFAULT

navigationposition=TOP_LEFT

scalestyle=DEFAULT

scaleposition=BOTTOM_LEFT

zoomstyle=DEFAULT

zoomposition=TOP_LEFT

title=démo tarbes composant google map

zoom=15

[markers]

adress=;cursor=;icon=http://google-maps-icons.googlecode.com/files/home.png;sha

```

adress=;cursor=;icon=;shadow=;title=classic;animation=;name=;icon.width=32;icon.
adress=;cursor=;icon=http://maps.google.com/mapfiles/kml/pal4/icon40.png;shadow
adress=;cursor=;icon=http://google-maps-icons.googlecode.com/files/cycling.png;sh
[infowindows]

```

```

content=<h1>Nuage</h1><br>Image locale;anchor=2;maxwidth=0;zindex=0;lat=43.
[routes]

```

```

startlat=43.23443;startlng=0.0737;endlat=43.24342;endlng=0.09581;wplat0=43.241

```

```

/ Aureilhan;marker.icon=http://maps.google.com/mapfiles/ms/micons/blue-pushpin.p
startlat=43.23313;startlng=0.07419;endlat=43.22849;endlng=0.10676;travelmode=D

```

```

/ Séméac;marker.icon=http://www.visual-case.it/vc/pics/casetta_base.png;marker.vis
startlat=43.23481;startlng=0.07882;endlat=48.85631;endlng=2.35001;travelmode=D

```

```

/ Paris;marker.icon=http://maps.google.com/mapfiles/ms/micons/purple-pushpin.png
startlat=43.00821;startlng=-0.09972;endlat=42.85121;endlng=-0.13994;travelmode=
d'espagne;marker.icon=;marker.visible=true;marker.clickable=true;marker.flat=false
[mobiles]

```

```

0

```

```

distance=148;marker=2;route=1;speed=60;segment=2;mobile=true;direction=endsta
distance=338133;marker=1;route=2;speed=130;segment=2272;mobile=true;directio
distance=21139;marker=3;route=3;speed=30;segment=329;mobile=true;direction=s

```

```

[streetview]

```

```

latitude=-1

```

```

longitude=-1

```

```

visible=false

```

```

heading=140

```

```

pitch=10

```

```

zoom=1

```

```

navstyle=DEFAULT

```

```

navposition=TOP_LEFT

```

```

adrposition=TOP_LEFT

```

```
navcontrol=true
```

```
adrcontrol=true
```

```
closebtn=true
```

```
link=true
```

```
[polylines]
```

```
color=255;opacity=50;weight=5;zindex=1;latlng=43.2287477339158,0.07309683223
```

```
[polygons]
```

```
color=16744448;opacity=80;weight=5;zindex=1;latlng=43.2392845896161,0.074169
```

```
[circles]
```

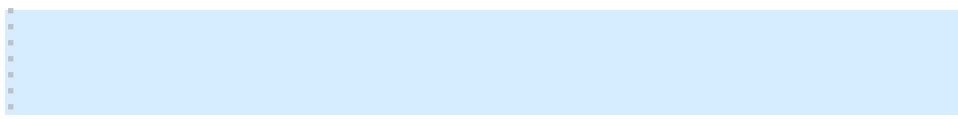
```
color=16384;opacity=80;weight=3;zindex=1;latlng=43.2348454,0.0756332;visible=tr
```

```
[rectangles]
```

```
[labels]
```

Import / Export overlays

Overlays have also toTxt property allowing the importation and exportation, either in their list ([Markers](#) , [Routes](#) etc.) than the item itself. In terms of import lists is necessarily an addition, it is not considered property toTxtType which is supported at the global level.



When using overlays toTxt you should not give section, all properties are not necessary.

Overlays are all the items based on geographical points that you can embed on your cards, they descend from the class `TECMapItem`.

TECMapItem

procedure **SetPosition**(const dLatitude, dLongitude: double);

The geographical position of the element

procedure **AddToGroup**(const GroupName:string);

Adds the item in a [Group](#)

property **Id**: integer;

Index of the element in its list

property **ItemType** : TOverlayType ;

TOverlayType = (ovNone, ovMarker, ovRoute, ovLine, ovPolygone, ovMap, ovCircle, ovRectangle, ovLabel, ovGlobe, ovKml, ovFusion, ovGroundOverlay, ovWeather);

property **Group** : [TECMapItemGroup](#);

Indicates the group to which the element belongs, position at Nil to the element of the Group

Liste d'overlays

Each type is managed in a separate list, you can add, delete, import / export your items

The lists are accessible through the properties:

- [Circles](#)
- [GroundOverlays](#)
- [KmlLayer](#)
- [Labels](#)
- [Markers](#)
- [InfoWindows](#)
- [Polylines](#)
- [Polygones](#)
- [Rectangles](#)
- [Routes](#)

they are manipulated through methods and properties similar, only setting up the add function differs.

function **Add**:integer;

Pour les **Polylines**, **Polygones** et **Labels**

function **Add**(const dLatitude,dLongitude:double):integer;

Pour les **Markers**, **Circles** et **Rectangles**

function **Add**(const Url:string;const dLatitude,dLongitude:double):integer;

Pour les **GroundOverlays**, l'url pointant une image

function **Add**(const Url:string):integer;

Pour les **KmlLayers**, l'url pointant un fichier Kml

function **Add**(const sName:string;const
dStartLatitude,dStartLongitude,dEndLatitude,dEndLongitude:double;const
OptimizeWaypoints:boolean=false;const WayPoints:TLatLngList=nil):integer;

For **Routes**

*Add back in all cases the index number of the
added element*

```
// Delphi map component EMap
// add circle at center of map
id := map.Circles.add(map.latitude,map.longitude);
// fix radius to 500 meters
map.Circles[id].Radius := 500;
```

procedure **Clear**;

removes all elements

function **Count**:integer;

returns the element number

procedure **Delete**(index:integer);

remove the element number **index**

property **ToTxt** : string

property read / write to export / import format text all the elements

*The import works as an addition, the pre-existing elements are preserved, if you want to replace them before it is imported to use **Clear***

property **ToKml** : string

Export **kml** of all elements

The labels are not exported to KML

TECMapOverlay

The circles, rectangles, polylines, polygons and the type **TECMapOverlay** groundOverlays down, **TECMap** has a property of this type named **EditOverlay**.

By assigning a consistent overlay (circle, rectangle, polyline, polygon or GroundOverlay), you can change the mouse position and size directly from your card.

To do this you must switch **EditMode** property to True.

EditOverlay has the property **OverlayType** type of **TOverlayType** that lets you know the type of overlay

```
TOverlayType = (ovNone, ovMarker, ovRoute, ovLine, ovPolygone, ovMap, ovCircle,
  ovRectangle, ovLabel, ovGlobe, ovKml, ovGroundOverlay);
```

EditOverlay also a property **Id** integer that gives you the index number of your overlay in its list. // Delphi map component **ECMap**

```
// add circle at center of map
id := map.Circles.add(map.latitude,map.longitude);
// fix radius to 500 meters
map.Circles[id].Radius := 500;
// edit overlay with mouse
map.EditOverlay := map.Circles[id];
// map.EditOverlay.OverlayType = ovCircle
// map.EditOverlay.Id = id
```

Methods & Common Properties

This type of overlay sharing properties

Color : TColor

Line Color outside

Clickable : boolean

Whether to allow the element to react to mouse click, if **False** the overlay will not switch to edit mode even if **EditMode** is **True**.

Geodesic : boolean

Make each edge as a geodesic (a segment of a "great circle"). A geodesic is the shortest path between two points along the surface of the Earth.

Id : integer

Index number of the overlay in its list

InfoWindow : integer

Index number of the [InfoWindow](#) associated, -1 if no

When clicked on the overlay if InfoWindow is defined it will be open

 *Not available CloudMade*

Opacity : double

Percentage of opacity of the color

Visible : boolean

Visible or not

Weight : integer

Line thickness

ZIndex : integer

Index of depth compared to other overlays to determine if an overlay is "above" another

ToTxt : string

Property read / write to export / import format text all the properties of the overlay

Tag : integer

You can use this property as seems

When you change the properties Color, Opacity, Weight ZIndex or you must explicitly make a call to redraw the procedure for this to be taken into account

```
// Delphi map component EMap  
  
map.Polylines[0].Color := clRed;  
map.Polylines[0].Weight := 5;  
// change property  
map.Polylines[0].reDraw;
```

The circles, rectangles and polygons have more properties

Area : double

Surface m²

Distance : double

Distance from the periphery in mètre

FillColor : TColor

Interior Color

FillOpacity : double

Percentage of opacity of the color of interior

Polylines also have ownership Distance

The circles have the properties

Center : TLatLng

Coordinates of the center of the circle

Radius : Integer

Radius in meters

The rectangles have the properties

Height : double

Height in meters

Width : double

Width in meters

TLatLngList

Polylines and polygons have a **Path** property type **TLatLngList**, list of points comprising the figure, the following main features.

function **Add**(const dLatitude,dLongitude:double):integer;

Add a dot at the end of list

procedure **Insert**(const index:integer;const dLatitude,dLongitude:double);

Insert point index

procedure **BeginUpdate**;

IMPORTANT : before adding a series of points made a call to **BeginUpdate** for triggering the reconstruction once all items inserted.

procedure **Clear**;

Clears all points

function **Count**:integer;

Returns the number of points

procedure **Delete**(index:integer);

Clears the index point

procedure **EndUpdate**;

IMPORTANT: At the end of a series of addition or insertion, made a call to **EndUpdate** To

reflect the changes

procedure **PanToBounds**;

Drag the map so that the figure is fully visible

function **getAdress**(const index:integer):string;

Returns the address of the point index

procedure **getAltitudes**(Event:TOnGetAltitude=nil);

Calculates the elevations of points, you can pass a procedure type **TOnGetAltitude** to display a progress bar (see [demoOverlay](#) and [DemoRoute](#))

```
// Delphi map component EMap

// calcul altitude for all point of polyline 0
// you can pass nil if you don't show a progressbar
map.Polyline[0].Path.GetAltitudes(doGetAltitude);
...
{ *
  event fired by getAltitudes

  @param Sender TLatLngList
  @param Total number of altitude's point calculated
  @cancel flag for abort calcul
}
procedure
TFDemoRoute.doOnGetAltitude(Sender: TLatLngList; const
Total:integer; var cancel:boolean);
begin
  ProgressAltitude.Position := total;
  // cancel if press button
  cancel := btAbortAlt.tag = -1;
end;
```

You can directly create a TLatLngList and fill it with your points to get their altitude without having to go through a Polyline.

property **Point**[index:integer]:TLatLng read getPoint; default;

Returns TLatLng corresponding to Index

function **getLatLngFromMeter**(const SensStartEnd:boolean;const IMeter:longint;**var**

dLatitude,dLongitude:double;**var** idPoint:integer;**var** heading:integer;**var** bEnd:boolean):boolean;

Calculates the latitude and longitude of a point on the polyline / polygon depending on its distance in meters, returns **True** if we found a point

SensStartEnd direction of travel, true to start -> finish

IMeter the distance in meters

dLatitude,dLongitude type variables **double** who will receive the latitude and longitude

idPoint a variable that will contain the index in the table **Point** where is the point, the calculation returns an approximation because your polyline / polygon does not contain all the real points

Heading a variable to hold the angle of the point from the north (0 to 360°)

bEnd indicates whether it has exceeded or reaches the end of the polyline / polygon (or early depending on the direction)

The routes have the same function

Example of handling a Polyline

```
// Delphi map component ECGMap
```

```
// add new polyline
```

```
id := map.Polylines.add;
```

```
// IMPORTANT for more speed !
```

```
map.Polylines[id].Path.BeginUpdate;
```

```
// add points to polyline id
```

```
map.Polylines[id].Path.Add(37.772323, -122.214897);
```

```
map.Polylines[id].Path.Add(21.291982, -157.821856);
```

```
map.Polylines[id].Path.Add(-18.142599, 178.431);
```

```
map.Polylines[id].Path.Add(-27.46758, 153.027892);
```

```
// update data
```

```
map.Polylines[id].Path.EndUpdate;
```


Events

The circles, rectangles, polylines, polygons and meet the following events groundoverlays

OnOverlayClick(sender: Tobject;const Index:integer;const dLatitude,dLongitude:double;const OverlayType:TOverlayType)

simple click on an overlay

index is the index of the overlay in its list

dLatitude,dLongitude geographical coordinates of the click

OverlayType the type of overlay (ovCircle, ovRectangle, ovLine, ovPolygone,ovLabel ou ovGroundOverlay)

OnOverlayMouseDown(sender: Tobject;const Index:integer;const dLatitude,dLongitude:double;const OverlayType:TOverlayType)

the left mouse button is pressed on an overlay

index is the index of the overlay in its list

dLatitude,dLongitude the geographical coordinates of the click

OverlayType the type of overlay (ovMarker,ovCircle, ovRectangle, ovLine, ovPolygone,ovLabel ou ovGroundOverlay)

The markers meet this event !

OnOverlayMouseUp(sender: Tobject;const Index:integer;const dLatitude,dLongitude:double;const OverlayType:TOverlayType)

the left button of the mouse is found on an overlay

index is the index of the overlay in its list

dLatitude,dLongitude the geographical coordinates of the click

OverlayType the type of overlay (ovMarker,ovCircle, ovRectangle, ovLine, ovPolygone,ovLabel ou ovGroundOverlay)

The markers meet this event !

CloudMade does not respond to events OnOverlayMouseDown and OnOverlayMouseUp

OnOverlayDbClick(sender: Tobject;const Index:integer;const dLatitude,dLongitude:double;const OverlayType:TOverlayType)

Double click on an overlay

OnOverlayRightClick(sender: Tobject;const Index:integer;const dLatitude,dLongitude:double;const OverlayType:TOverlayType)

Right click on an overlay

CloudMade does not support right click, Alt + click trigger event (as in Google Map)

OnOverlayMove(sender: Tobject;const Index:integer;const dLatitude,dLongitude:double;const OverlayType:TOverlayType)

Triggered when moving an overlay, by code or mouse

Under Google Map you can drag and drop to move the overlay pointed by EditOverlay, but under CloudMade you must first do a CTRL + click on the map, then move the mouse while holding down CTRL to move your item (also works with Google Map)

OnOverlayMouseOut(sender: Tobject;const Index:integer;const dLatitude,dLongitude:double;const OverlayType:TOverlayType)

The mouse leaves an overlay

OnOverlayMouseOver(sender: Tobject;const Index:integer;const dLatitude,dLongitude:double;const OverlayType:TOverlayType)

Mouse over an overlay

OnOverlayChange(sender: Tobject;const Index:integer;const OverlayType:TOverlayType)

Triggered when any change on an overlay (color, point ...)

OnOverlayPathChange(sender: TObject; const Index: integer; const OverlayType: TOverlayType; const PathIndex: integer)

Triggered when one of the points defining the overlay changes

PathIndex indicates the index in Path

Markers Interactive

The interactive overlay markers assigned to EditOverlay also trigger events

OnEditOverlayPointClick(sender: TObject; const Index: integer)

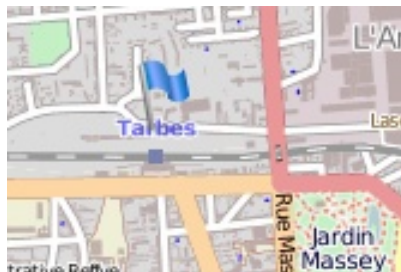
Click on the marker that represents the Index point in Path


OnEditOverlayPointDragEnd(sender: TObject; const Index: integer; const dLatitude, dLongitude: double)

Point displacement Index on Path and its new coordinates

OnEditOverlayPointRClick(sender: TObject; const Index: integer)

Click on the marker that represents the Index point in Path



By default interactive markers are represented by  , you can change the icon through the Propertie **IconOverlay** type String, image may be a remote image on the Internet or a local file.

```
// Delphi map component EMap
// red marker
map.IconOverlay :=
;http://maps.google.com/mapfiles/ms/micons/red-pushpin.png'
```

If you change the icon you may also need to edit the properties

IconSizeOverlay : TPoint

Width and Image Height

IconOriginOverlay : TPoint

Original X and Y displayed area extracted from the image (same image can contain multiple icons)

IconAnchorOverlay : TPoint

X and Y point in the area that will match the latitude and longitude



Fig. 32 Polygon editing mode

Demonstration

The program DemoOverlay shows you how to easily manipulate all these elements

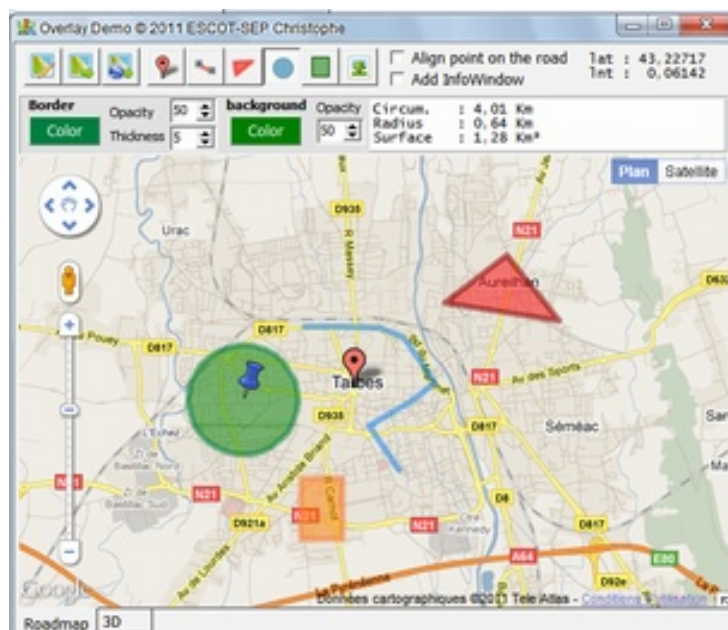


Fig. 33 DemoOverlay

All the concepts related to overlays are employed, creation, Publishing, deletion, the [import / export of your card](#) , including export to [Google Earth](#) .

The markers are managed by a list of type **TECMapMarkers** accessible through the property **Markers** of TECMap

TECMapMarkers

This list has the following methods and properties :

function **Add**(const dLatitude,dLongitude:double):integer;

Adds a marker at the point Latitude, Longitude and returns its index in the list.

You can also add a marker with the function AddMarker of TECMap

```
// Delphi map component EMap  
  
// add marker at center of map  
id := map.AdMarker(map.latitude,map.longitude);  
// set title to this marker  
map.Markers[id].title := 'my first marker!';
```

function **ByLatLng**(const dLatitude,dLongitude:double):TECMapMarker;

Returns the marker positioned at Latitude, Longitude

function **ByName**(const value:string):TECMapMarker;

Returns the marker based on its name (Property Name)

procedure **Clear**;

Deletes all markers

procedure **Delete**(index:integer);

Removes the marker which we pass the index

function **IndexOf**(const value:TECMapMarker):integer;

Returns the index of the marker in the list

property **Cluster**:boolean;

Property **read / write**, includes the markers according to their position and according to the zoom



Fig. 34 Combination of markers of activation

Inactive under CloudMade

property **Count**:integer;

Returns the number of marker in the list

procedure **fitBounds**

Adjusts the zoom of the map to display all of the markers

property **Marker[index:integer]**:TECMapMarker;

Table that allows access to markers is the default property so you can access it directly by **map.Markers [index]** instead of **map.Markers.Marker [index]**

property **ToKml** : string;

Read-only property that returns a string in **kml format** containing the list of markers

property **ToTxt** : string;

Property **read / write** which gives access to the list of markers in a text format.

*In writing there is an addition and not a replacement, if you do not want to keep the old values before making a **Clear** adding*

Example of use

```
// Delphi map component EMap

// add marker at center of map
id := map.Markers.add(map.latitude,map.longitude);
// set title to this marker
map.Markers[id].title := 'my first marker!';
```

TECMapMarker

This class manages a marker, it has the following methods and properties :

procedure **setPosition**(const dLatitude,dLongitude:double);

Moves the marker in Latitude, Longitude, triggers the event **OnMarkerMove**

procedure **PanTo**;

Move the map so that its center coincides with the marker, triggers the event **OnMapMove**

The movement will be smoother if the move does not exceed half the height or width of the card.

procedure **CopyImage**(const Source:TECMapMarker);

Copy the image options the marker passed as a parameter

procedure **setImage**(const _Icon:string;Size,Origin,Anchor:TPoint);

_Icon contains the name of the source image file, local or internet see **Icon**

Size defined the width and height, see **IconSize**

Origin set the X and Y from the origin of the image in the source image, see **IconOrigin**

Anchor set the X and Y point corresponding to the Latitude and Longitude, see **IconAnchor**

procedure **setImageShadow**(const _Icon:string;Size,Origin,Anchor:TPoint);

Change options for the shadow image

procedure **setShape**;

Zone enforcement clickage the marker, see property Shape

property **Address** : string;

Property **read / write**, can get the address of the marker or set it to an address, triggers the event **OnMarkerAddress**

To avoid multiple connections to the server address is cached, a new connection will be made only if the marker is moved

property **Animation** : string;

Property **read / written** values available are listed in the property **MarkerAnimations** type TStringList of TECMap

With the Google API you 'BOUNCE' and 'DROP', put an empty string to stop the animation

Inactive under CloudMade

property **Altitude** : double ;

Property **read / written**, the value is cached to avoid multiple connections to the server, a new connection will be made only if the position of marker exchange.

property **Cursor** : string;

Property **read / written**, set the mouse cursor when hovering, the values are listed in the property **Cursors** type TStringList de TECMap

Inactive sous CloudMade

property **Clickable** : boolean;

Property **read / written**, defined if the marker receives the click of the mouse or not

property **Draggable** : boolean;

Property *read / written*, defined if the marker is moved to the mouse.

Even if a marker is not moved with the mouse you can change its position through setPosition code or properties Latitude and Longitude

property **Dragging** : boolean;

Indicates whether the marker is about to be moved

property **Flat** : boolean;

Property *read / written*, defined if the marker has a shadow

property **Heading** : integer;

Indicates the angle relative to north

property **Icon** : string;

Property *read / written*, defined the image of the marker, this can be either a local file or a URL of a file on the Internet, see also setImage

With Google Maps you can also use a

```
// Delphi map component EMap
// set vector icon
map.Markers[0].Icon := '{path:
google.maps.SymbolPath.CIRCLE,scale: 3,strokeColor: "#393"}'
```

property **Shadow** : string;

Property *read / written*, defined the image of the shadow of the marker, this can be either a local file or a URL of a file on the Internet, see also setImageShadow

property **IconSize** : TPoint;

set the Width and Height of the image

property **IconOrigin** : TPoint;

set the X and Y from the origin of the image in the source file

property **IconAnchor** : TPoint;

set the X and Y in the image of the point corresponding to the Latitude and Longitude

property **Index**;

Index marker in the list of markers

property **InfoWindow** : integer;

Index of [InfoWindow](#) associated marker in the list of InfoWindow

When clicked on the marker if InfoWindow is defined, it will open

property **Name** : string;

Property *read / written*, set the name of your marker, see [ByName](#)

property **Latitude** : double;

Property *read / written*, defining the latitude of the marker triggers the event **OnMarkerMove**

property **Longitude** : double;

Property *read / written*, defining the longitude of the marker triggers the event **OnMarkerMove**

property **Tag** :integer;

Property *read / written*, you can use it freely for whatever you want stoker

property **Title** : string;

Property *read / written*, defining the title of your marker, it appears as a tooltip when hovering the cursor

property **Visible** : boolean;

Property *read / written* to display or not the marker

property **Zindex** : integer;

Property *read / written*, a priority key attribute for displaying the marker, a marker with a higher zIndex will be displayed on top of that having a lower zIndex

property **ShowOnMap** : boolean;

Read-whether the position marker is in the visible portion of the map

property **Shape** : TECMapShape;

Used to define the clickable area of the marker.

```
// Delphi map component EMap

// fix shape rectangle
map.Markers[0].shape.stype := 'rect';
// fix coord
map.Markers[0].shape.Coord.add([10,10,25,30]);
// set shape
map.Markers[0].setShape;
```

With **SType** 'circle', 'poly' or 'rect'

See [google documentation](#)

Inactive under CloudMade

property **ToTxt** : string;

Property **read / write** accesses the marker in the form of a text string

property **ToKml** : string;

Read-only property that returns the marker as a string in [kml format](#)

Events

*You can connect the component to receive events **TECMap** but you can also be connected directly from a marker for a specific response, in which case **the event overall TECMap will not be called***

The markers trigger events:

OnMarkerMove(sender: TObject;const Index:integer;var dLatitude,dLongitude:double)

Triggered when moving a marker, by code or mouse

Index is the index of the list marker Markers

dLatitude and **dLongitude** the new position

You can change it in the event

OnMarkerClick(sender: Tobject;const Index:integer;const dLatitude,dLongitude:double)

Triggered when a click on the marker

Index is the index of the list marker Markers

dLatitude and **dLongitude** position of marker

OnMarkerRightClick(sender: Tobject;const Index:integer;const dLatitude,dLongitude:double)

Raised in a right click on the marker

Index is the index of the list marker Markers

dLatitude and **dLongitude** position of marker

OnMarkerDbiClick(sender: Tobject;const Index:integer;const dLatitude,dLongitude:double)

Triggered when double click on the marker

Index is the index of the list marker Markers

dLatitude and **dLongitude** position of marker

OnMarkerDrag(sender: Tobject;const Index:integer;const dLatitude,dLongitude:double)

Triggered when moving the mouse of a marker

Index is the index of the list marker Markers

dLatitude and **dLongitude** position of marker

OnMarkerDragStart(sender: Tobject;const Index:integer;const dLatitude,dLongitude:double)

Triggered initially moving the mouse of a marker

Index is the index of the list marker Markers

dLatitude and **dLongitude** position of marker

OnMarkerDragEnd(sender: Tobject;const Index:integer;const dLatitude,dLongitude:double)

Triggered at the end of the mouse to move a marker

Index is the index of the list marker **Markers**

dLatitude and **dLongitude** position du marker

OnMarkerAdress(sender: Tobject;const sAdresses:string;var Index:integer;const IndexMarker:integer)

Triggered by assigning an address to the marker

sAdresses contain all addresses, separated by **#13#10** can match your query.

Index contain the index of the selected address, default 0, you can change it in this event.

IndexMarker index marker in the list of **Markers**

Output from this event the marker is positioned on the address selected

*The markers also meet events **OnOverlayMouseDown** and **OnOverlayMouseUp***

Set a marker on a path

```
function MoveMarkerOnRouteToMeter(const iNumMarker,iNumRoute:integer;
const RouteType: TOverlayType;
const Direction:TDirectionSens;
const IMeter:longint):integer;
```

This feature allows you to place a marker on a path (road, polyline or polygon) based on a direction and distance in meters.

iNumMarker is the index marker in the list **Markers**

iNumRoute is the index of the path in **Routes**, **Polylines** ou **Polygones**

RouteType determines the type of path, **ovRoute**, **ovLine** ou **ovPolygone**

Direction is the direction of travel is **dsStartEnd** or **dsEndStart**

IMeter the distance in meters

In the next chapter you'll find other features to [make your mobile markers](#) simply

You can define a list of [markers](#) that have the ability to move along a predetermined path ([roads](#) , [polylines](#) or [polygons](#)) .

This list type [TMobileList](#) is available through the property **Mobiles**.

TECMap available in more than two properties related to mobiles

property **MobilesEnabled**:boolean

Property **read / write** or not to authorize the automatic displacement of mobile, false by default

property **MobilesTiming** : dword

Property **read / write** in milliseconds indicating the time interval between each shift automatic, default **300ms** (the minimum interval is **100ms**)

TMobileList

You have access to methods and properties

function **IndexOf**(const iMarker:integer):longint;

Returns the index of the mobile according to an index marker

function **add**(const iMarker,iRoute,iSpeed,iDistance:integer):longint;

Adds a mobile

iMarker is the index of the list marker [Markers](#)

iRoute index is the way forward is in [Roads](#) (default) or in [Polygons or Polylines](#)

iSpeed is the mobile speed in **km / h**

iDistance is the starting position on the road in **meters**

```
// Delphi map component EMap

map.DistanceMatrix.origins.clear;
map.DistanceMatrix.destinations.clear;
// origins
map.DistanceMatrix.origins.add('Tarbes');
map.origins.add('Lourdes');
// destinations
map.DistanceMatrix.destination.add('Aureilhan');
map.DistanceMatrix.destination.add('Séméac');
// calcul - fire OnDistanceMatrice
```



```

Map.DistanceMatrix.Update;
...

// event fired when matrix is ok, call by map.DistanceMatrix.Update
procedure
  TFormDemoMatrix.mapDistanceMatrix(Sender: TObject);
begin
  if map.DistanceMatrix.count=0 then exit;
  // Tarbes/Aureilhan
  map.DistanceMatrix[0,0].distanceText;
  // Tarbes/Séméac
  map.DistanceMatrix[0,1].distanceText;
  // Lourdes/Aureilhan
  map.DistanceMatrix[1,0].distanceText;
  // Lourdes/Séméac
  map.DistanceMatrix[1,1].distanceText;

end;

```

procedure **clear**;

Removes all mobiles

procedure **delete**(const index:integer);

Clears the mobile index Index

function **count**:longint;

Returns the number of mobile

property **Mobiles**[index:integer]:[TMobile](#)

Table that allows access to mobile is the default property so you can access it directly by **map.Mobiles[index]** instead of **map.Mobiles.Mobile[index]**

property **Tracking** :integer

Property **read / write** indicating what is the motive that we will follow the card is automatically refocus on him when he crosses the boundaries visible.

Set this property to -1 to not follow any mobile

property **ToTxt** : string

Property **read / write** which gives access to the list of mobiles in a text format.

By default, the mobile does not move, you can move in either automatic mode or manual mode.

In automatic mode they will move at the speed that you've assigned them in manual mode or you can directly manipulate the distance is to be calculated according to time and speed.

Even in automatic mode you can manually change the distance, speed, you can also change the road on which the mobile moves, its position is automatically adjusted.

To do this you must access the class `tmobile`

TMobile

function **Move**:integer;

Mobile Moves by calculating its distance from the starting point versus time elapsed since the last move, speed and direction.

property **Distance** : integer;

Property **read / write** sets the distance in meters, mobile since its real starting point, management is not taken into account.

Any change of this property leads to a shift of the marker associated with mobile

property **Direction**: TDirectionSens;

Direction of movement, **dsStartEnd** or **dsEndStart**

For a road **dsStartEnd** corresponds to a displacement of the point of departure or arrival point for polylines and polygons of the first point to last.

The distance is always calculated based on the real starting point, management will not be decisive for a shift automatic or manual call to **Move**.

property **Index** : integer

Index in the list of mobile phones

property **Marker** : integer

Property **read / write** the index of the **Marker** to move

property **Mobile** : boolean

Property **read / write** set if the mobile moves automatically or not, by default set to **false** the mobile does not move by itself.

property **Route** : integer;

Property **read / write** sets the index of the path in its respective list (**Routes**, **Polylines** or **Polygones** depending **RouteType**)

property **RouteType**: **TOverlayType**;

Property **read / write** defined path type, **ovRoute**, **ovLine** or **ovPolygone**

property **Speed** : integer

Property **read / write** that determines the speed km / h

property **Step** : integer;

Index in the **Path** property of the road, polyline or polygon that is mobile, the calculation returns an approximation of the point as your route does not contain all the real points.

property **Heading** : integer;

Angle of mobile degrees relative to North (0 ° to 360 °)

property **ToTxt** : string ;

Property **read / write** which gives access to mobile in a text format.

Events

Two events are related to mobile

OnMarkerMove(sender: Tobject;const
Index:integer;var dLatitude,dLongitude:double)

Triggered when moving a marker, by code
or mouse

Index is the index of the list marker **Markers**

dLatitude an **dLongitude** the new position

You can change it in the event

procedure **OnEndMobile**(sender:
Tobject;const iMobile:integer);

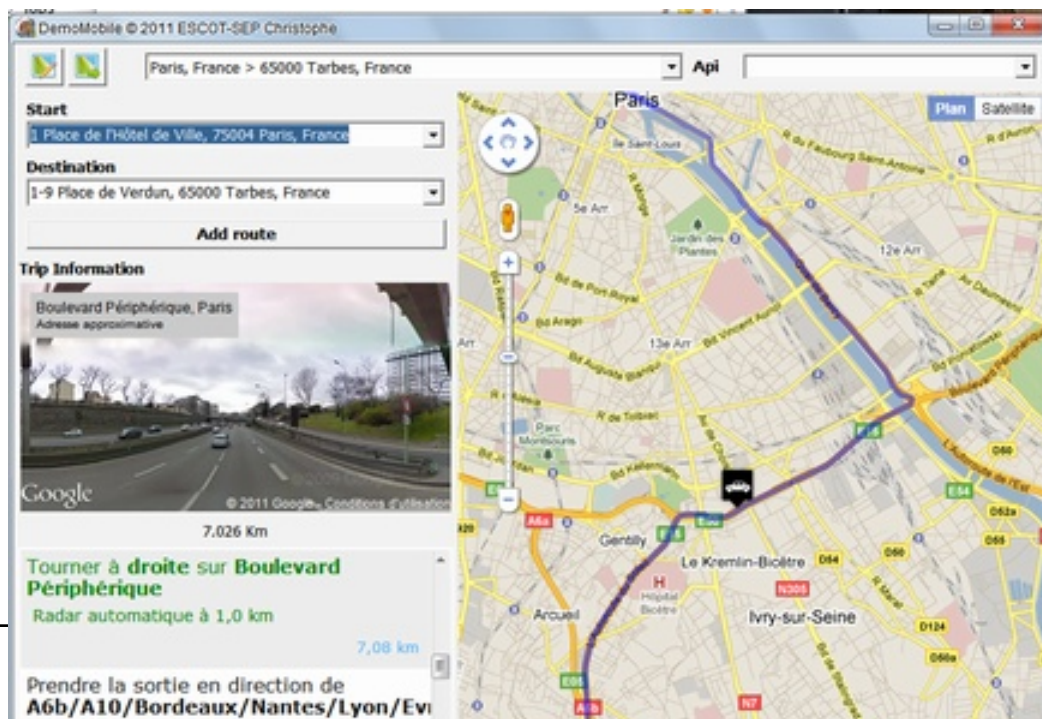
iMobile index in the list of mobile **Mobiles**

Depending on the direction the mobile is either at
the terminus or point of departure

Property **Mobile** is set to **false**, to boost your
mobile in automatic mode you must return it to
true and reverse the direction.

Demonstration

The program shows you **DEMOmobile** use of mobile



A InfoWindow is a bubble of information that can contain HTML text, they are accessible through the list InfoWindow type TECMapInfoWindows

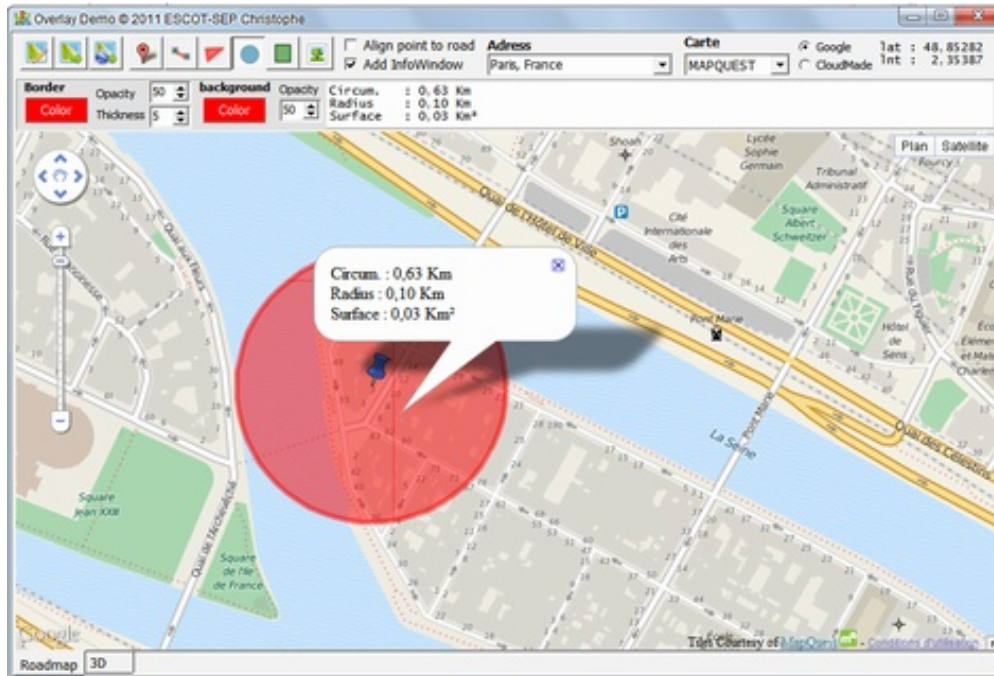


Fig. 36 A InfoWindow associated with a circle

TECMapInfoWindows

This class has methods and properties

```
function Add(sContent:string):integer;
```

Adds InfoWindow

content can be plain text or HTML

procedure **Clear**;

Clears all InfoWindow

procedure **Delete**(index:integer);

Deletes InfoWindow which we pass the index

property **Count**:integer;

Returns the number of marker in the list

property **InfoWindow**[index:integer]:TECMapInfoWindow;

Table that allows access to InfoWindow is the default property so you can access it directly by **map.InfoWindows[index]** instead of **map.InfoWindows.InfoWindow[index]**

property **ToKml** : string;

Read-only property that returns a string in *kml format* listing the infowindows

property **ToTxt** : string;

Property **read / write** which gives access to the list in text format InfoWindow.

*In writing there is an addition and not a replacement, if you do not want to keep the old values before making a **Clear** adding*

TECMapInfoWindow

The InfoWindow have methods and properties

procedure **setPosition**(const dLatitude,dLongitude:double);

Moves InfoWindow in Latitude, Longitude

Not available CloudMade

property **Anchor** : integer;

Property **read / written** InfoWindow combines to a *marker*

Under the CloudMade InfoWindow must necessarily be associated with a marker to be displayed

property **Content** : string;

Property **read / written** defining the content of InfoWindow

property **Latitude** : double;

Property **read / written** with access to the latitude

 *Not available CloudMade*

property **Longitude**: double;

Property **read / written** with access to the longitude

 *Not available CloudMade*

property **MaxWidth** : integer;

Property **read / written** to define the maximum width that can reach InfoWindow

 *Not available CloudMade*

property **Open** : boolean

Property **read / written** to open / close a InfoWindow

The closure of a InfoWindow triggering event **OnCloseInfoWindow**

property **Zindex** : integer;

Property **read / written**, attribute a priority index for displaying the InfoWindow a InfoWindow with a zIndex higher will be displayed on top of a lower zIndex

property **Index** : integer;

Index InfoWindow in its list

property **ToKml** : string;

Read-only property that returns a string in [kml format](#) containing InfoWindow

property **ToTxt** : string;

Property *read / write* which gives access to InfoWindow in a text format.

OnCloseInfoWindow

Event triggered the closure of a InfoWindow

procedure **OnCloseInfoWindow**(sender: Tobject;const Index:integer)

Index index in the list InfoWindow [InfoWindow](#)

Difference between API

Under CloudMade to be displayed a InfoWindow must be associated with a [marker](#)

```
map.Routing.TurnByTurn.OnInstruction
:= doOnTurnByTurnInstruction;
map.Routing.TurnByTurn.OnAfterInstruction
:= doOnTurnByTurnInstruction;
map.Routing.TurnByTurn.OnAlert
:= doOnTurnByTurnAlert;
map.Routing.TurnByTurn.OnArrival
:= doOnTurnByTurnArrival;
map.Routing.TurnByTurn.OnError
:= doOnTurnByTurnError;

procedure TFDemoNativeRoute.doOnTurnByTurnAlert(sender
: TECTurnByTurn;const Instruction:string;const
Distance:double);
begin
    // alert is fired before OnInstruction
end;

procedure TFDemoNativeRoute.doOnTurnByTurnInstruction(sender
```

```
: TECTurnByTurn;const Instruction:string;const  
Distance:double);  
begin  
    // execute instruction in Distance (km)  
end;
```

While in Google Maps you can directly open a InfoWindow without associating it with a marker

Not available CloudMade

Labels are containers Html movable with the mouse, they are manipulated through the [Labels list](#).

```
// Delphi map component EMap
// add label at center of map
id := map.Labels.add;
map.labels[id].setPosition(map.latitude, map.longitude)
// set draggable
map.Labels[id].draggable := true;
```

They have properties and methods

procedure **setPosition**(dlatitude,dLongitude:double);

Moves the label in Latitude, Longitude, triggers the event OnOverlayMove

property **Content** : string

Properties for *reading / writing* describing the contents of the HTML label

property **Css** : string;

Properties *read / write* CSS styles applied to the label

property **Draggable** : boolean

Property *read / written*, set the label is moved to the mouse.

Even if a label is not moved with the mouse you can change its position through setPosition code or properties Latitude and Longitude

property **Id** : integer

Index number of the label in its list

Property *read / written*, defining the scope of the label, the event triggers OnOverlayMove

property **Longitude** : double;

Property **read / written**, defining the longitude of the label, the event triggers OnOverlayMove

property **Visible** : boolean;

Property **read / written** to display or not the label

property **ZIndex** : integer;

Property **read / written**, a priority key attribute for displaying the label, a label with a zIndex higher will be displayed on top of that having a lower zIndex

property **ToTxt** : string;

Property **read / write** access to the label which gives the form of a text string

```
// Delphi map component EMap
// add label at center of map
id := map.Labels.add;
map.labels[id].setPosition(map.latitude, map.longitude)

// set draggable
map.Labels[id].draggable := true;
// set content html
map.Labels[id].Content := 'my <b>label</b> html';
// set css style
map.Labels[id].Css := 'background: red;border: 2px solid
#fff;padding: 3px;';
```

Events

Labels respond to events:

OnOverlayClick(sender: Tobject;const Index:integer;const dLatitude,dLongitude:double;const OverlayType:TOverlayType)

simple click on a label

index is the index of the label in its list

dLatitude,dLongitude geographical coordinates of the click

OverlayType the type of overlay (ovCircle, ovRectangle, ovLine, ovPolygone,ovLabel ou ovGroundOverlay), So **ovLabel** for a label

OnOverlayMove(sender: Tobject;const Index:integer;const dLatitude,dLongitude:double;const OverlayType:TOverlayType)

Triggered when moving a label, with code or mouse

OnLink(sender: TObject;const Url:string)

If your label contains a link you catch the click event in this event

Url contains the target link

The roads are managed by the list **Routes** Type **TECMapRoutes**

TECMapRoutes

This list has the following methods and properties :

```
function Add(const sName:string;const dStartLatitude,dStartLongitude,dEndLatitude,dEndLongitude:double;
const OptimizeWaypoints:boolean=false;const WayPoints:TLatLngList=nil):integer;
```

Adding a road, returns the index of the road in the list

Raises the event [OnRouteChange](#) on success otherwise OnRouteError

Name name of your road, can be empty

dStartLatitude,dStartLongitude starting point of the road

dEndLatitude, dEndLongitude endpoint of the road

OptimizeWayPoints force the calculation of the most direct route by reorganizing your crossings (**has no action under CloudMade**)

WayPoints list type [TLatLngList](#) of crossings.

Pass nil if you do not specify a particular crossing points

```
function AddByAdr(const sName,sStartAddress,sEndAddress:string;
const OptimizeWaypoints:boolean=false;const WayPoints:TLatLngList=nil):integer;
```

Adding a road, returns the index of the road in the list

Raises the event [OnRouteChange](#) on success otherwise OnRouteError

Name name of your road, can be empty

sStartAddress Address of the starting point of the road

sEndAddress Address endpoint of the road

OptimizeWayPoints force the calculation of the most direct route by reorganizing your crossings (**has no action under CloudMade**)

WayPoints list type [TLatLngList](#) of crossings.

Pass nil if you do not specify a particular crossing points

AddByAdr is not available CloudMade

procedure **Clear**;

Clears all routes

function **Count**:integer;

Returns the number of routes

procedure **Delete**(index:integer);

Clears the road index **Index**

procedure **Undo**(index:integer);

Cancels the change of direction made the mouse on the road index **Index**
You can undo changes in the last ten trip.

Not available CloudMade

function **Ready**:boolean;

Indicates whether you can add a route, you can not do when a road is about to be calculated.

property **TravelMode**: TDirectionsTravelMode

Property *read / write* which can select the type of route you want

property **Distance** : longint;

Read-only property that returns the length of all roads in **meters**

property **Duration** : longint;

Read-only property that returns the time it takes, in **seconds**, to make journeys.

Not available CloudMade

property **avoidHighways** : boolean;

Property **read / written** offering the choice of avoiding motorways or not

property **avoidTolls** : boolean;

Property **read / written** offering the choice whether to avoid toll roads

property **Color** : TColor;

Property **read / written** offering the choice to set the color of the road

property **Draggable** : boolean;

Property **read / written** that allows the mouse to change the route of the road.

With the API CloudMade you can not change the points of departure / arrival and passages

property **Opacity** : double;

Property **read / written**, which adjusts the percentage of opacity of the color of the road

property **Weight** : integer;

Property **read / written** which sets the size of the Roadway

property **ZIndex** : integer;

Property **read / written**, attribute a priority index for the display of the road, a road with a zIndex higher will be displayed on top of that with a lower zIndex

property **MarkerOptions** : [TECMapMarker](#);

property **Route**[index:integer]:TECMapRoute; default;

Table for road access is the default property so you can access it directly by **map.Routes [index]** instead of **map.Routes.Route [index]**

property **ToKml** : string;

Read-only property that returns a string in [kml format](#) containing the list of roads

property **ToTxt** : string;

Property **read / write** which gives access to the route list in text format.

*In writing there is an addition and not a replacement, if you do not want to keep the old values before making a **Clear** adding*


```

        // Delphi map component EMap

wp := TLatLngList.create(nil);
try
    // add waypoint
    wp.add(43.2237336276672,0.0462043685729441);

    map.Routes.Color := clGreen;

    id := map.Routes.Add('my first route',
        43.2328643,0.0741207999999946,
        43.0946324606031,-0.0254427986328665,true,
        wp);

finally
    wp.free;
end;

```

Events

OnRouteChange(sender: TObject;const idRoute:integer;const NewRoute:boolean)

idRoute is the index of the road in the list **Routes**

NewRoute equal **True** it is an addition to road, **False** if a rerouting

OnRouteError(sender: TObject;const idRoute:integer;const sError:string)

idRoute is the index of the road in the list **Routes**

sError explanation of the error

TECMapRoute

TECMapRoute is the class that handles a route, it gives you access to methods and properties :

```

function getLatLngFromMeter(const SensStartEnd:boolean;const IMeter:longint;var
    dLatitude,dLongitude:double;var idPoint:integer;var heading:integer;var bEnd:boolean):boolean;

```

Calculates the latitude and longitude of a point on the road according to its distance in meters, returns **True** if we found a point

SensStartEnd meaning of course, true for start -> finish

IMeter the distance in meters

dLatitude,dLongitude type variables **double** who will receive the latitude and longitude

idPoint a variable that will contain the index **Path** where is the point, the calculation returns an approximation, since your drive does not contain all the real points

*The Polylines / Polygons have a single function through their property **Path** type **TLatLngList***

Heading a variable to hold the angle of the point from the north (0 to 360°)

bEnd indicates whether it has exceeded or reaches the end of the road (or early depending on the direction)

procedure **fitBounds**

Adjusts the zoom of the map to display all of the road

procedure **Update**;

Redefines the road to reflect changes made to any of the properties

procedure **OptimizeWaypoints**;

Rearranges the way points for their order to be optimal

Not available CloudMade

procedure **updateOptions**;

Redraw the road to reflect properties **Color**, **Opacity**, and **Weight Draggable**

Read-only property that returns the length of the road in **meters**

property **Duration** : longint;

Read-only property that returns the time it takes in **seconds** for the journey.

Not available CloudMade

property **Path** : [TECMapRoutePath](#);

List Type TECMapRoutePath containing the road constituent points

property **WayPoints** : [TLatLngList](#);

List of crossings

property **Copyright** : string;

The possible copyright holder for this route

property **StartAddress** : string;

Address of starting point, *read-only*

Not available CloudMade

property **StartLatitude** : double;

Property *read / write* from the point of Latitude

property **StartLongitude** : double;

Property *read / write* Longitude of starting point

property **EndAddress** : string;

Address of the destination point, *readonly*

Not available CloudMade

property **EndLatitude** : double;

Property *read / write* from the point of arrival Latitude

property **endLongitude** : double;

Property *read / write* from the point of arrival Longitude

property **Name** : string;

Read-only property that returns the name of the road telqu'il has been indicated to the creation

property **Id** : integer;

Read-only property that returns the index of the road in the list [Routes](#)

property **NorthEastLatitude** : double;

Read-only property that returns the latitude of the north-east corner of the area surrounding the entire road

property **NorthEastLongitude** : double;

Read-only property that returns the longitude of the northeast corner of the area surrounding the entire road

property **SouthWestLatitude** : double;

Read-only property that returns the latitude of the southwest corner of the area surrounding the entire road

property **SouthWestLongitude** : double;

Read-only property that returns the longitude of the southwest corner of the area surrounding the entire road

property **TravelMode**: TDirectionsTravelMode

Property **read / write** which can select the type of route you want

property **avoidHighways** : boolean;

Property **read / written** offering the choice of avoiding motorways or not

property **avoidTolls** : boolean;

Property **read / written** offering the choice whether to avoid toll roads

property **Color** : TColor;

Property **read / written** offering the choice to set the color of the road

property **Draggable** : boolean;

Property **read / written** that allows the mouse to change the route of the road.

With the API CloudMade you can not change the points of departure / arrival and passages

property **Opacity** : double;

Property **read / written**, which adjusts the percentage of opacity of the color of the road

property **Weight** : integer;

Property **read / written** which sets the size of the Roadway

property **ZIndex** : integer;

Property **read / written**, attribute a priority index for the display of the road, a road with a zIndex higher will be displayed on top of that with a lower zIndex

property **MarkerOptions** : [TECMapMarker](#);

Property that allows you to manipulate the markers of departure and arrival by changing the icon for instance, unfortunately we can not yet differentiate.

 *Not available CloudMade*

property **Tag** :integer;


Property **read / written**, you can use it freely for whatever you want stoker

property **ToKml** : string;

Read-only property that returns a string in [kml format](#) contains the route

property **ToTxt** : string;

Property **read / write** which gives access to the road in a text format.

 *In writing there is an addition and not a replacement, if you do not want to keep the old values \u200b\u200b before making a **Clear** adding*

TECMapRoutePath

List of points on the route, it gives you access to methods and properties

function **Distance**:integer;

Distance in meters

function **Duration**:integer;

Duration in seconds

function **Count**:integer;

Returns the number of points constituting the road

function **IndexOfLatLng**(const dLatitude,dLongitude:double):integer;

Returns the index of a point we pass the latitude and longitude, -1 if no item

property **Point**[index:integer]:TECMapRoutePathPoint; default;

Table for access points is the default property so you can access it directly by **map.Routes [idroute]. Path [idPoint]** instead of **map.Routes [idroute]. Path.Point [idPoint]**

TECMapRoutePathPoint

Class manager point of the road, it gives you access to properties

property **Altitude** : double;

Read-only property that returns the height in meters from the point

property **Instructions** : string;

Read-only property that returns the instructions in **HTML**, associated with the point

property **Latitude** : double;

Read-only property that returns the latitude of the point

property **Longitude** : double;

Read-only property that returns the longitude of

property **Distance** : longint;

Read-only property that returns the distance in meters

property **Text** : string;

Read-only property that returns the instructions in text format

AddPolylineFromRoute

TECMap has a feature that allows you to add a [polyline](#) from the index of a road

```
// Delphi map component EMap  
  
// set version 3.7 for google api  
map.apiVersion := '3.7';
```

Cela peut-être utile pour "**figer**" une route, en effet lors de la sauvegarde d'une route seul les points de départ et d'arrivée sont sauves, lors de la lecture un recalcul de la route sera effectué.

Polylines can [calculate batch altitude](#) points while for the roads it must be done step by step, **DemoRoute** uses AddPolylineFromRoute to calculate the elevation.

Utility functions

```
function TECMap.GetRoutePathFrom(const dLatLngs: array of double;const params): TECMapRoutePath;
```

```
procedure TECMap.GetASyncRoutePathFrom(const dLatLngs: array of double;const params: string);
```

```
function TECMap.GetRoutePathByAdress(const StartAdress, EndAdress: string;const params: string):  
TECMapRoutePath;
```

```
procedure TECMap.GetASyncRoutePathByAdress(const StartAdress, EndAdress: string;const params: string);
```

These four routines to get a road without made of path information, you have two asynchronous procedures that will run in the background and raise the OnRoutePath event when the data are available.

If you do not release the [TECMapRoutePath](#) obtained, it will be automatically during the destruction of TECMap

*See open.mapquestapi.com/directions/ for the parameter **Params***

You can create a polyline from a TECMapRoutePath

```

// Delphi map component EMap

// ge data road between Tarbes and Lourdes via Aureilhan
routePath := map.GetRoutePathByAdress('Tarbes',
'Aureilhan|Lourdes');
// draw polyline from routePath
if routePath<>nil then
begin

id := map.polylines.addFromRoutePath(routePath);
// see the entire route
map.polylines[id].fitBounds;

routePath.free;
end;

```

function TECMap.**DistanceFrom**(const LatStart, LngStart, LatEnd, LngEnd: double): double;

Returns the distance in KM between two geographical points, uses the [haversine formula](#)

function TECMap.**DistanceRouteByAdress**(const StartAdress, EndAdress: string; const params: string = ''): double;

Returns the distance in KM between two addresses passing by road

function TECMap.**DistanceRouteFrom**(const dLatLngs: array of double; const params: string = ''): double;

Returns the distance in KM between two geographic points through the road

See open.mapquestapi.com/directions/ for the parameter **Params** of the functions *DistanceRouteXXX*

function TECMap.**AddRouteByAdress**(const Adresses: TStrings): integer;

Adds a route, **Adresses** contains the addresses of passages.

```

// Delphi map component EMap

// find the distance in km by road between Tarbes and

```



```
Lourdes via Aureilhan
dKm := map.DistanceRouteByAdress('Tarbes',
'Aureilhan|Lourdes');

Adresses := TStringList.create;
try

    Adresses.add('Tarbes');
    Adresses.add('Aureilhan');
    Adresses.add('Séméac');
    Adresses.add('Lourdes');

    // create route Tarbes/Lourdes via Aureilhan and Séméac
    map.AddRouteByAdress(Adresses);

finally
    Adresses.free;
end;
```

DemoRoute

The program DemoRoute shows you how to operate the routes

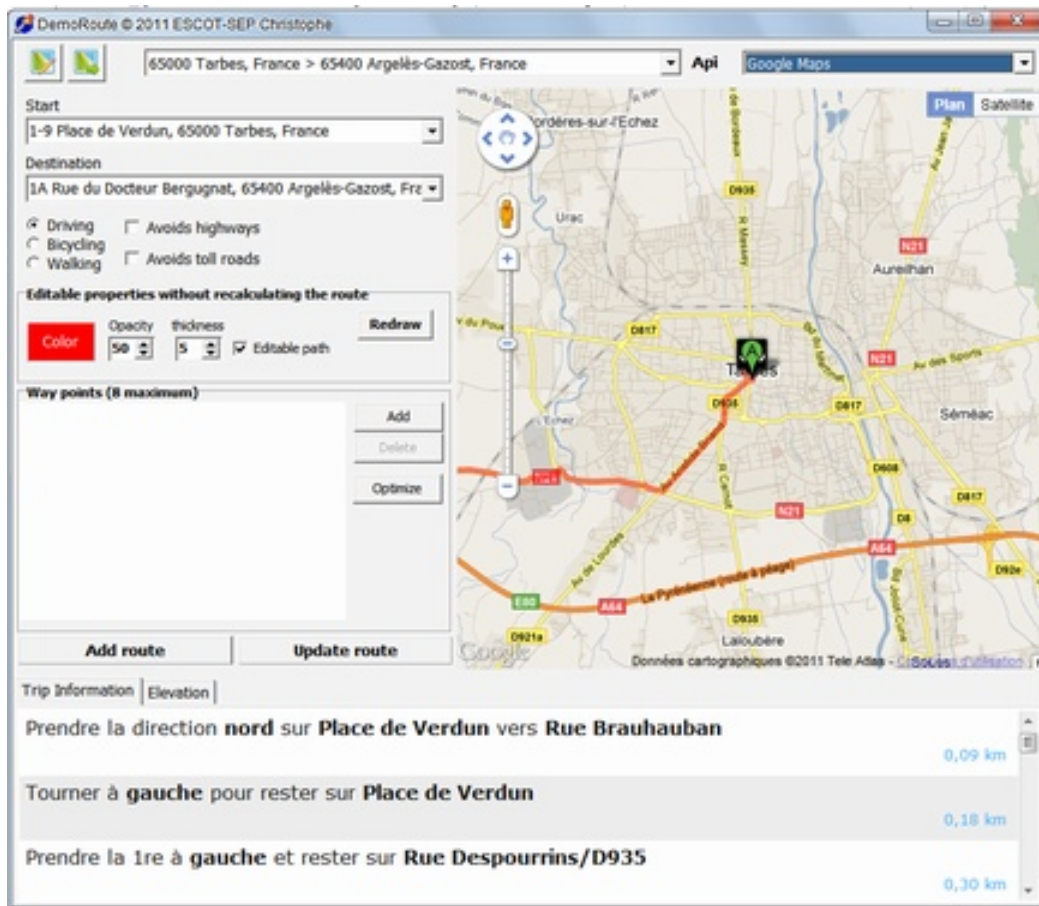


Fig. 37 DemoRoute - Instructions

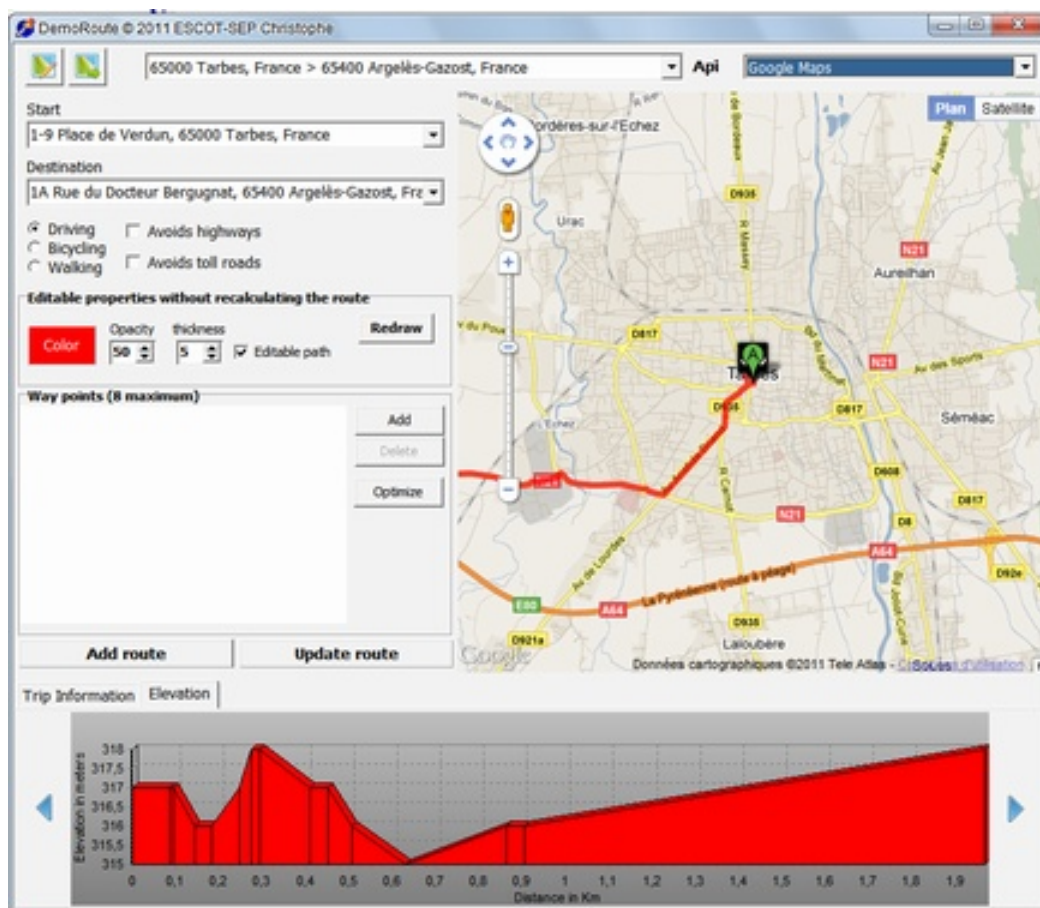


Fig. 38 DemoRoute - Altitudes

BicyclingLayer

Shows bike paths only available with api Google

```
var wn:TECCesiumShapeInfoWindow;

Wn := cmap.AddInfoWindow(10,10) ;
Wn.content := 'You can enter <b>html</b> content <br>text

and <i>image</i> '
// you can style with css
Wn.Style := 'here css';
// use color for color background
Wn.Color := clWhite;

Wn.CloseButton := true;
```

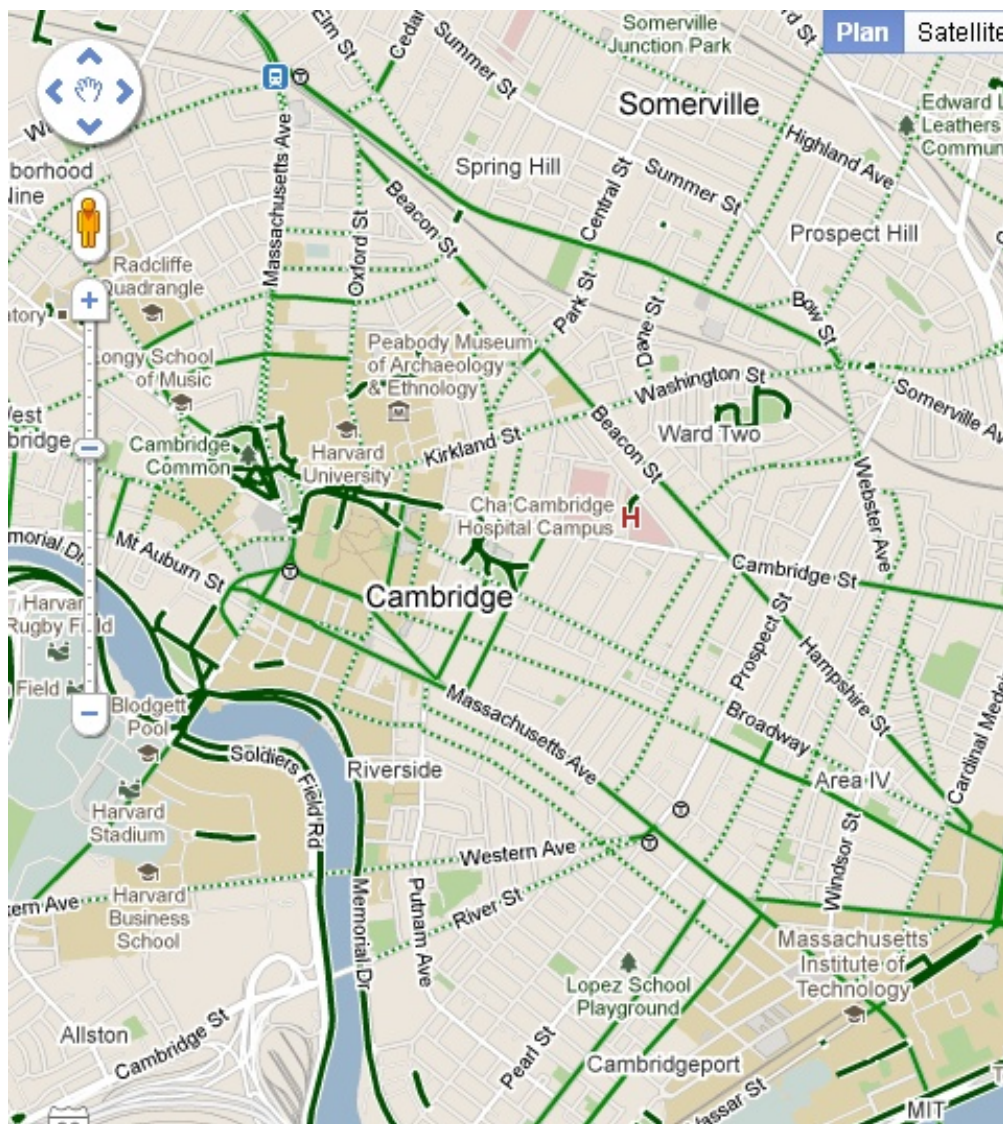


Fig. 39 BicyclingLayer

For the moment the layer is not particularly well developed in Europe, you can replace it with a card that OpenMapStreet CycleMap to the advantage of operating under all apis.

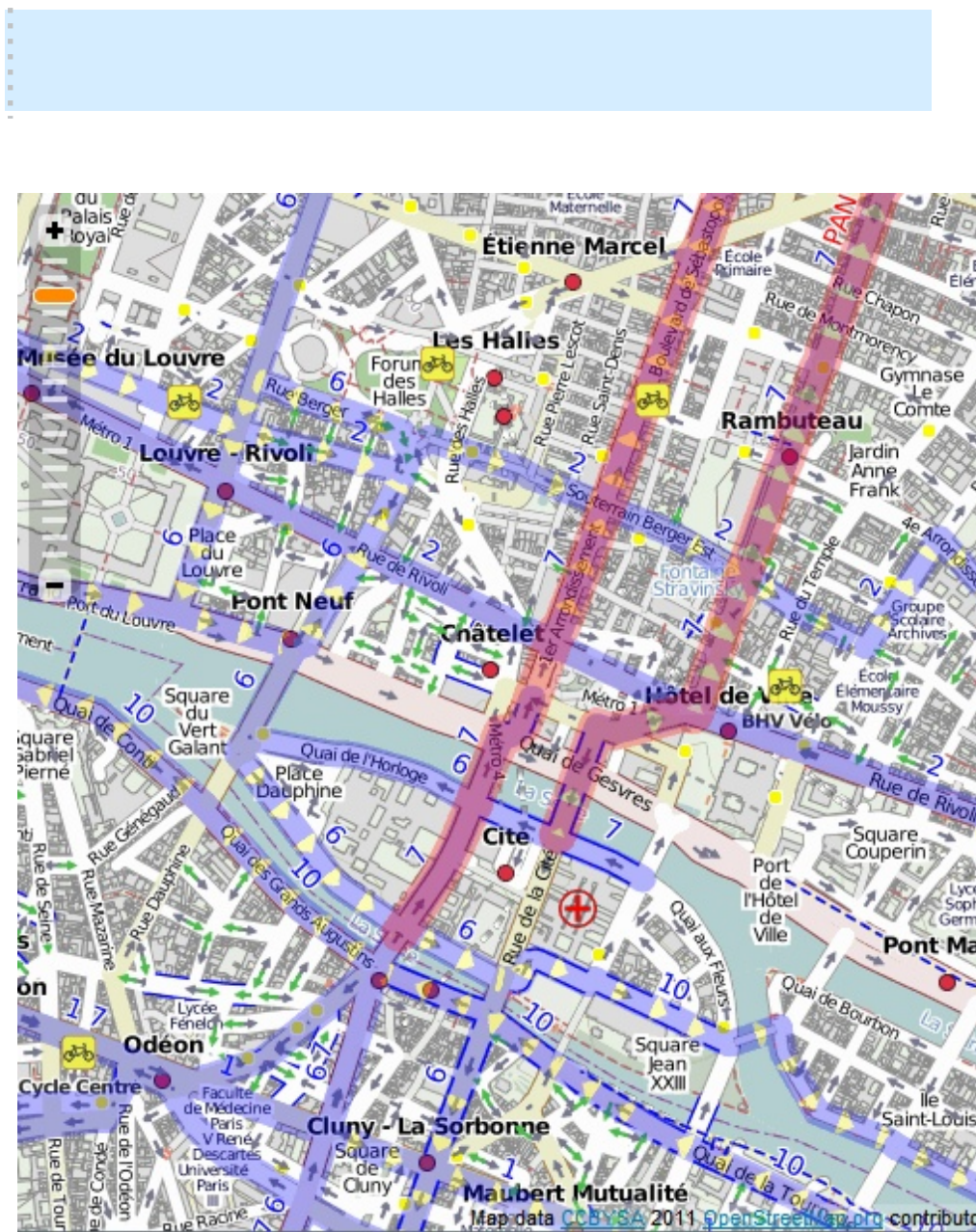


Fig. 40 OpenMapStreet CycleMap

TrafficLayer

Displays the status of traffic does not seem enabled on all regions, only available with api Google

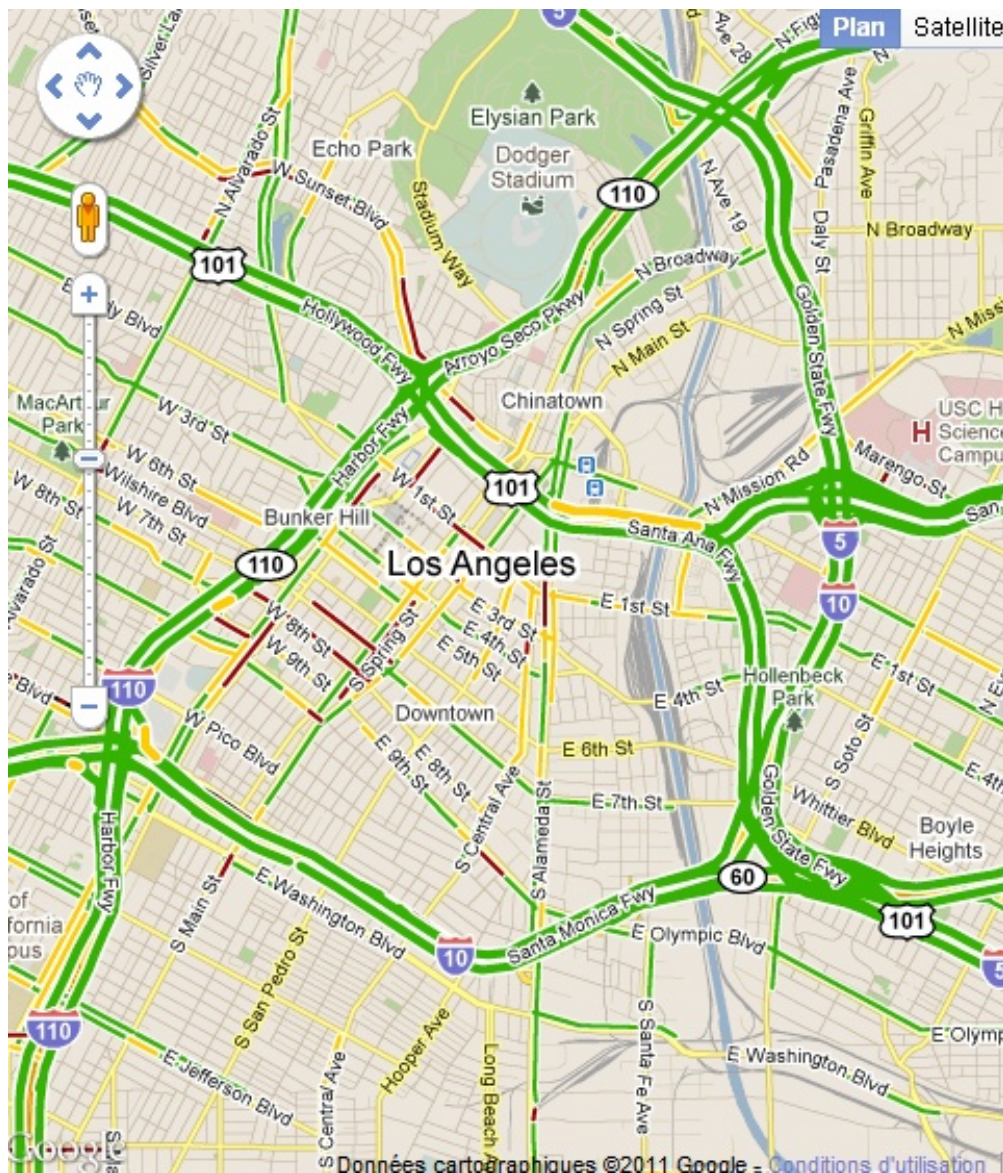
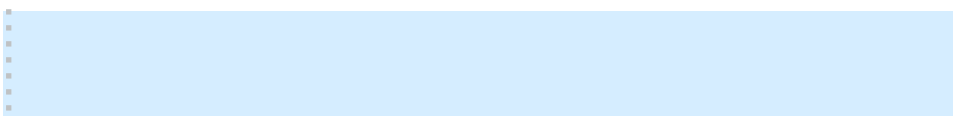


Fig. 41 TrafficLayer

TrafficArea

The procedure **TrafficArea**(const dLatSW,dLngSW,dLatNE,dLngNE:double) determines the incidents taking place in a given area.

You must you connect on the **OnTraffic** of component event to use the response



Properties **TrafficIconLowImpact**, **TrafficIconMinor**, **TrafficIconModerate**, **TrafficIconSerious** and **TrafficIconRoadClosed** allow you to use your own icons.

Fig. 42 Show TrafficArea

TrafficArea uses services from [Bing Traffic](#), you need to [get your own key Bing](#) to use this feature.

FusionTablesLayer

Displays data from a [table Google Fusion](#) only available with api Google

You manage your layers **FusionTablesLayers** through the property type **TECMapFusionTablesLayers**

TECMapFusionTablesLayers

See the [documentation on google FusionTablesLayer](#)

function **Add**(const Column,Id,Style,Clause:string):integer;

Adds a layer and returns its index

Column is the column of the table corresponds to **select** the google doc

Id is the identifier of the table is **from** in the google doc

Style the style is applied to layer

Clause is the SQL query that lets you filter your data is **WHERE** in the google doc

procedure **Clear**;

Clears all layers

function **Count**:integer;

Returns the number of layers

procedure **Delete**(index:integer);

Removes the layer which we pass the index

property **FusionTablesLayer**[index:integer]: TECMapFusionTablesLayer

Table that allows access to layers, it is the default property so you can access it directly by **map.FusionTablesLayer** **[index]** instead of **map.FusionTablesLayers.FusionTablesLayer [index]**

property **ToTxt**: string ;

Property **read / write** which gives access to the list in text format FusionTablesLayer.

*In writing there is an addition and not a replacement, if you do not want to keep the old values before making a **Clear** adding*

TECMapFusionTablesLayer

Controlling a class FusionTablesLayer

procedure **update**;

Update properties, you must call to reflect the changes in properties

property **TableId** : string

identifier of the table is *from* in the google doc

property **LocationColumn** : string

column of the table corresponds to *select* in the google doc

property **Clause** : string

SQL query that lets you filter your data, corresponds to *Where* in google doc

property **Style** : string

applicable to the layer style

property **Heat** : boolean

corresponds to heatmap in the google doc

property **SuppressInfoWindow** : boolean

Enable or not displaying a InfoWindow when clicking on an element of layer

property **Visible** : boolean

Displays whether the layer

property **ToTxt** : string

Property *read / write* accesses the layer in a text format.

OnFusionTablesLayerClick

TECMap has an event to intercept a click on an element of layer

procedure **OnFusionTablesLayerClick**(sender: TObject;const
idLayer:integer;const htmlInfoWindow:string;dLatitude,dLongitude:double);

idLayer number of layer in the list FusionTablesLayers

htmlInfoWindow a string containing the HTML content of the tooltip of the item, if available

dLatitude,dLongitude coordinates of the point clicked

KmlLayer

You can import KML / KMZ into your cards and display overlay, not available with Leaflet.

*KMZ files are not supported by CloudMade by cons
GeoXml it supports the format (extension. Xml)*

Layers Kml have properties and methods

Id : integer

Index layer, read-only

Url : string

Url kml file, read-only

Visible : boolean

Layer of visibility, read / write

ToTxt : string

property read / write to export / import format text all the elements

procedure **Show**;

Displays the default zone of the layer if it was specified

OnKmlLayerClick

TECMap has an event to intercept a click on an element of layer.

CloudMade not responding to this event

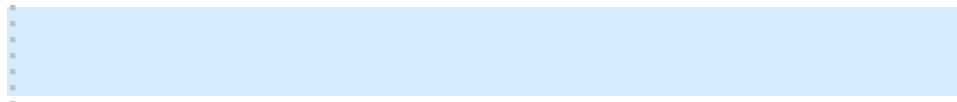
procedure **OnKmlLayerClick**(sender: TObject;const
idKmlLayer:integer;const htmlDescription,htmlInfoWindow:string;dLatitude,dLongitude:double);

idKmlLayer number of layer in the list KmlLayers

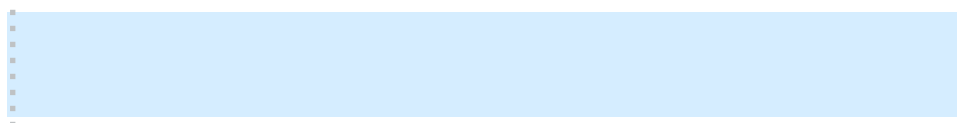
htmlDescription a string containing the HTML content of the element, if there

htmlInfoWindow a string containing the HTML content of the tooltip of the item, if available

dLatitude,dLongitude coordinates of the point clicked

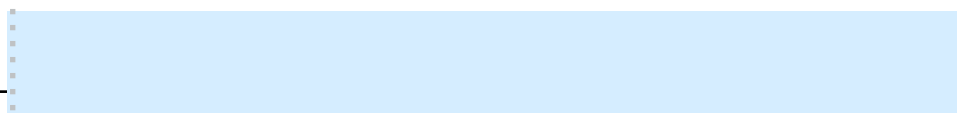


TECMap exports the majority of its data, including kml layers, in KML.



WeatherLayer

Available only with the Google api



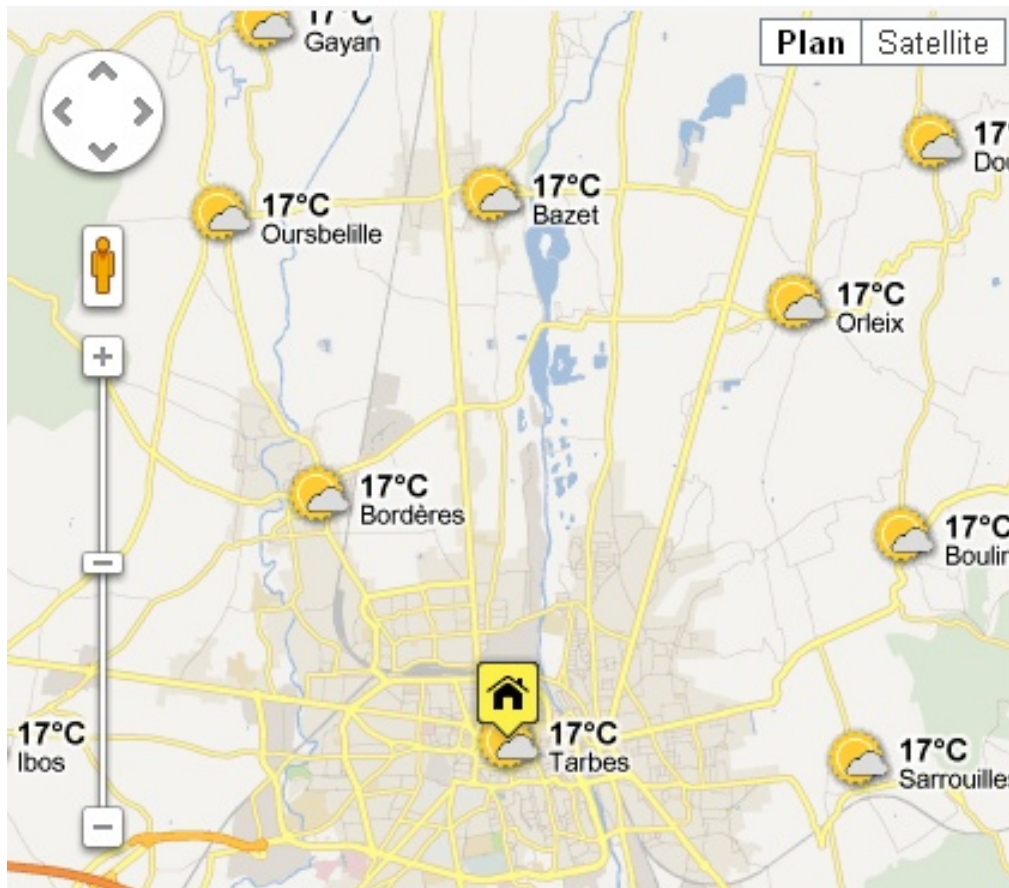


Fig. 43 WeatherLayer

OnWeatherLayerClick

TECMap has an event to intercept a click on an element of layer.

procedure **OnWeatherLayerClick**(sender: TObject;const htmlInfoWindow:string;dLatitude,dLongitude:double);

htmlInfoWindow a string containing the HTML content of the tooltip of the item, if available

dLatitude,dLongitude coordinates of the point clicked

DemoLayer

DemoLayer program allows you to see how to use these different layers and **Panoramio** , which is also a layer

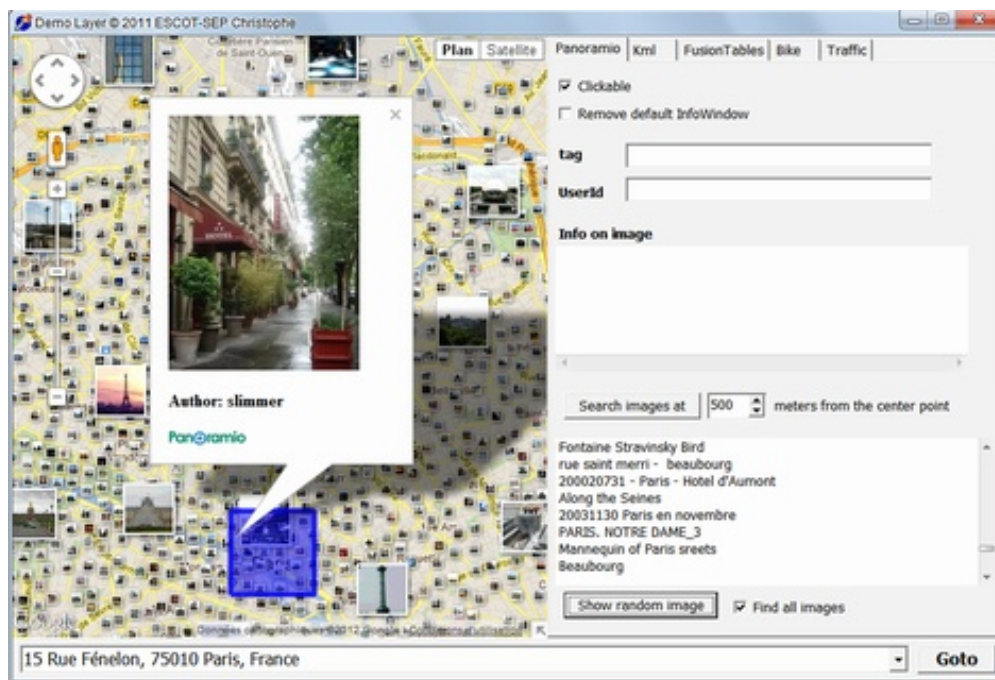


Fig. 44 DemoLayer

Pois (point of interest) are equivalent to the [markers](#), the difference is that they are supported natively by Delphi, they are so light and fast, you can incorporate several thousands without worries.

You have 4 predefined shapes (Ellipse, rectangle, triangle and star) but you can also take in charge their design.

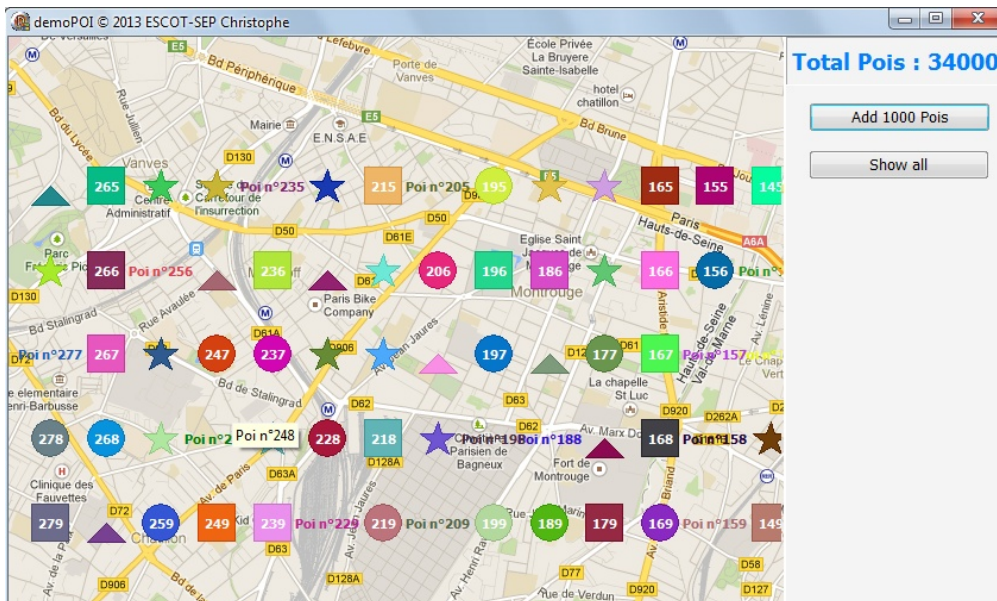


Fig. 45 DemoPOI

Does not work on Chromium !

POIs are managed by an **TECMapPois** list accessible through the Pois TECMap property

TECMapPois

This list has the following methods and properties :

function **Add**(const dLatitude,dLongitude:double):integer;

Adds a POI to the point Latitude, Longitude and returns its index in the list.

```

// Delphi map component TMap
// add POI at center of map
id := map.Pois.add(map.latitude,map.longitude);
// set caption to this POI
map.Pois[id].caption := 'my first POI!';

```

function **ByLatLng**(const dLatitude,dLongitude:double):TECMapPoi;

Returns the POI positioned at Latitude, Longitude

procedure **Clear**;

Clears all POIs

procedure **Delete**(index:integer);

Removes the POI which you pass the index

property **Count**:integer;

Returns the number of POI in the list

procedure **fitBounds**

Adjusts the zoom of the map to display the set of POIs

property **Poi[index:integer]**:TECMapPOI;

Table allowing access to POIs, it is the default property so you can access it directly by **map.Pois[index]** Instead of **map.Pois.Poi[index]**

property **ToKml** : string;

Property **read-only** that returns a string in the [Kml format](#) containing the list of POIs

property **ToTxt** : string;

Property **read/write** which gives access to the list of POIs in a text format.

*In writing it is an addition, not a replacement, If you do not want to keep the old values made a **Clear** before adding*

property **ShowHint** : boolean

Indicates whether to display the caption of a POI in a bubble when hovering the mouse over

property **OnOwnerDrawPOI** : TOnOwnerDrawPOI

Specify a procedure to support the design of type POI **poiOwnerDraw**

```
// Delphi map component EMap

map.Pois.OnOwnerDraw := doOwnerDrawPOI;
...

// owner draw poi, here transparency text
procedure TFormPoi.doOwnerDrawPOI(const canvas:TCanvas;var
Rect:TRect;item:TECMapPoi) ;
begin

    canvas.brush.Style := bsClear;

    if item.Hover then
        canvas.font.color := item.HoverColor
    else
        canvas.font.color := item.color;

    canvas.font.Style := [fsBold];

    item.Width := canvas.TextWidth(item.caption) ;
    item.height := canvas.TextHeight(item.caption) ;

    item.x := item.x - (item.Width div 2);
    item.y := item.y - (item.height div 2);
```



```
        canvas.TextOut(item.x,item.y,item.caption);  
  
    end;
```

property **DragPoi** : TECMapPoi

If a POI is moved with the mouse this the reference property

TECMapPOI

This class manages a POI, it has the following methods and properties :

procedure **setPosition**(const dLatitude,dLongitude:double);

Move the POI in Latitude, Longitude, raises the **OnOverlayMove** event

procedure **PanTo**;

Moves the map so that its centre coincides with the POI, raises the **OnMapMove** event

property **Caption**: string

property **Color**: TColor

Color of the POI

property **HoverColor**: TColor

The mouse-over color

property **Latitude**: double

Read-only property, use SetPosition

property **Longitude**: double

Read-only property, use `SetPosition`

property **X** : integer

Position horizontal pixel, calculated automatically depending on the latitude

property **Y** : integer

Vertical position in pixels, calculated automatically depending on the longitude

property **Width** : integer

Width in pixels

property **height** : integer

Height in pixels

property **Draggable** : boolean

Indicates if the POI is draggable mouse

property **Shape** : `TPOIShape`

poiEllipse, **poiRectangle**, **poiTriangle**, **poiStar** et **poiOwnerDraw**

property **Hover** : boolean

Indicates if the POI is overflowed by the mouse

property **OnBeforeDrawPOI** : `TOnOwnerDrawPOI`

Specify a procedure to draw on top of type POI **poiEllipse**, **poiRectangle**, **poiTriangle** et **poiStar**

Example, written his number on a POI

```
// Delphi map component EMap  
  
i := map.pois.add(map.Latitude,map.Longitude);  
map.pois[i].Shape := poiStar;
```

```

map.pois[i].width := 25;
map.pois[i].height := 25;

map.pois[i].OnBeforeDrawPOI := doNumDrawPOI;

// draw poi id
procedure TFormDemoECMap.doNumDrawPOI(const
    canvas:TCanvas;var r:TRect;item:TECMapPoi) ;
var x,y,w,h : integer;
    s : string;
begin

    canvas.font.style := [fsBold];

    s := inttostr(item.id);

    w := canvas.TextWidth(s) ;
    h := canvas.TextHeight(s) ;

    x := 1+((r.Left + r.Right) - w) DIV 2 ;
    y := 1+((r.Top + r.Bottom) - h) DIV 2 ;

    canvas.brush.Style := bsClear;

    canvas.font.color := clWhite;

    canvas.TextRect(r,x,y,s);
end;

```

Évenements

Pois meet the same [events as overlays](#) (**OnOverlayXXX**), their TOverlayType is **ovPoi**

Groups enable you to manage a set of elements, you can load them, save them, display them, hide them, move them to a single command.

```
// Delphi map component EMap
// show group 'group-one'

map.Groups['group-one'].visible := true;

// move group
map.Groups['group-one'].setCenter(43.232951,0.078082);
```

The elements that you can group are the [markers](#), [polylines](#), [polygones](#), [circles](#), [rectangles](#), [groundoverlays](#) and [labels](#)

the property **Groups[]** Returns a **TECMapItemGroup**, Access **groups** will create a group if it does not exist.

The groups name is not case sensitive, map.Groups['group'] is equivalent to map.Groups['Group']

```
// Delphi map component EMap
var i:integer;

GLines : TECMapItemGroup;
begin

GLines := map.groups['GLines'];
GLines.BeginUpdate;
GLines.clear;

for i:=0 to map.polylines.count - 1 do
    GLines.add(map.Polylines[i]);

for i:=0 to map.polygones.count - 1 do
    GLines.add(map.Polygones[i]);

GLines.EndUpdate;
```

TECMapItemGroup

procedure **BeginUpdate**;

Procedure to use before multiple changes to the group to optimize speed

procedure **EndUpdate**;

Procedure to use, coupled with **BeginUpdate**, before multiple changes to the group to optimize speed

procedure **Clear**;

Empty the Group of its elements, but they are not deleted from the map

procedure **Add**(const item:TECMapItem);

Adds an element group

procedure **AddBounds**(const SouthWestLat,SouthWestLng,NorthEastLat,NorthEastLng:double);

Adds all of the TECMapItem in the area bounded by the low point (South West) and high (North East).

procedure **Remove**(const item:TECMapItem);

Removes a group element, the element is not removed from the map

procedure **Delete**;

Removes all the elements of the Group and of the map, the elements no longer exist

If you delete a group by `map.Groups['Group'].Free` the elements are not removed from the map

procedure **Show**;

Makes visible all of the elements of the Group

procedure **Hide**;

Cache all of the elements of the Group

function **SaveToFile**(const filename:string):boolean;

Saves the Group to a text file.

Use the .gpx , .kml and .json to specify a format, otherwise it is the internal format of EMap that will be used

procedure **LoadFromFile**(const value: string);

Load the contents of a text file in the internal format of EMap

Use the .gpx , .kml and .json to specify a format, otherwise it is the internal format of EMap that will be used

LoadFromFile can download files on internet

function **Contains**(const lat, Lng: double): boolean;

Indicates whether the point located in the Lat, Lng is situated in the area of the Group

procedure **setCenter**(const lat,Lng:double);

Moves all of the items in the group so that its centre is located at Lat, Lng

procedure **PanToBounds**;

Dragging the map so the group is visible.

procedure **fitBounds**;

Adjusts the view so that all of the elements of the group is visible

property **Visible**:boolean ;

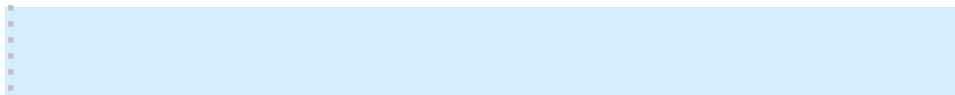
This property is used by **Show / Hide**

property **MinZoom**:byte ;

Minimum zoom to make the group appear, leave 0 to skip the zoom level

property **MaxZoom**:byte ;

Maximum zoom to make the group appear, leave 0 to skip the zoom level



property **OnChange** : TNotifyEvent

Raised when the scope of the group changes or the addition of an element

property **SouthWestLat**: double ;

Specifies the latitude of the left lower corner of the area that includes the Group

property **SouthWestLng**: double ;

Specifies the longitude of the left lower corner of the area that includes the Group

property **NorthEastLat**: double ;

Indicates the latitude of the top corner of the area that includes the Group

property **NorthEastLng**: double ;

Indicates longitude from the top corner of the area that includes the Group

property **CenterLat** : double ;

Specifies the latitude of the center of the group, use **setCentre** to modify

property **CenterLng** : double ;

Specifies the longitude of the center of the group, use **setCentre** to modify

property **Name** : string;

property **toGPX** : string;

Property **read/write** which gives access to the data of the card in GPX format.

property **ToKml** : string;

Property in **read / write** which returns the [overlays](#) (all except the [Labels](#)) in [Kml format](#)

property **ToGeoJSON** : string

Property in **read / write** which returns / adds [overlays](#) (just the markers, the polylines, polygons, circles, and rectangles) in the [GeoJSON format](#)

property **ItemsToTxt** : string;

Property in **read / write** which returns / adds [overlays](#) in internal text of TECMAP

property **Count** : integer

Returns the number of item in the Group

property **Item**[index:integer]:[TECMapItem](#)

Returns an item from the Group

Property type TECMapStreetView **StreetView** enables you to manage the display StreetView

Not available CloudMade



Fig. 46 Vue StreetView

TECMapStreetView

It gives access to properties and methods

procedure **SetPosition**(const dLatitude,dLongitude:double);

Change the position of the point of view

Trigger events **OnStreetViewPosition** and **OnStreetViewAvailable**

procedure **ReDraw**;

Redraw the view to consider options other than those relating to the camera and position

property **Visible** : boolean;

Property *read / write* for the show or not to StreetView

Raises the event **OnStreetViewVisible**

property **Adress** : string;

Property *read* returns the address of the place

property **Available** : boolean;

Read-to determine whether StreetView is available at this location

property **Latitude** : double;

Property *read / write* to set the latitude, see **SetPosition**

property **Longitude**: double;

Property *read / write* to set the longitude, see **SetPosition**

property **Heading** : integer;

Property *read / write* to define the viewing direction of camera, 0° = **North**, 90° = **East**,
180° = **South** et 270° = **West**

Raises the event **OnStreetViewPOV**

property **Pitch** : integer;

Property *read / write* to set the vertical rotation of the camera, to 90° at -90°

Raises the event **OnStreetViewPOV**

property **Zoom** : integer;

Property *read / write* to set the zoom level

Raises the event **OnStreetViewPOV**

property **NavigationControl** : boolean;

Property *read / write* or not to display the control bar

property **NavigationPosition** : TControlPosition;

Property *read / write* sets the position of the control bar

Possible values are :

- cpTopLeft
- cpTopCenter
- cpTopRight,
- cpRightTop
- cpRightCenter
- cpRightBottom
- cpBottomRight
- cpBottomCenter
- cpBottomLeft
- cpLeftBottom
- cpLeftCenter
- cpLeftTop

property **NavigationStyle** : string;

Property *read / write* to style the control bar

You have the choice between :

- DEFAULT

- ANDROID
- LARGE
- SMALL

property **AdrControl** : boolean;

Property **read / write** or not to display the address panel

property **AdrPosition** : TControlPosition;

Property **read / write** sets the position of the address, see NavigationPosition for values

property **AdrCss** : string;

Property **read / write** to assign a CSS style at

property **CloseBtnVisible** : boolean;

Property **read / write** or not to display the close button to view StreetView

property **LinkVisible** : boolean;

Property **read / write** or not to display directional links

property **ToTxt** : string;

Property **read / write** which gives access to parameters StreetView in the form of a text string

```
// red color between 1.2 km and 3.8 kilometer
line := map.shapes.lines[0].Slice(1.2,3.8);
line.Color := claRed;
```

Events

OnStreetViewAvailable(sender: TObject; const bVisible: Boolean);

Triggered when the availability of sight StreetView exchange **bVisible** is true if StreetView is available, false otherwise

OnStreetViewPosition(sender: TObject; const dLatitude, dLongitude: Double);

Triggered when changing position in latitude, longitude

OnStreetViewPOV(Sender: TObject);

Triggered when changing axis of the camera or changing zoom

OnStreetViewVisible(sender: TObject;const bVisible: Boolean);Triggered when the view becomes visible
StreetView is not visible

The closure of the X button to trigger this event with bVisible to false

Demonstration

The program **DemoMobile** shows a use of StreetView

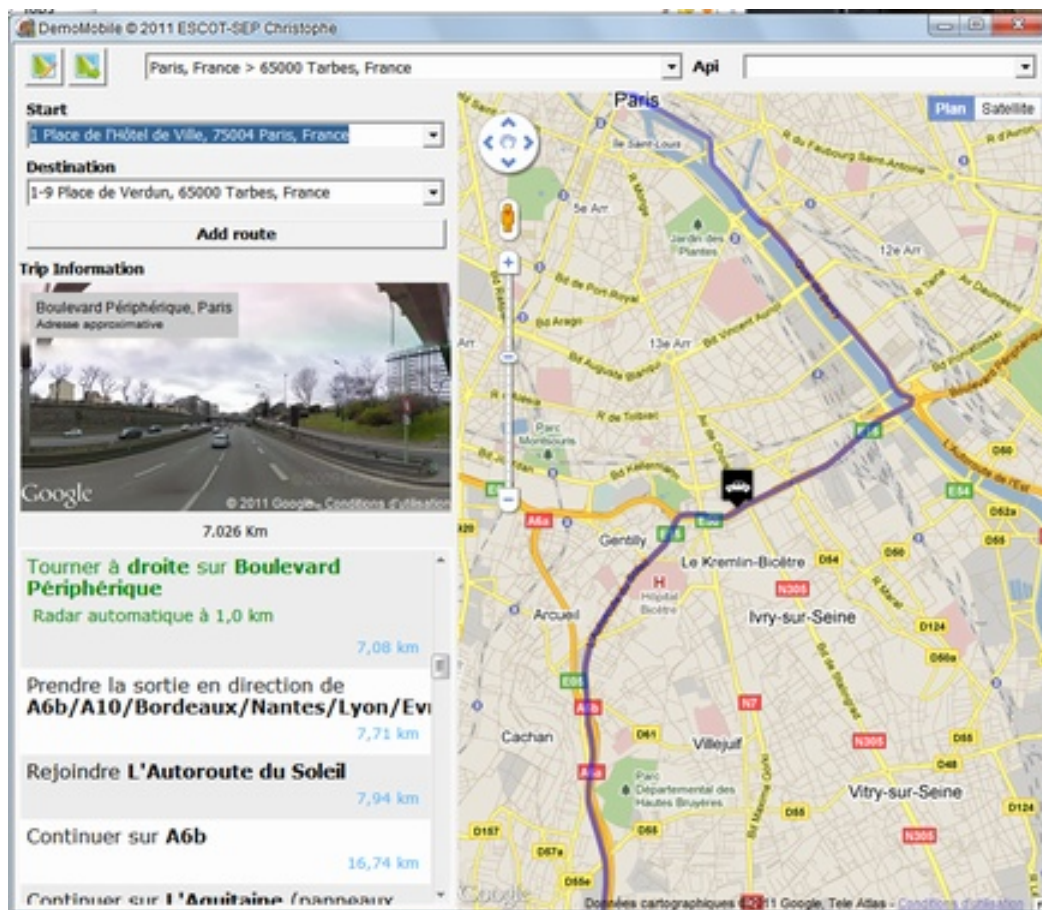


Fig. 47 A view StreetView connected to a map view

EarthView type `TECMapEarthView` property lets you manage [3D Google Earth](#) display or [cesium](#)

Google Earth is available only with the [api Google](#)

The Google Earth plugin will be abandoned by Google [in December 2015](#)

The suite `TECMap` offers independent component `TECCesiumMap` to manage your views 3D, this component is also integrated with `EarthView` to replace Google Earth.

The integration of **Cesium** in **EarthView** is partial, the logo  indicates compatibility with Cesium

*It is recommended to use directly the **Cesium** property which gives full access to the Component Manager Cesium.*

TECMapEarthView

It gives access to the following methods and properties

property **Cesium**; 

Access to the `TECCesiumMap` component that allows you maximum control over the Cesium engine

property **UseCesium**; 

Choice of the Cesium engine to handle 3D

procedure **Clear**; 

Clears data

data are erased only in the 3D view, they are always available in the standard 2D view

procedure **EnableLayer**(const sLayer:string;const bShow:boolean) 

Enable/disable layer

You can choose amongst

- LAYER_BORDERS
- LAYER_BUILDINGS
- LAYER_BUILDINGS_LOW_RESOLUTION
- LAYER_BUILDINGS_LOW_RESOLUTION.
- LAYER_ROADS
- LAYER_TERRAIN
- LAYER_TREES

Under Cesium only LAYER_TERRAIN is functional

LAYER_TERRAIN is enabled by default

```
// Delphi map component EMap  
  
// active LAYER_BUILDINGS and LAYER_TREES  
map.EarthView.EnableLayer('LAYER_BUILDINGS',true);  
map.EarthView.EnableLayer('LAYER_TREES',true);
```


procedure **Camera**(const dLatitude,dLongitude:double) 

Positioning the camera in dLatitude, dLongitude and displays the view in accordance with the properties **Altitude,Roll,Tilt** and **Heading**

procedure **LookAt**(const dLatitude,dLongitude:double) 

Viewing point in dLatitude, dLongitude and displays the view in accordance with the properties **Altitude,Range,Tilt** and **Heading**

function **getGroundAltitude**(const dLatitude,dLongitude:double):double;

Returns the height of the point located in dLatitude, dLongitude

procedure **LoadKml**(const UrlKmlKmz:string); 

Loading a remote Kml/Kmz file (for Cesium only Kml)

UrlKmlKmz URL of the file to import

The data is added , made a Clear front if you want a total replacement

procedure **PlacemarkCamera**(const id:integer)

Position the camera on the index id mark and displays the view in agreement with the properties **Altitude,Roll,Tilt** and **Heading**

procedure **PlacemarkLookAt**(const id:integer)

Viewing the index id mark and displays the view in agreement with the properties **Altitude,Range,Tilt** and **Heading**

procedure **setPlacemarkLatLngAlt**(const id:integer;const Lat,Lng,Alt:double);

Positioning the index id mark in Latitude, Longitude and Altitude

procedure **getPlacemarkLatLngAlt**(const id:integer;var Lat,Lng,Alt:double);

Get the latitude, longitude and altitude of the index id mark

procedure **FromXYToLatLngAlt** (const iX, iY: double; var dLatitude, dLongitude, altitude: double)

Gives the latitude, longitude and altitude of the point X, Y (pixels)

property **Altitude** : double 

Read/write property to determine its altitude

property **MouseAltitude** : double 

Read property indicates the altitude of the point under the mouse cursor

property **AltitudeMode**: TAltitudeMode

Read/write property to fix how is calculated the altitude

You have the choice between

- amAbsolute
- amRelativeToGround
- amClampToGround
- amRelativeToSea
- amClampToSea

property **FlyToSpeed** : double

Property **read/write** to set the speed, from 0 to 5.0

Use 6.0 for an immediate teleportage

property **Loaded** : boolean

Property **read-only** to determine if the plugin has been loaded

property **Latitude** : double 

Read/write property to determine the latitude

property **Longitude** : double 

Read/write property to determine longitude

property **Heading** : double 

Read/write property to determine the direction from the North (from 0 ° to 360 °)

property **Tilt** : double 

Read/write property to determine the angle of the view (from 0 ° to 90 ° for **LookAt** and 0 ° to 180 ° for **Camera**)

property **Range** : double

Read/write property to determine the distance to the observed point (not used by **Camera**)

property **Ready** : boolean

Property *read-only* to determine the moment when the plugin has finished its screen rendering, corresponds to the event OnEarthViewFrameEnd

property **Visible** : boolean 

Read/write to toggle the 3D view property

property **Atmosphere** : boolean

Read/write property to show or not the atmosphere

property **Grid** : boolean

Read/write property to display the lines of latitude and longitude on the globe

property **MouseNavigationEnabled** : boolean 

Read/write property to turn the mouse-move

property **NavigationControlVisibility** : TNavigationControlVisibility

Read/write for display or non-property controls, you have the choice between **ncvShow**, **ncvHide** and **ncvAuto**

```
// Delphi map component EMap
map.EarthView.NavigationControlVisibility := nvcAuto;
```

property **StatusBar** : boolean

Read/write property to display the information bar

property **ScaleLegend** : boolean

Read/write property to display scale

property **Sun** : boolean

Read/write property to display the Sun

property **OverviewMap** : boolean

Read/write to display a mini property world map

property **TerrainExaggeration** : double

Property **read/write** which is a factor of increase in altitude from 1.0 to 3.0

property **ToKml** : string; 

Property **read/write** which allows import/export [Kml format](#)

TECMap has the toKml function to export almost all of the overlays in Kml format

```
// Delphi map component EMap
// import overlays (markers, rectangles, circles,...routes)
map.EarthView.toKml := map.toKml;
```

property **ToTxt** : string;

Property **read/write** which gives access to the properties in a text format.

Google Earth under the given Kml are not taken into account

property **PlaceMark**[const index:integer]: string

Property **read/write** that determines the content of the index **id** mark

The string is in Kml format, such

```
<Placemark>
<name>Placemark from KML string</name>
<Point>
<coordinates>-122.448425,37.802907,0</coordinates>
</Point>
</Placemark>
```

property **PlacemarkCount** : integer

Returns the number of marks

property **Models** : [TECMapEarthModels](#)

List of TECMapEarthModel type models

A pattern is defined by the Kml tag **<Model>**

property **CesiumModels** : TECCesiumShapeModelList 

List of 3D models of type TECCesiumShapeModel

property **StreamPercent** : integer

Property *reading* that indicates the percentage of the frame load

property **ApiVersion** : string

Returns the version of the api Google Earth

property **PluginVersion**

Returns the version number of the 3D plugin

*To make changes to the **Altitude, Heading, Range, Roll and Tilt** properties visible you need to trigger a call to **LookAt** or **Camera***

TECMapEarthModels

This class has methods and properties

procedure **LoadKml**(const filename:string);

load a KML / KMZ defining a 3D model, triggers the event [OnEarthViewUpdateModels](#)

```

1 // Delphi map component EMap
2
3 // load 3d model
4 map.EarthView.models.LoadKml (
5   )http://earth-api-samples.googlecode.com/svn/trunk/demos/drive-simulator/
6
7
8
9
10

```

*If templates are present in the KML loaded by **TECMapEarthView.ToKml** or **TECMapEarthView.LoadKml** they will also be available in the Models list*

function **IndexOf**(const idModel:string):integer;

Returns the index of the model that we pass the ID

This is the ID property of the tag Model

procedure **Delete**(const index:integer);

Removes the index model

function **Count**:integer;

Returns the number of 3D model

procedure **Clear**;

Clears all the 3D models

property **Model**[Index:integer]:[TECMapEarthModel](#)

Table for access types, the default property so you can access it directly by

map.EarthView.Models[index] instead of **map.EarthView.Models.Model[index]**

TECMapEarthModel

Class manages a 3D model, it have methods and properties

procedure **setLatLngAlt**(const Lat,Lng,Alt:double);

Position the model in latitude, longitude, altitude

procedure **LookAt**;

Watch model

property **Id** : string ;

Property *read* that gives the identifier of model

This is the ID property of the tag Model

property **Index** : integer

Index model in the list **Models**

property **Latitude** : double

Property *read / write* to determine latitude

property **Longitude** : double

Property *read / write* for determining longitude

property **Heading** : double

Property *read / write* to determine the orientation with respect to North (0 ° to 360 °)

property **Tilt** : double

Property *read / write* for determining the angle of view (0 ° to 90 °)

property **Range** : double

Property *read / write* to determine the distance to the observed point)

property **Roll** : double

Property *read / write* to determine the rotation model

property **Altitude** : double

Property **read / write** to determine the height of the model

Guided tour

Google Earth allows you to play tours (see [la documentation de google earth api](#))

EarthView offers you the possibility to manage in a simple way.

property **Tour** : string

Read/write property to load a tour

- You can load a kml/kmz file from the internet or locally
- You can also pass a string containing the kml data

```
// load kml tour

map.EarthView.Tour := 'http://www.my.com/mytour.kml';
map.EarthView.Tour := 'c:\mytour.kml';
map.EarthView.Tour :=
; <?xml version="1.0" encoding="UTF-8"?>...<gx:Tour>...'
```

The OnLoadTour event is raised when the tour is loaded and available

procedure **PlayTour**;

Start the tour

The OnEndTour event is raised when the tour is over

procedure **PauseTour**;

Paused the tour

procedure **ResetTour**;

Positions the tour initially, waiting to be played

procedure **ExitTour**;

Exit tour

property **CurrentTimeTour** : integer;

read/write property that gives/fixed the position of the tour in time (in seconds)

property **Duration** : integer

read property that gives the time of the tour in seconds

property **StateTour** : TEarthStateTour

read property that gives the status of the tour (stNone,stPlay,stPause,stEnd)

Events

In addition to the events below you also have access to **OnMapClick**, **OnMapDbClick**, **OnMouseMove**, **OnOverlayMouseDown**, **OnOverlayMouseUp** *et* **OnOverlayMouseOut**

property **OnEarthViewInit** : TNotifyEvent;

Fired when the plugin is loaded, this happens when you switch to true
TECMapEarthView.Visible

property **OnEarthViewShow** : TNotifyEvent;

Fired when when you switch to true **TECMapEarthView.Visible**

property **OnEarthViewChange** : TNotifyEvent;

Triggered when the view changes

property **OnEarthViewFrameEnd** : TNotifyEvent;

Triggered when 3D rendering is done, this is a good place to move 3D models for
smooth animation

property **OnEarthViewUpdateModels** : TNotifyEvent;

Triggered by the addition of a model

property **OnEarthViewUpdatePlacemarks** : TNotifyEvent;

Triggered by the addition of a Placemark

property **OnEarthViewClickPlacemark**(sender: TObject;const index:integer;const KmlString:string)

Triggered by a click on a Placemark

index is the index of the mark in the list **Placemark**

KmlString the content of the mark in KML, gender

<Placemark>

<name>Placemark from KML string</name>

<Point>

<coordinates>-122.448425,37.802907,0</coordinates>

</Point>

</Placemark>

Demonstrations

Demo3D shows you how to move a particular 3D model by following a specific route.



DemoOverlay the program shows you how to transfer data to a 3D view

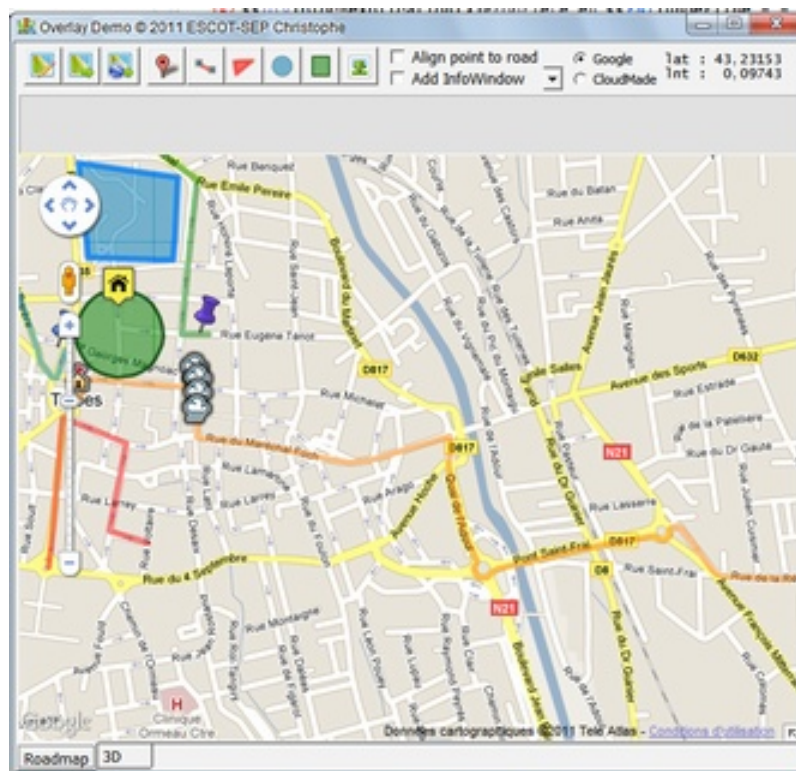
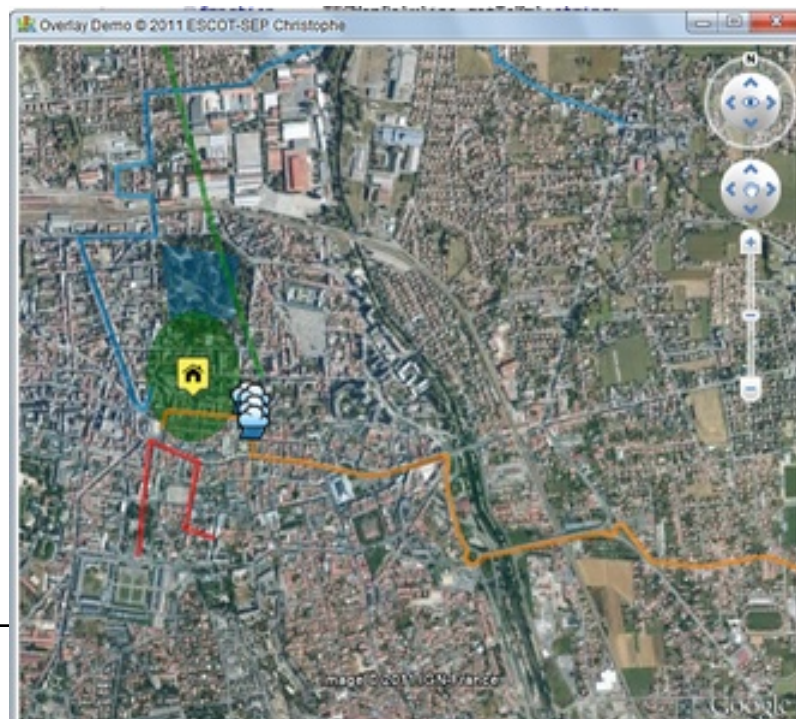


Fig. 49 DemoOverlay



Property type **PanoramioView** **TECMapPanoramioLayer** lets you view images from [Panoramio](#)

It is available only with Google Maps but an [emulation is possible](#).



Fig. 51 Enabling view Panoramio

```
// Delphi map component EMap  
  
// show panoramio layer  
map.PanoramioView.visible := true;
```

TECMapPanoramioLayer

This class has properties

property **Clickable** : boolean;

Property *read / written* or not making clickable images

Raises the event [OnPanoramioLayerClick](#)

property **SuppressInfoWindows** : boolean;

Property *read / written* or not allowing the display of the InfoWindow associated thumbnail image

property **Tag** : string;

Property *read / written* that allows filtering by keyword

property **UserId** : string;

Property *read / written* that allows filtering by UserId, identifying Panoramio poster's images.

property **Visible** : boolean;

Property *read / written* to display or not to panoramio

property **ToTxt** : string;

Property *read / write* access to **PanoramioView** which gives the form of a text string

procedure **Search**;

Launches a search for images in a rectangular area of coordinates LatSW, LngSW, LatNE, LngNE

procedure **SearchAt**(const Lat,Lng,Radius:double);

Launches search for images in a rectangular area centred on the Lat, Lng coordinates and with a Radius in Km

The image search function is available under CloudMade and Google Maps

Research is not blocking, the OnPanoramioSearch event is raised when the results are available.

Each query returns a block of 100 images maximum, to find out if other images are available you must test the **HasMore** property and boost research by **NextSearch**, the right place to do so is in the OnPanoramioSearch event.

procedure **NextSearch**;

Research for the following images, the OnPanoramioSearch event is also raised by this procedure

function **IsSearch**:boolean;

Indicates whether a search is in progress

function **Count**:integer;

Returns the number of images found

function **HtmlPhoto**(const index:integer;size:TECPanoramioImageSize):string;

Returns an html tag IMG to display an image in a specified size
(pimMini_square,pimSquare,pimThumbnail,pimSmall,pimMedium ou pimOriginal)

function **HtmlCopyright**(const index:integer):string;

Returns a html content indicating the copyright of the image

Example of using show the picture in miniature with its title and copyright in an InfoWindows

```
map.infoWindows[idInfoPanoramio].content := '<h3>' + map.PanoramioView.photo_title[1] + '</h3>'
+ map.PanoramioView.HtmlPhoto(1,pimSmall) + '<h4>Author: ' + map.PanoramioView.owner_name[1] + '</h4>'
+ map.PanoramioView.HtmlCopyright(1);
```



Fig. 52 Display an InfoWindows with a Panoramio image information

property **HasMore**:boolean

Indicates if other images are available in the research area

The following properties are associated with the found images

property **owner_id**[index:integer] : string
 property **owner_name**[index:integer] : string
 property **photo_date**[index:integer] : string
 property **photo_id**[index:integer] : string
 property **photo_lng**[index:integer] : double
 property **photo_lat**[index:integer] : double
 property **photo_title**[index:integer] : string
 property **photo_file_url**[index:integer]: string

The following properties determine the Southwest and the northeast of the area of research point

property **LatSW** : double
 property **LngSW** : double
 property **LatNE** : double
 property **LngNE** : double

OnPanoramioLayerClick

Event triggered by a click on a thumbnail Panoramio

procedure **OnPanoramioLayerClick**(sender : TObject; const Author,PhotoId,title,Url,UserId,Html:string;const dLatitude,dLongitude : double)

Author author photo

PhotoID Panoramio Photo ID

Title photo title

Url url of the photo

UserId Panoramio poster's ID photo

Html Html content of InfoWindow associated with the photo

dLatitude,dLongitude Latitude and longitude of the photo

OnPanoramioSearch

Event triggered by an image search this be with **SearchAt** or **NextSearch**

procedure OnPanoramioSearch(sender: TObject; const First, Last: Integer);

First and **Last** indicate the bounds of the images were found, indeed each search can return maximum than 100 images

DemoLayer

The **DemoLayer** program allows you to see how to use [Panoramio](#) and other types of [layers](#)

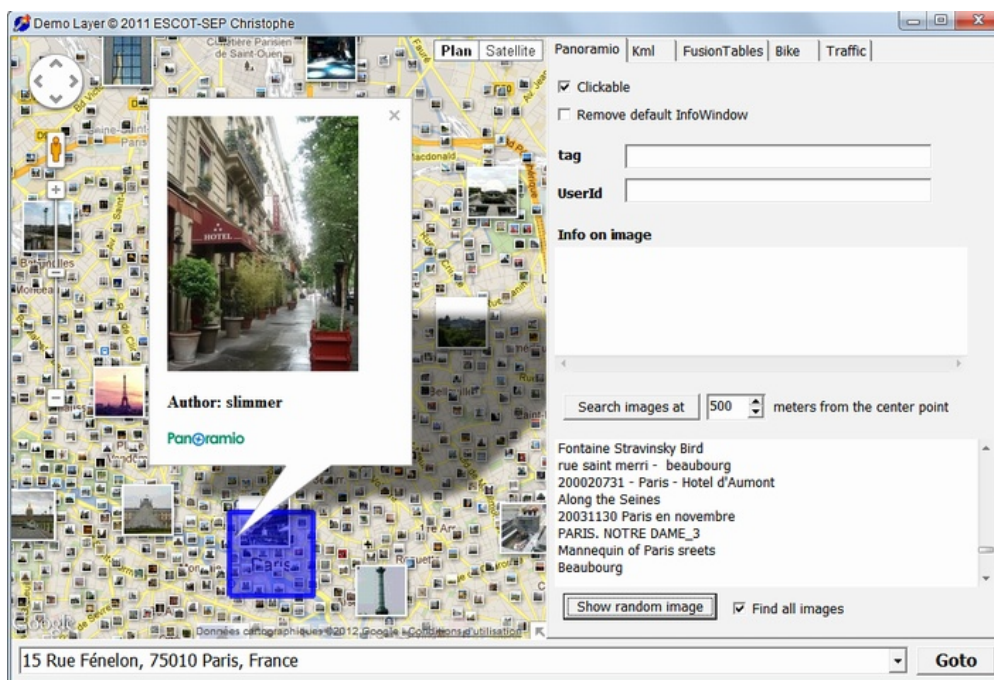


Fig. 53 DemoLayer

Property type **DistanceMatrix** [TECDistanceMatrix](#) gives you the distance and travel time for each pair of addresses start / finish.

Not available on CloudMade

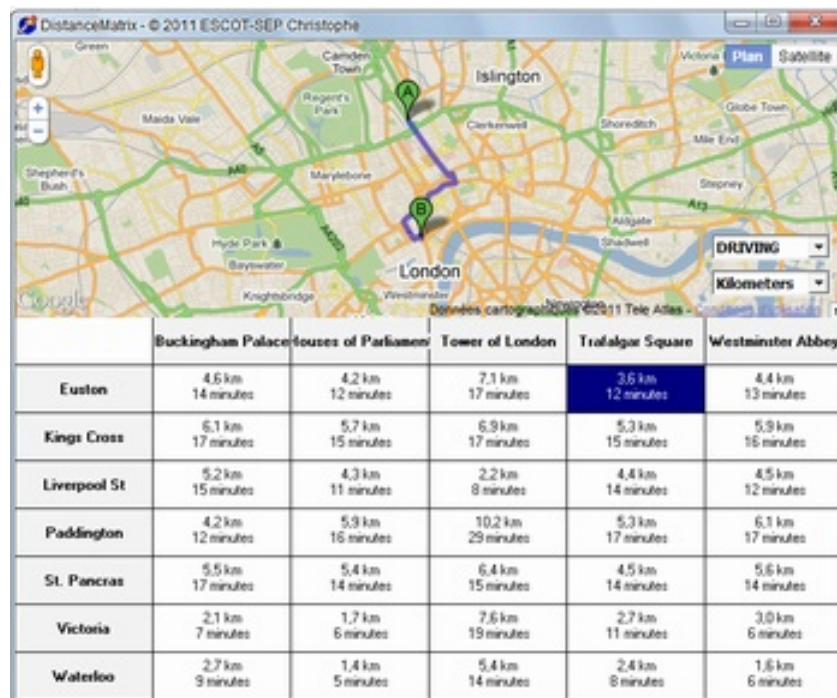


Fig. 54 DemoMatrix demonstrate the use of DistanceMatrix

The demonstration program DemoMatrix is an adaptation in Delphi of the [example of google javascript](#)

The example of Google not working in IE because there is a comma at the end arrays origins and destinations that crashes the IE javascript engine

TECDistanceMatrix

Provides access to properties and methods

function **Count**:integer;

Returns the number of couples Departure / Arrival

procedure **Update**;

Start the calculation of distance and time for each couple.

OnDistanceMatrix event is triggered when the calculation is finished

Property **Origins**:TStringList;

List of starting points

Property **Destinations**:TStringList;

List of points of arrival

property **TravelMode** : TDirectionsTravelMode;

Choosing the type of road **tmDriving** or **tmWalking**

property **UnitSystem** : TUnitSystem;

Choice of unit system, **usMetric** (km) or **usImperial** (miles)

property **AvoidHighWays** : boolean;

Avoid highways or not

property **AvoidTolls** : boolean;

Avoid tolls or not

property **Cells**[indexOrigine,indexDestination:integer]:TECDistanceMatrixItem; default;

Table for data access is the default property so you can access it directly by

map.DistanceMatrix[indexOrigine,indexDestination] instead of

map.DistanceMatrix.cells[indexOrigine,indexDestination]

indexDestination is the index of the arrival point from the list **Destinations**

indexOrigine is the index of the starting point in the list **Origins**

Returns nil if the indices are out of bounds

```
// Delphi map component EMap

// show group 'group-one' only if zoom>=14 and zoom<=16

map.Groups[ 'group-one' ].MinZoom := 14;
map.Groups[ 'group-one' ].MaxZoom := 16;

map.Groups[ 'group-one' ].Visible := true;
```

property **Item**[index:integer]:TECDistanceMatrixItem;

Item allows you to get your data sequentially, use Count to know the total number of pairs.

They are arranged in the following order

(Start1-Finish1), (Start1-Finish2),..., (Start2-Finish1), (Start2-Finish2),...

Returns nil if the index is out of bounds

TECDistanceMatrixItem

This class represents a couple Departure / Arrival, it gives you access to properties

property **Origin** : string

Starting Address

property **Destination** : string

End Address

property **Distance** : integer

Distances in meters

property **Duration** : integer

Duration in seconds

property **DistanceText** : string

Distance in the form of a text that reflects the unity tendering system chosen (**tmMetric** or **tmImperial**)

property **DurationText** : string

Duration in text

property **Status** : string

Indicates a possible error, **OK** if no error and if a match was found

OnDistanceMatrix

Event triggered after the call to **DistanceMatrix.Update** when results are available

TECNativeMap is available in **VCL** and **Firemonkey** (Windows , Mac OS X, iOS and Android) for [the same price](#) !

you access an [offline mode](#), not using the Google Maps service and other Bing Maps lets you keep control over your data.

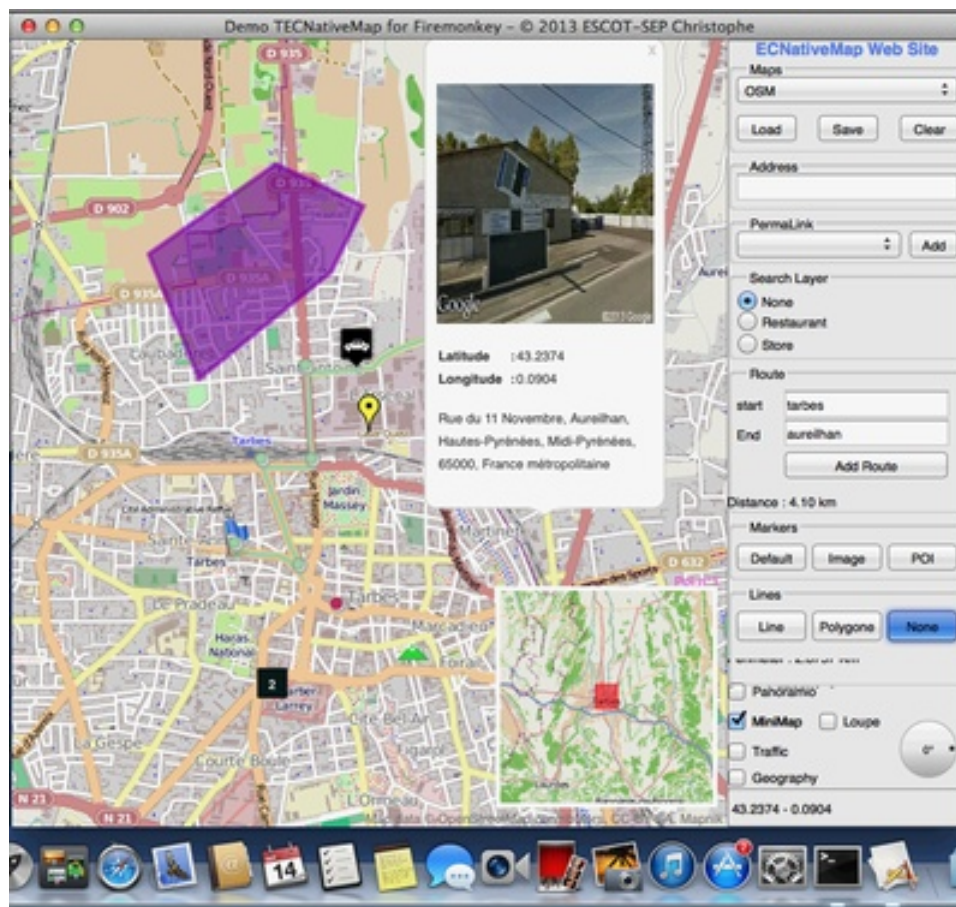


Fig. 55 ECNativeMap on Mac OS X

test a demonstration for Android

TECNativeMap allows you to use OpenStreetMap maps, MapBox, OpenCycleMap, OPNV, TomTom etc., [These maps are also available with TECMap](#). To select a map provider use the property **TileServer**

map.TileServer := tsOsm;

Bing

By filling in the **BingKey** property with [your Bing key](#) you can use Bing Maps tiles (tsBingRoad, tsBingAerial and tsBingAerialLabels)

```
map.BingKey      := YOUR_BING_KEY
map.TileServer   := tsBingRoad;
```

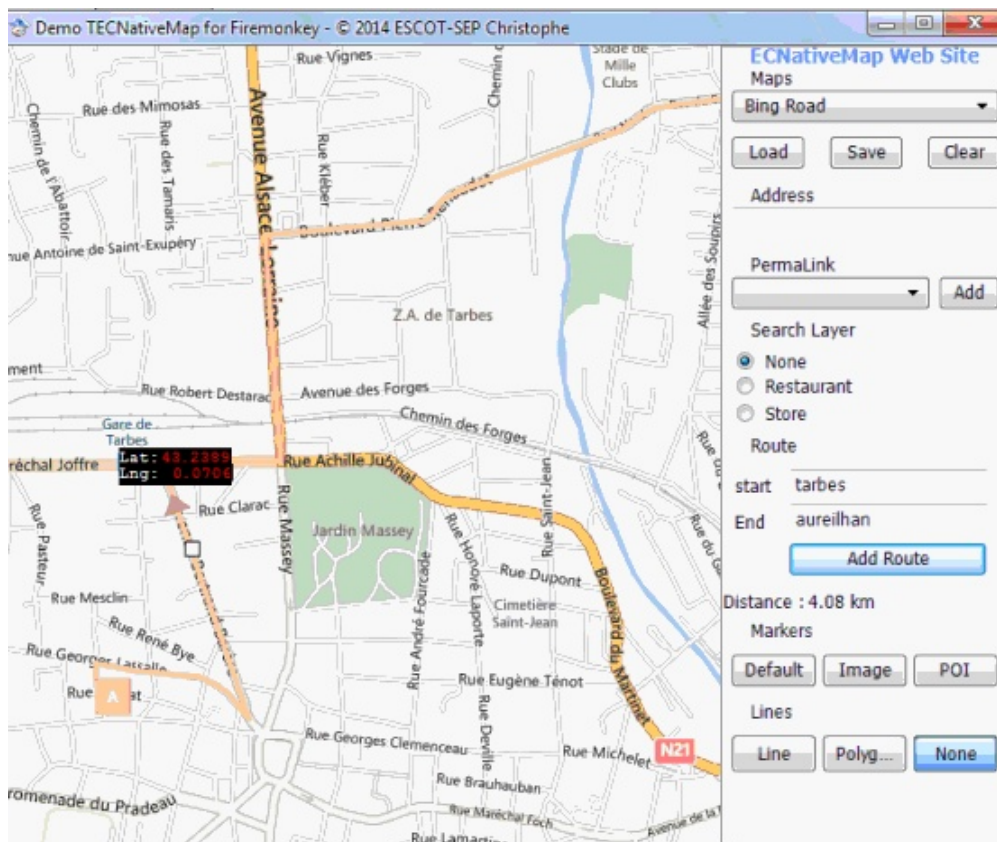


Fig. 56 Bing Maps



Here

By filling the [HereID](#) and [HereCode](#) properties You can use the [Here tiles](#) (`tsHereNormal`, `tsHereTerrain`, `tsHereSatellite`, `tsHereHybrid`, `tsHereMobile`, `tsHereHiRes`, `tsHereTransit`, `tsHereTraffic`, `tsHereFlow`, `tsHereTruck`, `tsHereTruckTransparent`)

```
map.HereID      := YOUR_HERE_ID
map.HereCode    := YOUR_HERE_CODE
map.TileServer := tsHereTraffic;
```

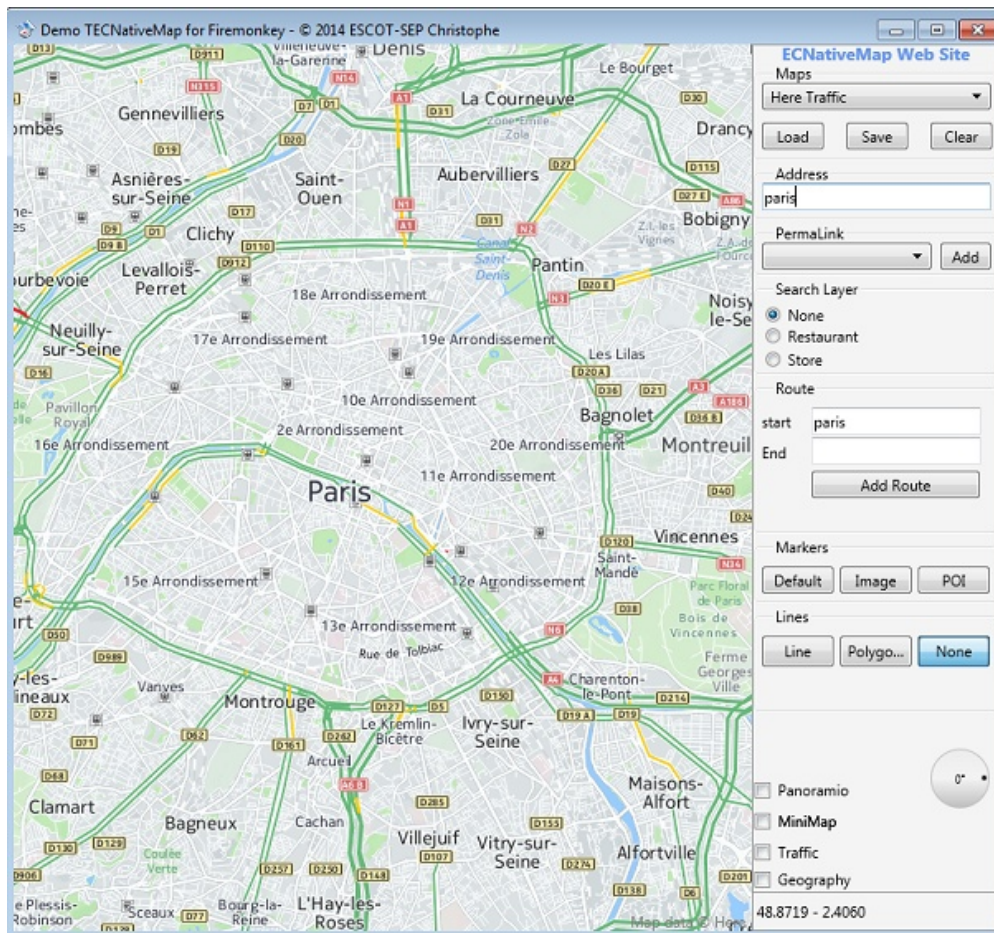


Fig. 57 Here Traffic

MapBox

To use the tiles from [MapBox](#) (`tsMapBoxSatellite`, `tsMapBoxStreets`, `tsMapBoxStreetsSatellite`, `tsMapBoxStreetsBasic`) enter your token in **MapBoxToken**.

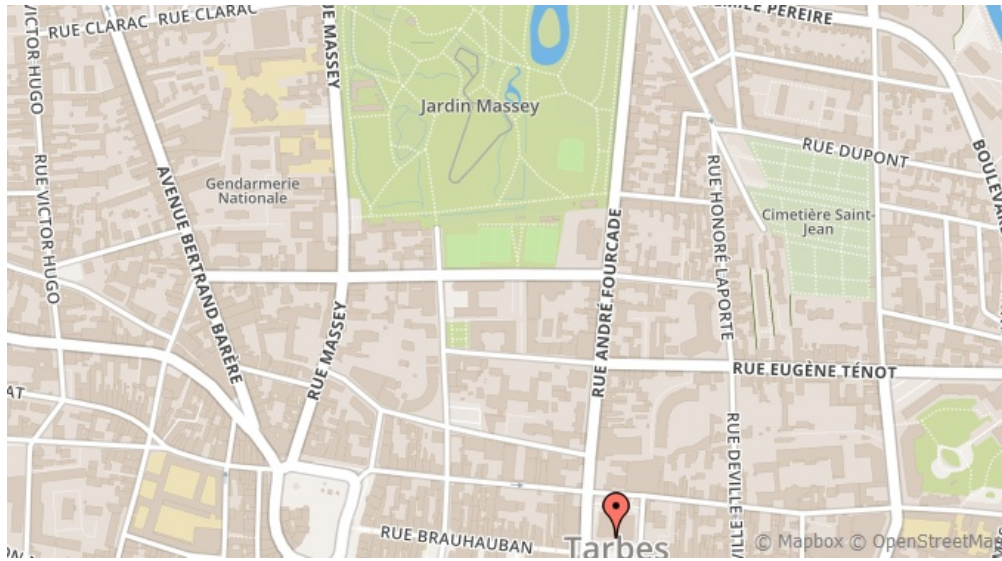


Fig. 58 MapBoxStreets

Yandex

`tsYandexNormal`, `tsYandexSatellite` et `tsYandexHybrid` are also usable (thanks alexey t.) and allow you to use tiles of Yandex.

Yandex uses the elliptical Mercator projection that is different from the default projection (spherical Mercator style google maps), this is supported by TECNativeMap so you have nothing to do.

You won't be able to use `tsYandexHybrid` with a tile base

that does not use the same projection !

TomTom

To use TomTom tiles (`tsTomTomBasic`, `tsTomTomHybrid`, `tsTomTomNight`, `tsTomTomIncident`, `tsTomTomFlow`) [you must obtain a key](#) and fill in the **TomTomKey** property of your map.

```
map.TomTomKey := your_tomtom_key  
map.TileServer := tsTomTomBasic;
```



```
map.TileServer := tsHereSatellite;  
  
// add overlay tiles for TomTom Hybrid  
map.AddOverlayTileServer(tsTomTomHybrid);  
  
// mandatory, reset api key for overlay  
map.TomTomKey := Your_TomTomKey;
```



Fig. 60 Here Satellite + TomTom Hybrid

For tsTomTomIncident and tsTomTomFlow you have the choice between 4 styles

```
var oc:TECOverlayTileLayer;
...
// add overlay tiles for incident
oc := map.AddOverlayTileServer(tsTomTomIncident);

oc.MapStyle := 's1';    // default style
oc.MapStyle := 's2';
oc.MapStyle := 's3';
oc.MapStyle := 'night';

// mandatory, reset api key for overlay
// use your key !
map.TomTomKey := map.TomTomKey;

oc := map.AddOverlayTileServer(tsTomTomflow);

oc.MapStyle := 'absolute'; // default style
oc.MapStyle := 'relative';
oc.MapStyle := 'relative-delay';
oc.MapStyle := 'reduced-sensitivity';

// mandatory, reset api key for overlay
// use your key !
map.TomTomKey := map.TomTomKey;
```

Use your own tile server

In addition to the previous maps you can use any other provider using the same format, the example below should not be used in production for legal reasons.

```
procedure TForm.FormCreate(Sender: TObject);
begin
  Map.LocalCache      :=
  ExtractFilePath(application.exename) + 'cache';
  Map.MaxZoom        := 21;
  // provider of custom tiles
  Map.TileServerInfo.Name      := 'GGL-ROAD';
  Map.TileServerInfo.GetTileFilename := GetGoogleTile;
end;
```

```

procedure TForm.GetGoogleTile(var TileFilename:string;
const x,y,z:integer);
begin
    TileFilename := format(
    'http://mt%d.googleapis.com/vt?x=%d&y=%d&z=%d',[random(4
    ),x,y,z]);
end;

```

*For licensing questions you cannot directly use the tiles from Google, but you can through the component **TECMap**.*

Overlay tiles

You can use several server of tiles to display above your base map.

```

var overlay:TECOverlayTileLayer;
...
    // overlay clouds tiles
    overlay := map.AddOverlayTiles(GetWeatherTile);

    // remove overlay

    map.RemoveOverlayTiles(overlay);

...

procedure TForm.GetWeatherTile(var TileFilename:string;
const x,y,z:integer);
begin
    TileFilename := format(
    'http://tile.openweathermap.net/map/clouds/%d/%d/%d.png',
    Char(Ord('a')+random(26)),x,y,z);
end;

```



You can still use a stream.

```

procedure TForm.getOverlayTileStream(var
  TileStream: TMemoryStream;const x, y, z: integer);
begin
  // here fill the stream with your tile

end;

...
map.AddOverlayStreamTiles(getOverlayTileStream);

```

Or directly using a transparent predefined server as [tsHereFlow](#) or [tsHereTruckTransparent](#)

```

  // mix Bing Aerial Labels and Here Flow
map.TileServer      := tsBingAerialLabels;
map.AddOverlayTileServer(tsHereFlow);

```

With VCL your overlay should use pngs, under FMX you can use the opacity property.

```

var overlay:TECOverlayTileLayer;
...
  // overlay clouds tiles
overlay := map.AddOverlayTiles(GetWeatherTile);

  // set opacity 0..1
overlay.opacity := 0.5;

```

Use **RemoveAllOverlayTiles** to delete all overlays

You can also if it is to scale

Positioning on the map

You can determine the coordinates of the center of the map through the **Latitude** and **Longitude** properties, they are of type **double** and accessible in **read/write**. You can directly modify the coordinates through the procedure **setCenter(const dlatitude,dlongitude:double)**

```
// Delphi map component EMap
var lat,lng:double;
begin
    // get center of map
    lat := map.Latitude;
    lng := map.Longitude;
    // move center of map
    map.setCenter(lat+0.001,lng);

end;
```

Mouse position

The latitude and longitude of the point under the mouse cursor are returned by the property **MouseLatLng**

You can connect on the event **OnMapMouseMove** to know in time real your position Réagir au changement de position

When the center of the map is moved, This either by code or the mouse directly, the event **OnMapMove(sender: TObject;const dLatitude,dLongitude:double)** is raised

Sender represents the ECNativeMap component that has changed, dLatitude and dLongitude the new coordinates, who are also accessible through **Latitude** and **Longitude**

When the map starts to move the event **OnMapDragStart** is

raised, then **OnMapDrag** while the move and **OnMapDragEnd** at the end of displacement.

Area displayed

You can determine the coordinates of North East point (top-right) and point South West (lower-left) of the displayed area by the component through the properties **NorthEastLatitude**, **NorthEastLongitude**, **SouthWestLatitude** and **SouthWestLongitude**, they are double and accessible *read-only* .

The procedure **BoundingBox**(maxLatitude,maxLongitude,minLatitude,minLatitude) limit the accessible area of your map, the event **OnOutOfBounds** is raised if you attempt to access a restricted area.

*A call to **BoundinBox** without parameter throws limitations.*

The procedure **fitBounds**(const dLatlo,dLnglo,dLathi,dLnghi:double) allows you to adjust the view to the coordinates passed.

the procedure **boundingCoordinates**(const lat, lng, radius: double; var latSW, lngSW,latNE, lngNE: double); Returns the coordinates of a rectangular area on the basis of the focal point and a distance in km.

The function **ContainsLatLng**(var dLatitude,dLongitude:double):boolean tells you if the dLatitude, dLongitude is in the visible portion of the map.

As soon as the view changes the event **OnChangeMapBounds(sender: TObject)** is raised.

The function **ScreenShot** Returns an integer containing the image of the map

Zoom

The **Zoom** property allows you to control the definition of your card, it is of type integer and is accessible in **read/write**

*You have access also to the **MaxZoom** and property **MinZoom** that allow you to determine the limits of the zoom*

Use the **ZoomAround(const LatLngZoom: TLatLng; const NewZoom: Integer)** procedure to zoom while remaining focused on a specific point (as in mouse)

The zoom change triggers the event **OnChangeMapZoom(sender: TObject)**

By setting the **DragRect** property on **drZoom** you can select an area with the mouse by holding the right button, when you release the button zoom is performed on the selected region.

ZoomScaleFactor

This property allows you to emulate the intermediate zooms

```
// emulate zoom 16.35  
  
map.zoom := 16;  
map.zoomScaleFactor := 35;
```

*Make right Click + scroll wheel to
change ZoomScaleFactor*



Fig. 62 ZoomScaleFactor

ZoomScaleFactor can range from 0 to 99

Use the procedure **ZoomScaleFactorAround** (const **LatLngZoom:TLatLng**; const **NewZoomScaleFactor:Integer**) to zoom while remaining focused on a specific point (as with mouse)

Adapt the markers to the Zoom

Use the **ScaleMarkerToZoom** property to resize markers depending of Zoom.



Fig. 63 ScaleMarkerToZoom

Selection

The [Selected](#) property allows you to select items.

```
// select items in area
nbr_item_selected

:= map.Selected.ByArea(NorthEastLat, NorthEastLng, SouthWestLat, SouthWestLng);

// select items <=Max_Distance_KM to CenterPointLat, CenterPointLng
nbr_item_selected

:= map.Selected.ByKMDistance(CenterPointLat, CenterPointLng, Max_Distance_KM);

// loop to all selected items

var shape:TECShape

for shape in map.Selected do
begin
..
end;
```

By setting **DragRect** to **drSelect** you can make a selection with the mouse by holding the right button and release once the area is delimited.

You can redefine the appearance of the selection rectangle using [styles](#), example to have a line of 10 pixels width, green color, and a dotted line

```
map.styles.addRule(
  '#_DRAGZOOM_.line {weight:10;color:green;penStyle:dash}');
```



Fig. 64 stylized selection

To deselect all your items use **map.Selected.UnSelectedAll**

The **GroupFilter** list allows to define a filter on the groups allowed

```
// only elements from these groups will be accepted
map.Selected.groupFilter.add('group1');
map.Selected.groupFilter.add('group2');
// clear list for accept all
map.Selected.groupFilter.clear;
```

You can define your own filter by connecting you on **OnSelectShape**.

```
map.Selected.OnSelectShape := doOnSelectShape;
...
procedure TForm.doOnSelectShape(Sender: TObject; const Shape:
TECShape;
  var cancel: Boolean);
begin

  // accept only TECShapePOI
  cancel := not(Shape is TECShapePOI);
end;
```

Selected items are drawn using the color HoverColor, you can add a

distinguishing mark by connecting you on their OnAfterDraw event.

This example adds a red star in the upper right corner of selected items

```
// when you create item you must connect like this
item.OnAfterDraw := doAfterDraw

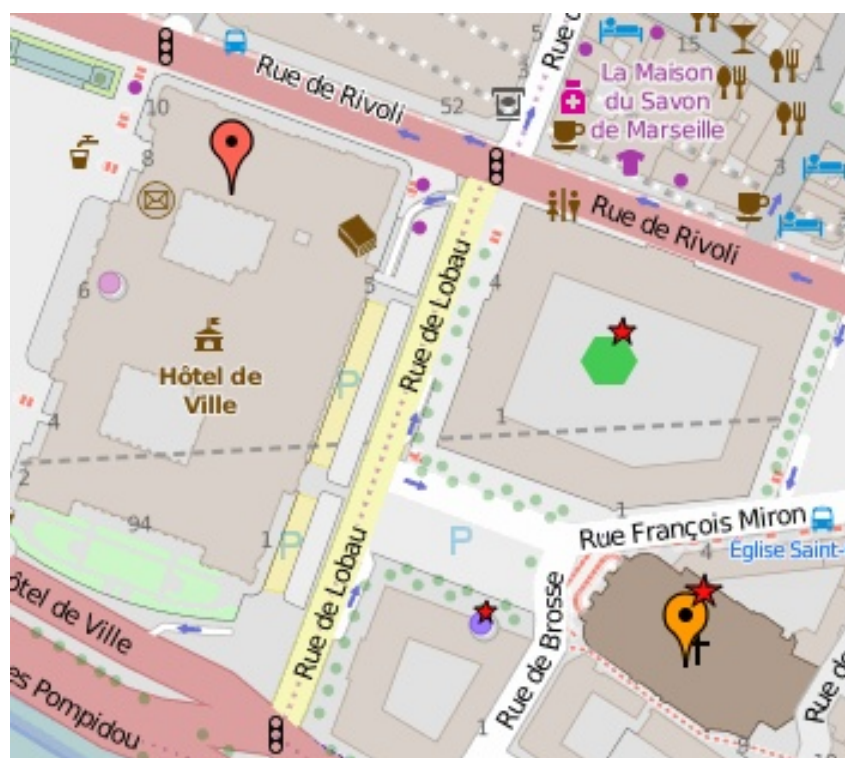
procedure TForm1.doAfterDraw(const canvas: TECCanvas; var rect:
TRect; item: TECShape) ;
var size_start : integer;
begin
    if assigned(item) and (item.Selected) then
    begin
        size_start := (item.Width div 2);
        if size_start < 10 then size_start := 10;

        canvas.Pen.Color := claBlack;
        canvas.PenWidth(1);
        canvas.Brush.Color := GetHighlightColorBy(claRed,16
    );

        rect.Left := rect.Left + (rect.Right-rect.Left)

        rect.Top := rect.Top - (size_start div 2);
        rect.Bottom := rect.Top + size_start ;

        canvas.DrawStar(rect);
    end;
end;
```



Or simply you can set a [style](#).

```
// all shapes selected is red
map.styles.addRule(':selected {color:red}');
// enlarge the selected markers
map.styles.addRule('.marker:selected {scale:1.2}');
```

Map rotation

Available only in the version Firemonkey

Put your component into a container (TPanel, TRectangle) which will be used to define the visible region and activate property

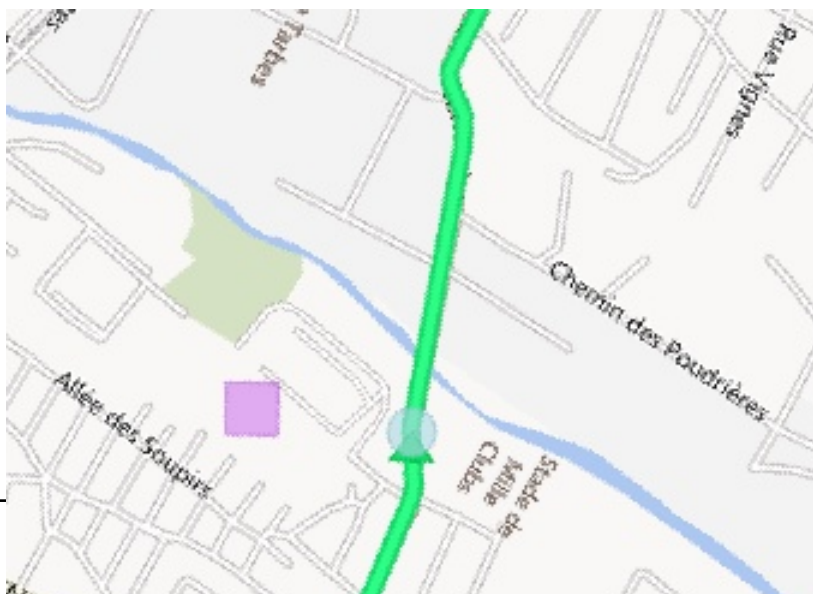
OverSizeForRotation

```
map.OverSizeForRotation := true;

// To allow or not the rotation gesture use the property EnableTouchRotation

map.EnableTouchRotation := true;
```

EnableTouchRotation *Enable rotation by gesture.*



Url

The **Url** property returns / accepts a string in the format '**#zoom/Latitude/Longitude**'

```
// zoom 18 latitude 48.856527 longitude 2.352104
// welcome to Paris !
map.Url := '#18/48.856527/2.352104';
```

Assigning a value to this property raises the event **OnBeforeUrl**(sender : TObject; var Url:string) that can allow you to change the url, and even to cancel the change by returning an empty string.

You can pass a such link in a **InfoWindow**

```
// welcome to Paris !
map.Shapes.InfoWindows[0].Content := 'Go to <a
href="#18/48.856527/2.352104">Paris !</a>';
```

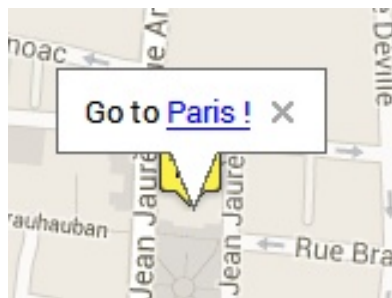


Fig. 67 Url Link

If you pass a 'classical' url it will open in your default browser.

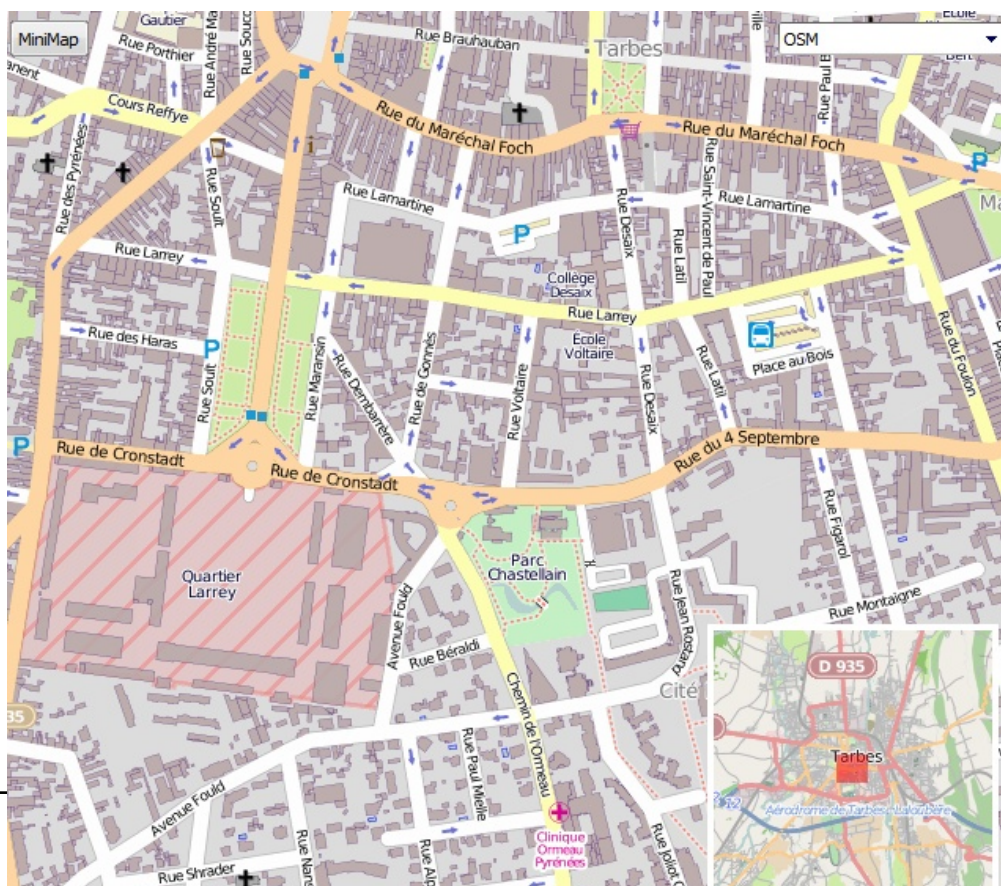
Distance

The function **DistanceFrom**(const dLatitudeStart,dLongitudeStart,dLatitudeEnd,dLongitudeEnd:double):double to calculate the distance between 2 points, the result is in kilometers.

Angle to the North

The function **Bearing**(const dLatitudeStart,dLongitudeStart,dLatitudeEnd,dLongitudeEnd:double):integer gives you the angle, from 0 ° to 360 ° for direction from the point *dLatitudeStart,dLongitudeStart* to the point *dLatitudeEnd,dLongitudeEnd*

MiniMap



To display a mini map in your main card simply add the **uecNativeMiniMap** unit to your listing and add the following lines.

```
// Delphi native map component TECNativeMap

var FMiniMap : TECNativeMiniMap;
...
// create and show minimap on Map
FMiniMap := TECNativeMiniMap.create(Map);

// for hide minimap
FMiniMap.Map := nil;

// for show
FMiniMap.Map := Map;
```

You do not have to release your minimap, It will be automatically during the destruction of the card to which it is attached.

You can change the corner of anchor with the property **ancragePosition** (apTopLeft, apTopRight, apBottomLeft, apBottomRight)

XMargin and **YMargin** allow you to adjust the position relative to the edges

BorderColor and **BorderSize** changes the color and size of the border surrounding the mini map.

Screenshot allows you to get an image of your miniMap, you will have to release it yourself!

Scale bar

To display the scale you need to add the `TECNativeScaleMap` unit and the following lines.

```

procedure TForm1.FormCreate(Sender: TObject);
begin

    FECNativeScaleMap := TECNativeScaleMap.create ;
    FECNativeScaleMap.Map := map;

end;

procedure TForm1.FormDestroy(Sender: TObject);
begin

    FECNativeScaleMap.free;

end;

```



Fig. 69 Scale bar

The following properties are available :

AnchorPosition (apTopLeft, apTopRight, apBottomLeft, apBottomRight)

XMargin and **YMargin** to adjust from the corners

BarSize fineness of the bar (default 2)

Color color (default black)

MaxWidth maximum length of bar in pixels (default 80)

MesureSystem measurement system (msMetric, msImperial)
default msMetric

Shadow Adds a shadow (default true)

Measurement tool

You can use a measuring tool by adding the **uecNativeMeasureMap** unit and lines

```
procedure TForm1.FormCreate(Sender: TObject);
begin

    FECNativeMeasureMap := TECNativeMeasureMap.create ;
    FECNativeMeasureMap.Map := map;

    FECNativeMeasureMap.StartMeasure(map.latitude,map.longitude);

end;

procedure TForm1.FormDestroy(Sender: TObject);
begin

    FECNativeMeasureMap.free;

end;
```



The following properties are available :

Color line color default black

MeasureSystem unit of measure (msMetric, msImperial)
default msMetric

Distance : double the distance in the selected unit

DistanceLabel : string the distance in text (the one that appears on the screen)

ShowDistance allows to show or not the distance on the screen
(default true)

Hint the text of information when leaving the mouse on the line

OnChange : TNotifyEvent raised when the distance changes

Observer

An observer allows you to be notified when events is fired in map or in shape.

This has the advantage of not to block the main events of the map and the shapes.

```
TNativeMapObserver = class(TObject)
public

    property OnMapFree : TNotifyEvent ;
    property OnMapHiResChange : TNotifyEvent ;
    property OnMapActiveChange : TNotifyEvent
    property OnMapAnimation : TNotifyEvent;
    property OnMapResize : TNotifyEvent;
    property OnMapRotation : TNotifyEvent;
```

```

property OnMapMove : TNotifyEvent;
property OnMapMouseMove : TNotifyEvent;
property OnMapMouseClicked : TNotifyEvent;
property OnMapEndMove : TNotifyEvent;
property OnMapzoom : TNotifyEvent;
property OnMapLoad : TNotifyEvent;
property OnMapBounds : TNotifyEvent;
property OnMapChangeBounds : TNotifyEvent;
property OnMapTileServer : TNotifyEvent;
property OnMapAddRoute : TNotifyEvent;
property OnMapErrorRoute : TNotifyEvent;
property OnMapChangeRoute : TNotifyEvent;
property OnMapPaint : TNotifyEvent;
property OnShapesPaint : TNotifyEvent;
property OnMapShapeDescription : TNotifyEvent;
property OnMapShapeProperties : TNotifyEvent;
property OnMapShapeHint : TNotifyEvent;
property OnMapShapeClick : TNotifyEvent;
property OnMapShapeMove : TNotifyEvent;
property OnMapShapeDbClick : TNotifyEvent;
property OnMapShapeMouseOut : TNotifyEvent;
property OnMapShapeMouseOver : TNotifyEvent;
property OnMapShapeDrag : TNotifyEvent;
property OnMapShapeDragEnd : TNotifyEvent;
property OnMapShapePathChange : TNotifyEvent;
property OnMapShapeRightClick : TNotifyEvent;
property OnMapShapesChange : TNotifyEvent;

```

You can add as many observers as you want.

```

FObserver1 := TNativeMapObserver.Create;
FObserver1.OnMapResize := Map_Resize;
FObserver1.OnMapMove := Map_Move;

FObserver2 := TNativeMapObserver.Create;
FObserver2.OnMapBounds := Map_Bounds;
FObserver2.OnMapTileServer := Map_TileServer;
FObserver2.OnMapRotation := Map_Rotate;

// attach observer
map.attach(FObserver1);
map.attach(FObserver2);
...
// detach
map.detach(FObserver1);
map.detach(FObserver2);
...
FObserver1.free;
FObserver2.free;

```

The [layers](#) using observers to react to the change of position.

The tiles are first searched in the memory cache (the tiles displayed recently), the local cache, archive local then on the internet if need.

Local cache

By assigning a directory to **LocalCache** property internet downloaded tiles are saved locally and are available offline.

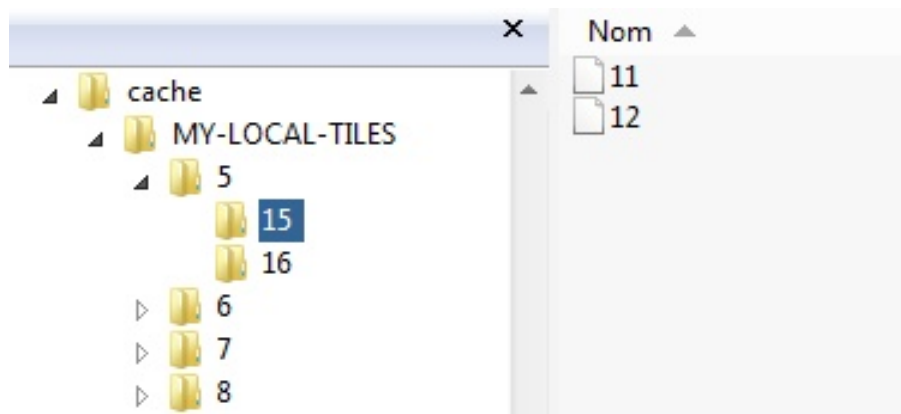


Fig. 71 My local tiles

Use the **MaxDayInCache** property to specify the duration of retention in the cache by default 30 days, 0 for an infinite cache.

```
map.MaxDayInCache := 7; // max 7 days
```

Archive local

An archive local is the local cache in a Zip, this simplifies deployment of the tiles and other files.

To improve the speed, the tiles are extracted from archive and placed in

the local cache during the first request, you must set a local cache to use an archive.

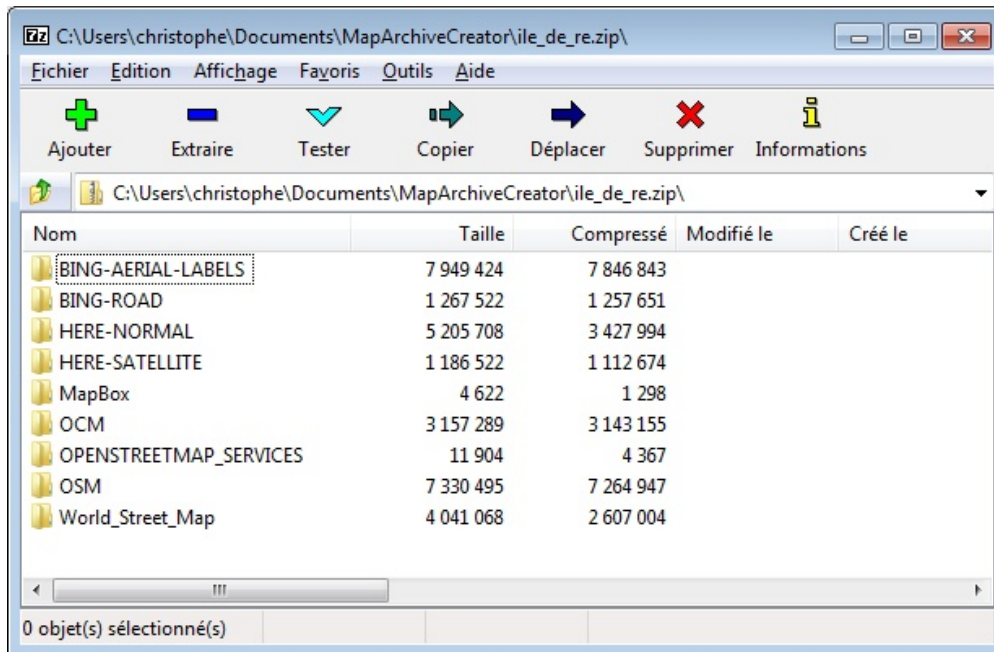


Fig. 72 Archive ile de ré

```
map.LocalCache
:= TPath.Combine(TPath.GetSharedDocumentsPath, 'cache');
map.TileServer := tsOSM;
map.LocalArchive := ExtractfilePath(ParamStr(0))+
'ile_de_re.zip';
```

If roads or geolocations of addresses are saved in archive recovery is completely transparent

```
map.Routing.engine(reMapBox);
// if an archive is connected and contains the route, no
internet connection is made to return the way
map.Routing.Request('saint-martin de ré',
'la couarde sur mer');
```

You can store your images or data files (kml, geojson etc...), to load start the name of the file by /

```
// load data
map.Shapes.LoadFromFile('/DATA/tdf.kml');
map.LoadFromFile('/DATA/tarbes.txt');
// load image in marker
marker.filename := '/IMAGE/node.png';
```

Use MapArchive to directly manipulate your archive and retrieve other data.

```
m := TMemoryStream.Create;
try

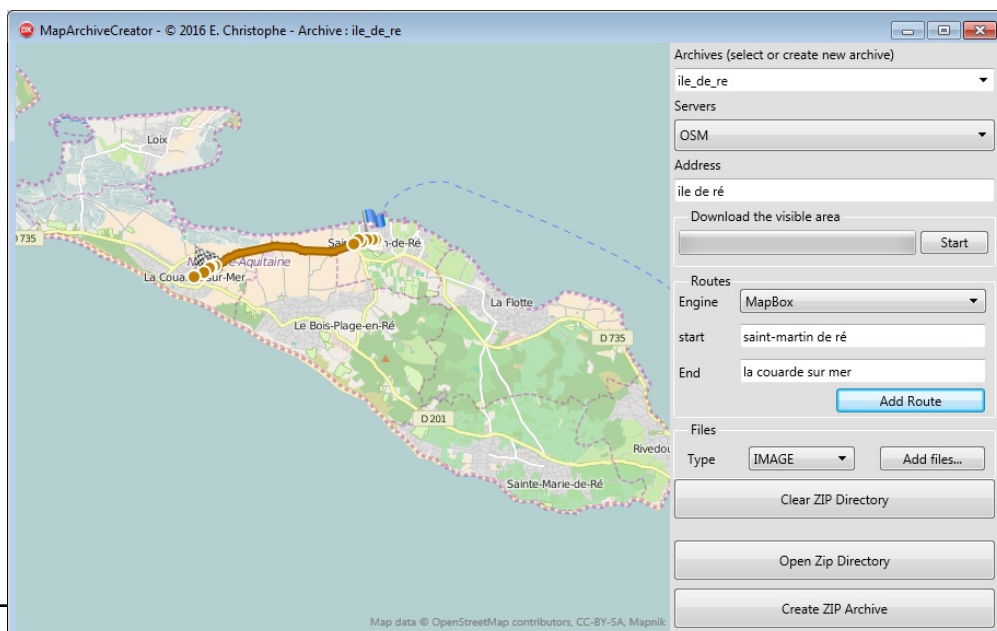
    map.MapArchive.ReadStream('DATA/mydata.txt',m);

    memol.Lines.LoadFromStream(m);

finally
    m.Free;
end;
```

MapArchiveCreator

This small utility allows you to create your archives, you can download areas of tiles, save routes, add images and files.



*You can ban any internet connection for the tiles and geocoding using the property **OnlyLocal***

```
// tiles and geocoding only uses local cache and local archive
map.onlyLocal := true;
```

Download full a zone

The **TECDownloadTiles** class allows you to download in the background one complete zone, the **EcNativeMapFiremonkeyDemo** shows you how to preload the viewable area on the screen.

```
FECDownloadTiles:=TECDownloadTiles.create;

FECDownloadTiles.OnDownload      := doDownloadtiles;
FECDownloadTiles.OnEndDownload  := doEndDownloadtiles;

// tiles are saved in DirectoryTiles
FECDownloadTiles.DirectoryTiles := map.LocalCache;

FECDownloadTiles.TileServer      := map.TileServer;
FECDownloadTiles.TileSize       := map.TileSize;

// download visible area from zoom+1 to MaxZoom

FECDownloadTiles.DownloadTiles(map.Zoom+1,map.MaxZoom,

                                map.NorthEastLatitude,map.NorthEastLongitude,

                                map.SouthWestLatitude,map.SouthWestLongitude,

                                ...

// for abort
FECDownloadTiles.Cancel;

// see ECNativeMapFiremonkeyDemo for complete use
```

Make sure that your provider of tile allows it !

Fill the Stream of tiles

You can also directly return a stream containing the jpeg or png of your tiles, useful if you have your tiles in a database.

```
procedure TForm.GetTileStream(var
  TileStream: TMemoryStream;const x, y, z: integer);
begin
  // here fill the stream with your tile

end;

procedure TForm.FormCreate(Sender: TObject);
begin

  map.TileServerInfo.GetTileStream := GetTileStream;
  map.TileServerInfo.Name := 'MyStream';
  map.TileServerInfo.TileFormat := stJpeg; // or stPng
  // called last, if not the first display is not performed
  map.TileServer := tsOwnerDraw;

end;
```

Also allows to use a

Support for vector tiles geoJSON

MapZen provides free [vector tiles](#) from the data of [OpenStreetMap](#)

You must obtain a free key and then populate the property `VectorMapZenKey` of your map.

ATTENTION: MapZen no longer exists, but `TECNativeMap` still support their formats because their tools are available as Open Source.

```
// connect your key for use vector tiles  
map.VectorMapZenKey := 'vector-tiles-XXXXXX';
```

`TECNativeMap` add a new `Server` `tsVectorMapZen`, the tiles will be in the format [geoJSON](#)

In this format the tiles can quickly be very heavy, 300 KB even + 500 KB, especially in the zoom <16

To limit the weight it selects [layers](#) based on zoom

Zoom 14 : 'water,roads'

Zoom 15 : 'places,water,roads'

Zoom 16,17 : 'places,water,roads,buildings,pois'

Zoom 18+ : 'landuse,places,water,roads,buildings,pois'

Of course this is not yet final, and you can always make

another choice

Display and download of the tiles is not faster than bitmaps tiles, on the contrary, but the big advantage is that you can access a lot more data, some [OSM tags](#) are available in addition to the geometric data, for example the population of a city, the number of houses ...

Alternatively, you can use the [layer XAPI](#) to display data from [OpenStreetMap](#).

In order to manage these new data **Properties** and **PropertyValue** are added to the [TECShape](#)

Properties is filled with the field "properties" tiles, in the format "nom:valeur", each pair is separated by a carriage return.

PropertyValue[Name] allows you to access the value of the property **Name**

You can freely use them to store your own data.

Style sheet

To be able to properly display all its vector data it becomes mandatory to use styles.

TECNativeMap therefore has the property **Styles**

```
map.Styles.LoadFromFile(your_filename);  
map.Styles.Rules := all_your_rules;
```

```
map.Styles.addRule(your_rule);
```

A rule can decompose in a selector and a list of properties placed between { }

```
Selecteur {prop1:value1;prop2:value2}
```

Style selector

The selector can be the name of a [Group](#), an element type ([marker](#), [poi](#), [line](#) or [polygone](#)), the name property of the element, a set Property:value

You can also aggregate these different possibilities (groupe+type+property:value)

```
#Group_Name {} // for all element in the group
.Marker {} // for all TECShapeMarker
#Name_Of_Shape {} // only for this name
.property_name:property_value {} // for all items with
this value
#Group_Name.Polygone {} // only for TECShapePolygone in
this group
.Line.property_name:property_value {} // only for
TECShapeLine with this value
```

For the TECShapePOI you can also select depending on the type

```
.Poi.Ellipse {}
.Poi.Star {}
.Poi.Rect {}
.Poi.Triangle {}
.Poi.Diamond {}
.Poi.Hexagon {}
.Poi.Text {}
```

It is also possible to define rules based on states hovered over by the mouse and selected.


```
// all the elements hovered by the mouse are green
map.styles.addrule(':hover {color:green}');

// the border of the lines hovered by the mouse is 4 pixels
map.styles.addrule('.line:hover {bsize:4}');
// all shapes selected is red
map.styles.addRule(':selected {color:red}');
// enlarge the selected markers
map.styles.addRule('.marker:selected {scale:1.2}');
```

You can apply the same properties to several selector separated by commas

```
Select1 , Select2, Select3 {...}
```

Style property

The properties available are :

```
bcolor (BorderColor)
bsize
BrushStyle (vcl
Clear|Cross|DiagCross|BDiagonal|Horizontal|FDiagonal|Vertical|Solid)
( fmx Graphic )
Color
FontItalic (true|false)
FontBold (true|false)
Fontsize
FontFamily
fcolor (fill color)
fopacity (0-100)
graphic (url|base64,data_png_in_base64)
hbcolor (hover border color)
hcolor (hover color)
height
level
opacity (0-100)
penStyle (solid|dash|dot|dashdot)
unit (pixel|meter)
visible (true|false|[text])
weight
width
zindex
StyleIcon
```

```
(3D,Flat,FlatNoBorder,Svg,OwnerDraw) (TECShapeMarker)
Scale (double) (TECShapeMarker & TECShapePoi);
```

For the colors you can use the syntax #RRGGBB or \$int_value or directly a name (red, black, green etc.)

You can also create a color by specifying two colors and a percentage of mixture

```
// create a color by mixing
map.styles.addRule('.line {color:gradient(Red,Yellow,0.6)'});
```

Using [text] It displays the item only if

PropertyValue['name']<>" ou

PropertyValue['addr_housenumber']<>"

Excerpt from a style sheet

```
.Polygone {weight:0;zindex:-1;}
.marker {zindex:11}
.marker: hover {scale:1.5}
.marker.kind:embassy{visible:true;graphic:base64,iVBORw0KGgoAAAANSUhEUgAAAA4A

.kind:rail {bcolor:#0C0D0D;color:white;weight:1;bsize:3;penStyle:dash;}
.kind:river {color:#54b4eA;weight:4;}
```



Conditional property

To apply a rule based on a PropertyValue, use the if property

With the following rules the villages with a population of less than 5,000 inhabitants appear in green, those with a population > 6000 in blue.

```
.kind:village {if:population<5000;fontsize:12;color:green;}  
.kind:village {if:population>6000;fontsize:12;color:blue;}
```



Fig. 75 green < 5000 - blue > 6000

You can use as a comparator = , !=, >, >=, et <=

You can test "special" properties

RTL (vcl ou fmx)

```
.Polygone {if:rtl=fmx;hcolor:green}
```

OS (windows, mac ,android ou ios)

```
.marker: hover {if:os=windows;color:red}
```

HOLE (true ou false) to detect if a polygon is a hole in another polygon, TECNativeMap does not support polygons with holes, This test at least to assign a color for holes.

```
.polygone.kind:building {if:hole=true;fcolor:silver;fopacity:100;zindex:0;}
```

Rule based on the Zoom

You can set the zooms for which the rule applies, just add at the end of selector [zoom valeur] or [zoom valeur1,valeur2] , [zoom *] indicates that the rule applies for all zooms

```
.Poi.Text [zoom *]{if:kind=building;visible:false;}
.Poi.Text
[zoom 18,19,20,21,22]{if:kind=building;visible:[text];}
.Poi.Text [zoom *]{if:kind=chapel;visible:false;}
.Poi.Text
[zoom 19,20,21,22]{if:kind=chapel;visible:[text];}
```

Multiple values based on the zoom

For the **StyleIcon**, **FontSize**, **Scale** and **Weight** properties you can set several values that depend on the zoom level

The syntax is **zoom_mini-zoom_max=value,zoom2=value**

```
// fontsize = 6pt in zoom 17 , 8pt in zoom 18 , 10pt in
zoom 19 & 20
#.poi {fontsize:17=6,18=8,19-20=10;}
// flat marker for zoom 0 to 17, siDirection for zoom 18
to 20
.marker {styleicon:0-17=flat,18-20=direction}
```

Define a name for a value

You can name your values and use the name instead of the value.

```
@dark {#404040}

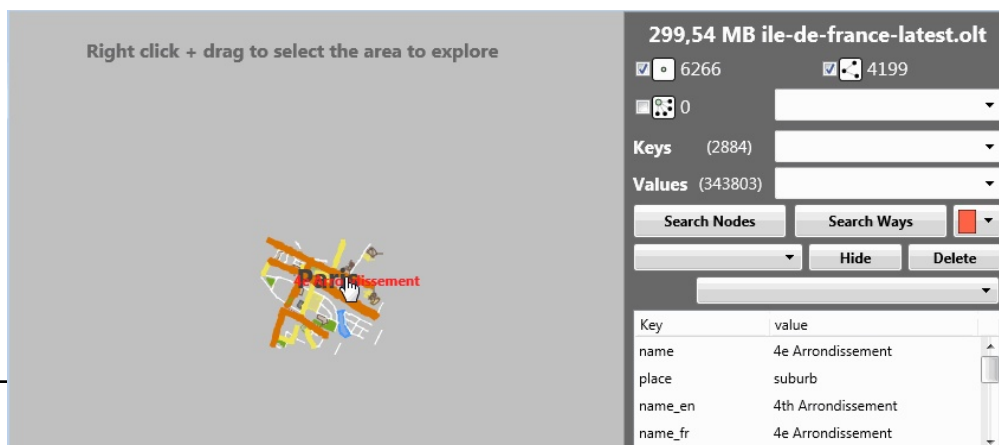
@scale-suburb-village {0-11=0,12=1.2,13=1.5,14=1.6,15-16=1.9,17-20=2.5}

.poi
{if:place=suburb;fontsize:12; scale:@scale-suburb-village;color:@dark;width:0}
```

You can use styles without using vector tiles, they apply to all your elements

The unit **uecOSMStyles_standard** declares the constant **UEC_OSM_STYLESHEET**, these are the styles used in the demo [OSMViewer](#)

```
// see uecOSMStyles_standard
map.styles.Rules := UEC_OSM_STYLESHEET;
```



clickable items

Each element of the tiles is clickable, you can intercept the click with the same events for your own items (OnShapeClick, ShapeRightClick et OnShapeLongClick).

These events are not restricted to the level of the tiles, they are passed to your map, and you have an OnMapClick after an OnShapeClick !

Research

You can use the same [search functions](#) than for your own items.

```
// find the road located less than 100 meters
map.FindShapeByKMDistance(map.Latitude,map.Longitude,0.1
,liste,[nsLine],FilterRoad);
...

procedure FilterRoad(const Shape: TECShape; var
cancel: boolean);
begin
    cancel := pos('road',shape.PropertyValue['kind'])< 1 ;
    if not cancel then
        cancel := shape.PropertyValue['name']='';
end;
```

Hide vector tiles

You can not display the vector tiles but continue to use the data (click and search)

```
// add vector tiles overlay
```

```
map.AddOverlayTileServer(tsVectorMapZen);  
    // hide the tiles but the data is always available  
map.DrawVectorTiles := false;
```

Demos

To get an idea of the result you can download [a demo for Windows](#) and a [for Android](#) (unzip and install the apk)



Fig. 77 Vectors tiles

All the elements are not styled, it is just the minimum for this is readable (I did not say beautiful !)

you have the possibility to connect several TECNativeMap on your motherboard, you'll get as many different view.

The views are independent from each other but they all use the elements of the main map, the data are not duplicated they exist only in the main map.

If you added elements in one of the side views, they are actually integrated in the main view.

Any changes on one of the views is immediately passed on to all of the maps



Fig. 78 Left the main map - right side view

```
procedure TNativeMapControl.AddView
```

```
(const view:TNativeMapControl);
```

```
procedure TNativeMapControl.ReleaseView
```

```
(const view:TNativeMapControl);
```



```
procedure TNativeMapControl.ReleaseAllView;
```

```
property TNativeMapControl.ViewCount : integer;
```

```
property TNativeMapControl.Views[index:integer]
: TNativeMapControl
```

```
property TNativeMapControl.OnAddShapeToView
: TOnAddShapeToView
```

Tune in to this event if you want to filter the items displayed in the view

Example, we want to see that elements of type [TECShapePOI](#) star shaped

```
// add view for map
Map.addView(ViewA);
// add filter on ViewA
ViewA.OnAddShapeToView := doOnAddShapeToView;
...
procedure TForm.doOnAddShapeToView(sender : TObject; const
Shape:TECShape;var cancel:boolean) ;
begin

    // sender is the view, cast for use TECNativeMap(sender);

    if Shape is TECShapePOI then
        cancel := not (TECShapePOI(shape).POIShape = poiStar)
    else
        cancel := true;

end;
```

Default TECNativeMap uses its own tile server but we can deport him on a remote workstation.

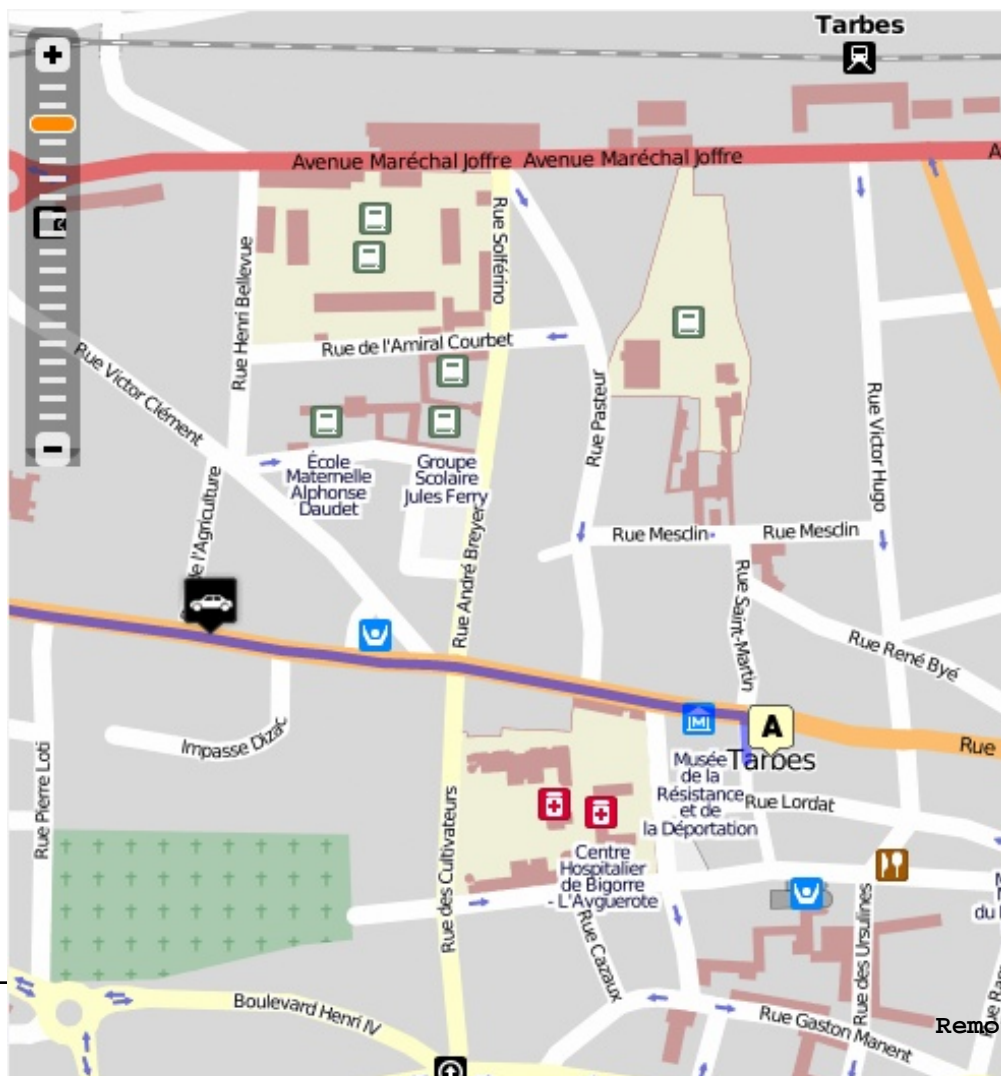
For our demo we use [WebSockets of André Mussche](#)

It's a minimalist demo that did not claim to be a tutorial for the use of the WebSockets !



Download sources and executables of demos

Tile server



You can specify a cache to limit internet connections

Our server does not have the `TECNativeMap` component, let's just use the class **`TNativeMapServer`** of the **`uecNativeTileServer`** unit

```

procedure TForm1.FormCreate(Sender: TObject);
begin

    // create tile server ( unit uECNativeTileServer )
    FMapServer      := TNativeMapServer.Create(nil);
    FMapServer.OnLoadTile := doOnLoadTile;
    FMapServer.TileServer := tsOSM;

    // create WebSocket server
    server := TIdWebsocketServer.Create(Self);
    server.SocketIO.OnSocketIOMsg := doMessage;

end;

    // request tile from client : ^Tile|x|y|z
procedure TForm1.doMessage(const
    ASocket: ISocketIOContext; const aText:string; const aCallback:
    ISocketIOCallback) ;
    var Text:string;
    sxStream:TMemoryStream;
    ix,iy,iz,isz:integer;
begin

    TThread.synchronize(nil,
    procedure
    begin

        Text := aText;

        if pos('^Tile',text)>0 then
        begin
            StrToken(text, '|');

            ix:=StrToIntDef(StrToken(text, '|'),0);
            iy:=StrToIntDef(StrToken(text, '|'),0);
            iz:=StrToIntDef(StrToken(text, '|'),0);
            sxStream:=TMemoryStream.Create;

            // request tile
            if FMapServer.GetStreamTile(sxStream,ix,iy,iz) then
            begin
                // if tile is ready send to client

```

```

        ASocket.Send('^Tile|'+inttostr(ix)+'|'+
+inttostr(iy)+'|'+inttostr(iz)+'|'+
+EncodeStreamTo64(sxStream));
        end;

        sxStream.Free;
        end;

    end);
end;

// tile is ready now, send to clients
procedure TForm1.doOnLoadTile(sender: TObject; const x,
y, z, t: integer);
var TileStream: TMemoryStream;
begin

    TThread.synchronize(nil,
        procedure
        begin

            // send response : ^Tile|x|y|z|Base64-Stream-Data-Tile
            server.SendMessageToAll('^Tile|'+inttostr(x)+'|'+
+inttostr(y)+'|'+inttostr(z)+'|'+
+EncodeStreamTo64(TileStream));

            end);

        end;

end;

// connect / disconnect server
procedure TForm1.btConnectClick(Sender: TObject);
begin

    if not Server.Active then
        begin
            Server.DefaultPort:=StrToIntDef(edPortServer.Text,
8080);
            end;
            Server.Active:=not Server.Active;
            if Server.Active then btConnect.Caption:='Stop server'
else btConnect.Caption:='Start server';

        end;

// set cache
procedure TForm1.ckCacheClick(Sender: TObject);
begin
    if ckCache.Checked then

```

```

begin
    FMapServer.LocalCache:=edCache.Text;
end else
begin
    FMapServer.LocalCache:= ' ';
end;
end;

procedure TForm1.FormCloseQuery(Sender: TObject; var CanClose:
Boolean);
begin
    if CanClose then
    begin
        FMapServer.Free;
    end;
end;

```

Client

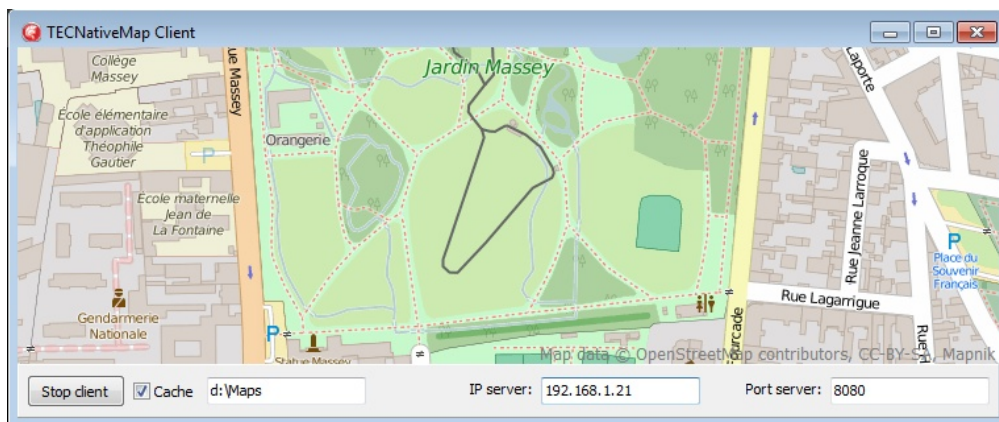


Fig. 80 Client

The client doesn't need to have internet access, it must just be able to connect to a local station that will have him access to internet and that will turn our server.

```

procedure TForm2.FormCreate(Sender: TObject);

```

```

begin

    // manual management tiles as stream
    map.TileServerInfo.GetTileStream := GetTileStream;
    // the name will be uses as subdirectory in cache
    map.TileServerInfo.Name := 'MyMAP';
    // important to specify a manual management tiles
    map.TileServer := tsOwnerDraw;

    // do the same for the tiles low resolutions
    map.LowTileServerInfo.GetTileStream := GetLowTileStream;
    map.LowTileServerInfo.Name := 'MyMAP';
    map.LowTileServer := tsOwnerDraw;

    // create WebSocket clients for communication with server
    client := TidHTTPWebSocketClient.Create(self);
    client.SocketIOCompatible := true;
    client.SocketIO.OnSocketIOMsg := doMessage;

    // client for low tiles
    client2 := TidHTTPWebSocketClient.Create(self);
    client2.SocketIOCompatible := true;
    client2.SocketIO.OnSocketIOMsg := doLowMessage;

end;

// asks for a tile
procedure TForm2.GetTileStream(var
    TileStream: TMemoryStream;
    const x, y, z: integer);
begin

    // send request to the server : ^Tile|x|y|z
    client.SocketIO.Send('^Tile|' + inttostr(x) + '|' +
    inttostr(y) + '|' + inttostr(z));

end;

// asks for a tile low resolution
procedure TForm2.GetLowTileStream(var
    TileStream: TMemoryStream;
    const x, y, z: integer);
begin

    client2.SocketIO.Send('^Tile|' + inttostr(x) + '|' +
    inttostr(y) + '|' + inttostr(z));

end;

```

```

    // response from server, get tile : ^Tile|x|y|z|Base64-Stream-Data-Tile
procedure TForm2.doMessage(const
  ASocket: ISocketIOContext; const aText: string;
  const aCallback: ISocketIOCallback);
var
  Text: string;
  sxStream: TMemoryStream;
  ix, iy, iz: integer;
begin

  Text := aText;

  if pos('^Tile', Text) > 0 then
  begin
    StrToken(Text, '|');
    ix := StrToIntDef(StrToken(Text, '|'), 0);
    iy := StrToIntDef(StrToken(Text, '|'), 0);
    iz := StrToIntDef(StrToken(Text, '|'), 0);
    sxStream := TMemoryStream.Create;
    if Decode64ToStream(Text, sxStream) then
    begin
      map.AddStreamTile(sxStream, ix, iy, iz);
    end;
    sxStream.Free;
  end;

end;

    // response from server for low tile, get tile : ^Tile|x|y|z|Base64-Stream-Data-Tile
procedure TForm2.doLowMessage(const
  ASocket: ISocketIOContext; const aText: string;
  const aCallback: ISocketIOCallback);
var
  Text: string;
  sxStream: TMemoryStream;
  ix, iy, iz: integer;
begin

  Text := aText;

  if pos('^Tile', Text) > 0 then
  begin
    StrToken(Text, '|');
    ix := StrToIntDef(StrToken(Text, '|'), 0);
    iy := StrToIntDef(StrToken(Text, '|'), 0);
    iz := StrToIntDef(StrToken(Text, '|'), 0);
    sxStream := TMemoryStream.Create;
    if Decode64ToStream(Text, sxStream) then
    begin

```

```
        map.AddStreamLowTile(sxStream, ix, iy, iz);
    end;
    sxStream.Free;
end;

end;

// connect / disconnect from server
procedure TForm2.btConnectClick(Sender: TObject);
begin

    if not client.Connected then
    begin
        client.Host := edIPServer.Text;
        client.Port := StrToIntDef(edPortServer.Text, 8080);
        client.Connect;

        client2.Host := edIPServer.Text;
        client2.Port := StrToIntDef(edPortServer.Text, 8080);
        client2.Connect;
    end
    else
    begin
        client.Disconnect(false);
        client2.Disconnect(false);
    end;

    if client.Connected then btConnect.Caption:='Stop client'
    else btConnect.Caption:='Start client';

end;

// set cache
procedure TForm2.ckCacheClick(Sender: TObject);
begin
    if ckCache.Checked then
    begin
        Map.LocalCache:=edCache.Text;
    end else
    begin
        Map.LocalCache:='';
    end;
end;
end;
```

The client can also have its own cache of tiles.

API Keys

TECNativeMap uses OpenStreetMap.

If you want to use these services of MapQuest [you must obtain a key from MapQuest](#).

Do the same to use the services of [MapZen](#) and [MapBox](#).

Offline mode

If you specify a directory in the [LocalCache](#) property, all your geolocation data will be cached and can be reused in offline mode.

Address format

You benefit from the services of geolocation via the property **GeoLocalise** de type **TECGelocalise**

The **Address** property gives you the address of the center of the map, it is of type **string** and is accessible in **read/write**

TECGeolocalise uses the services of OpenStreetMap and [OpenMapQuest](#)

You can also use **ArcGis** through functions

```
function TECGeolocalise.ArcGisReverse(const Lat,Lng:double):string;  
function TECGeolocalise.ArcGisFind(const data:string;var Lat,Lng:double):boolean;
```

Geolocalisation

To obtain the address of a specific you have the function **GetAddressFromLatLng(dLatitude,dLongitude:double):string;**

You can get the various parts of the address by using the property **TECGeolocalise.ReverseResults:TStringList**

```
adr := map.GetAddressFromLatLng(Latitude,Longitude) ;

// now ReverseResults contains tags in nominatim <addressparts>

country := map.Geolocalise.ReverseResults['country'];
road     := map.Geolocalise.ReverseResults['road'];
postcode:= map.Geolocalise.ReverseResults['postcode'];
...
```

You can get the coordinates of an address with the function **GetLatLngFromAddressAddress:string;var dLatitude,dLongitude:double):boolean;**

By assigning a value to the property **Address** you will change the position of the center of your map to match to the address

Places

TECGeolocalise allows you to search specific locations in a data area, for example to find restaurants within a radius of 500 metres.

It is [Overpass-api](#) that uses data from [OpenStreetMap](#) which is used

The property **TECGeolocalise.Places.XapiServer** allows you to use another server Xapi than MapQuest

```
// Delphi map component EMap
```

```
// use overpass-api.de

map.GeoLocalise.Places.XapiServer :=
  'http://www.overpass-api.de/api/xapi?';
```

procedure **Search**(Tags:string);

Launch a search, tags contains the query

[Documentation Xapi](#)

```
// Delphi map component EMap
Map.geolocalise.OnSearch := mapPlacesSearch;

Map.GeoLocalise.Places.latitude := map.latitude;
Map.GeoLocalise.Places.longitude := map.longitude;

Tags := 'node[amenity=restaurant]';

// 500 meters
Map.GeoLocalise.Places.radius := 500;

Map.GeoLocalise.Places.Search(Tags);

...

procedure TForm.mapPlacesSearch(sender: TObject);
var
  i, max: Integer;
  types, names: string;
begin

  if map.GeoLocalise.Places.Status <> 'OK' then
    exit;

  max := map.GeoLocalise.Places.results.count - 1;

  for i := 0 to max do
    begin

      types := map.GeoLocalise.Places.results[i].result[
        'types'];
      names := map.GeoLocalise.Places.results[i].result[
        'name'];
      end;

    end;
```

When the search is complete, the **GeoLocalise.OnSearch** event is raised

Each launch of Search clears the results of a previous search

property **Address** : string read FAddress write FAddress;

Property **read/write** indicating the focal point of the search box, it takes precedence over the properties **Latitude** et **Longitude**

property **Status** : string;

Property **read-only** that returns a string indicating the status of the search, 'OK' if all went well

The status is available in the event **GeoLocalise.OnSearch**

property **ItemDetail** : integer;

Property **read-only** that contains the index of the result which is more details

property **Latitude** : double;

Property **read/write** that indicates the latitude of the central point of the area of research, this invalid the contents of the property **Address**

property **Longitude**: double;

Property **read/write** indicating the longitude of the central point of the area of research, this invalid the contents of the property **Address**

property **Radius** : integer;

Property **read/write** that indicates the radius of search **in metres**

property **maxResult** : integer;

limits the number of results for a search in the OpenStreetMap data

property **Searching**: boolean;

Propriété en **lecture seule** qui indique si une recherche est en cours

property **Results**:TECPlaceResults;

Results list, the **OnPlacesSearch** event is raised when the results are available

TECPlacesResults

This class manages the list of the results returned by **Search**

procedure **Clear**;

Clears all the results

function **Count**:integer;

Returns the number of result for the query

procedure **Delete**(const index:integer);

Clears the result which we pass the index

property **Result**[index:integer]:TECPlaceResult

Table allowing access to results

TECPlaceResult

Class handling a result corresponding to a search

property **Result**[const key:string]:string;

Returns the value of the key that is passes as a parameter

property **Detail**[const Key:string]:string;

Returns the value of the key that is passes as a parameter for details

property **NameResult**[const index:integer]:string;

Returns the key of a result based on its index

property **NameDetail**[const index:integer]:string;

Returns the key of a detail based on its index

property **CountResult**:integer;

Returns the number of result element (key = value)

property **CountDetail**: integer read getCountDetail;

Returns the number of a detail element (key = value)

property **Latitude** : double;

Latitude of the result

property **Longitude**: double;

Longitude of the result

property **RawResult** : string;

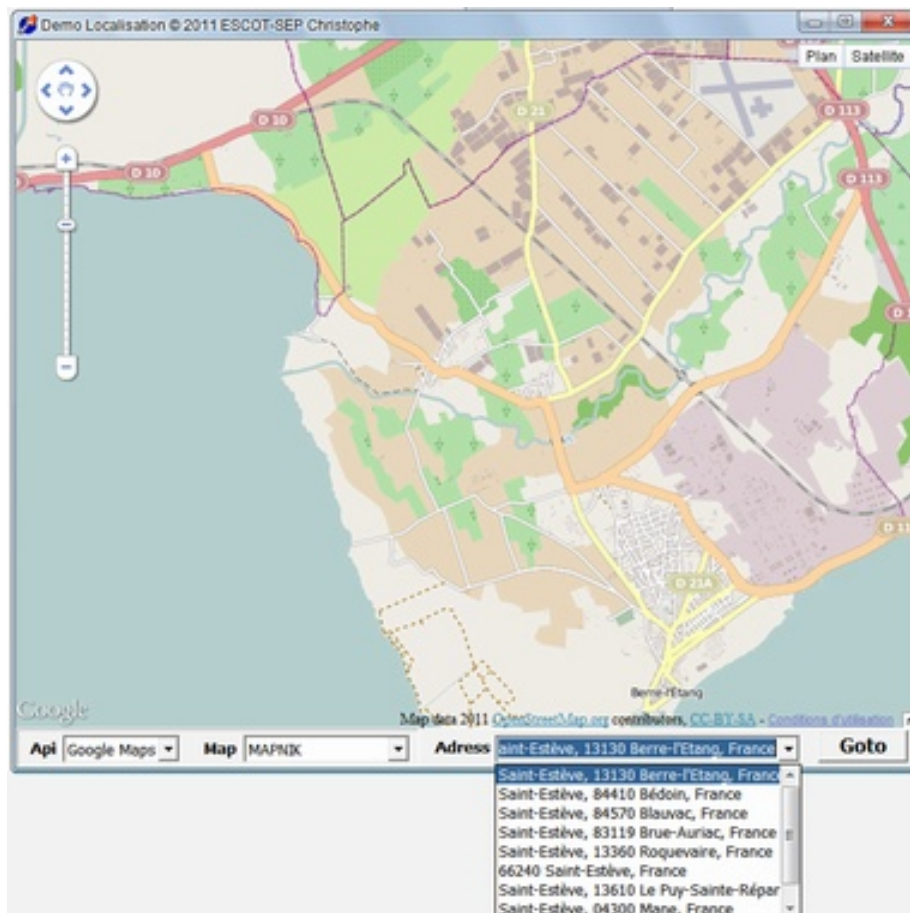
Returns the set of the result elements in the form of a string of rows **key = value**

property **RawDetail** : string;

Returns the set of elements of the detail in the form of a string of rows **key = value**

DemonativeLocalise

the **DemoNativeLocalise** program shows you how to manage **Places**



Geofences

Geofence is a virtual area that triggers an alert when you enter or exit.

function **Add**(const Lat,Lng:double;const RadiusMeter:integer;const Name:string=""):integer;overload;

Adds a circular area defined by its center point and RADIUS in meters.

```
map.geofences.add(43.231572,0.080853,50);
```

function **Add**(const dLatLngs: array of double;const Name:string=""):integer;overload;

Adds a polygonal area defined by an array containing the various points

```
map.geofences.add([lat1,lng1,lat2,lng2,lat3,lng3],"triangle");
```

function **Add**(const Polygone:TECShapePolygone;const Name:string=""):integer;overload;

Adds a polygonal area defined by a polygon

```
map.geofences.add(map.shapes.polygons[0]);
```

procedure **Delete**(index:integer);

Delete a geofence

function **Count**:integer;

Returns the number of geofences

function **IndexOf**(value:TECBaseGeofence):integer;

Returns the index of the geofence

procedure **Clear**;

Clears all the geofences

property **Active** : boolean ;

Activate or not the geofences detection

property **toTxt**:string ;

Import / export geofences in text format

By default, when you [Save the map in text format](#) the geofences are saved and can be recharged

property **Geofence**[index : integer]:TECBaseGeofence ;

Returns a geofence

TECBaseGeofence

property **TECBaseGeofence.Name**:string ;

Name of the geofence

property **TECBaseGeofence.Active** : boolean

Activate or not the geofence (enabled by default)

property **TECBaseGeofence.ActiveDuration** : integer

Maximum activation time (milliseconds) (default 0 infinite)

```
// Activate for 3 seconds
map.geofences.geofence[0].ActiveDuration := 3000;
```

property **Color** : TColor ;

Color of the area

property **HoverColor** : TColor ;

Color of the area when mouseover

property **Item** : TObject;

You can use this property to store your data

property **Tag** : integer;

You can use this property to store your data

property **TECBaseGeofence.Shapes** : TECShapesList

The list of items that are currently in the geofence

Événements Geofences

To respond connect you on **OnEnterGeofence** and **OnLeaveGeofence** of **TECNativeMap** events.

```
...
map.OnEnterGeofence := mapEnterGeofence;
map.OnLeaveGeofence := mapLeaveGeofence;
...
procedure TForm1.mapEnterGeofence(sender: TObject; const
Geofence: TECBaseGeofence; const item: TECShape);
begin
caption := 'item '+inttostr(item.id)+' enter
in '+Geofence.Name;
end;

procedure TForm1.mapLeaveGeofence(sender: TObject; const
Geofence: TECBaseGeofence; const item: TECShape);
begin
caption := 'item '+inttostr(item.id)+'
leave '+Geofence.Name;
end;
```

The detection is done when a [shape](#) is moved, the test is performed on location (latitude and longitude) not on the actual surface of the object.

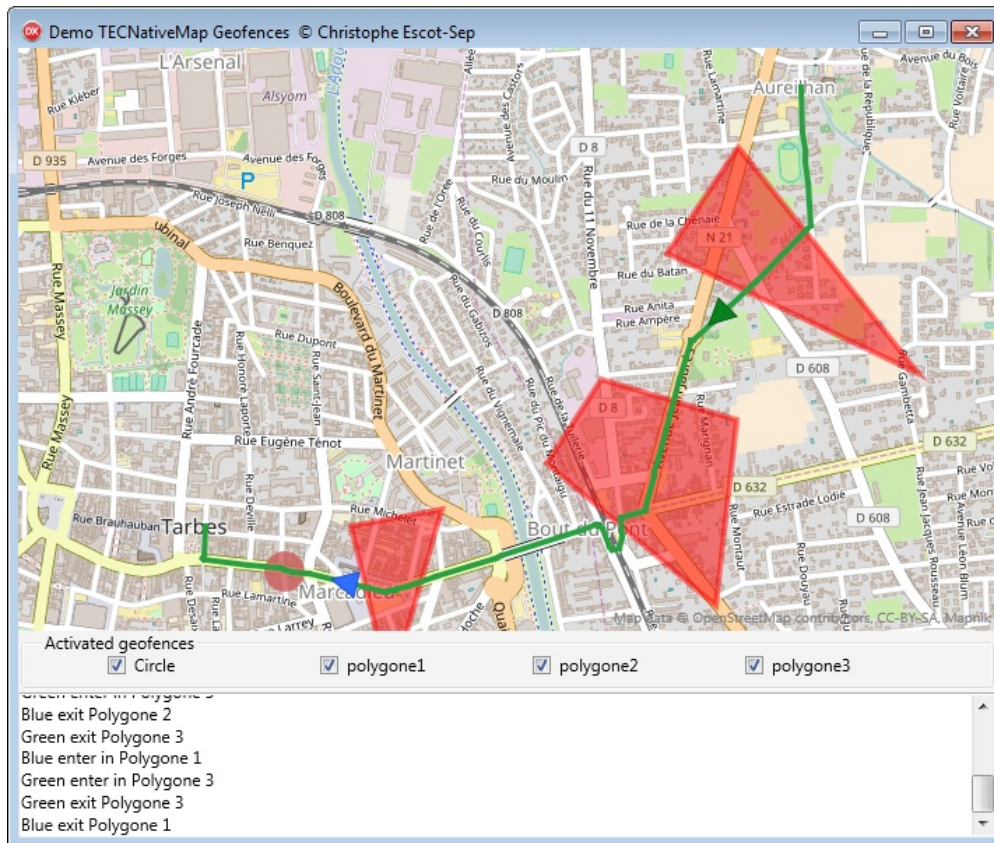


Fig. 82 Demo Geofences

Boundary

Boundary allows to obtain information and the polygon of an area by passing a geographical point.

function

Administrative(const lat,lng:double;Level:integer=0):boolean;

Find the administrative area for which we spend the level (from 2 to 10), with 0 level is determined according to the zoom

Returns true if found

```
map.boundary.Administrative(43.2332,0.0736,8); //
Level 8 = town
```

```
map.boundary.Administrative(43.2332,0.0736); //
Level in function of zoom
```

property **AdminLevelFromZoom**:string

Allows to control the administrative level according to the zoom

By default :

Zoom 0 - 4 = Level 2

Zoom 5 - 8 = Level 4

Zoom 9 - 11 = Level 6

Zoom 12 - 24 = Level 8

```
// only Level 4 et 8
map.Boundary.AdminLevelFromZoom := '0-8=4,9-24=8';

// Pass an empty string to reset to default values
map.Boundary.AdminLevelFromZoom := '';
```

function **Filter(const lat,lng:double;const AreaFilter:string):boolean;**

Generic function that allows you to set the area find.

```
// like administrative level 8
map.boundary.Filter(43.2332,0.0736,'"admin_level"="8"')
);
```

property **Id**:int64;

ID OSM of the found area

property **Tags**:TStringlist

Contains the list of tags OSM of the area, in the form key = value

property **OverPassUrl**:string;

OverPass is used to find the area (by default <https://overpass-api.de/api/interpreter>)

For the polygon from the id of the area, it connects to polygons.openstreetmap.fr

To change that assign a procedure to **GetPolygoneFromID**

```
// set nil for use polygons.openstreetmap.fr
map.Boudary.GetPolygoneFromID := myGetPoygonFromId;

procedure TForm.myGetPoygonFromId(const id:int64;var JSON:
string);
begin

    // here retourne yours polygons in json format

    JSON := ...
end;
```

See DemoNativeBoundaryArea for a demo

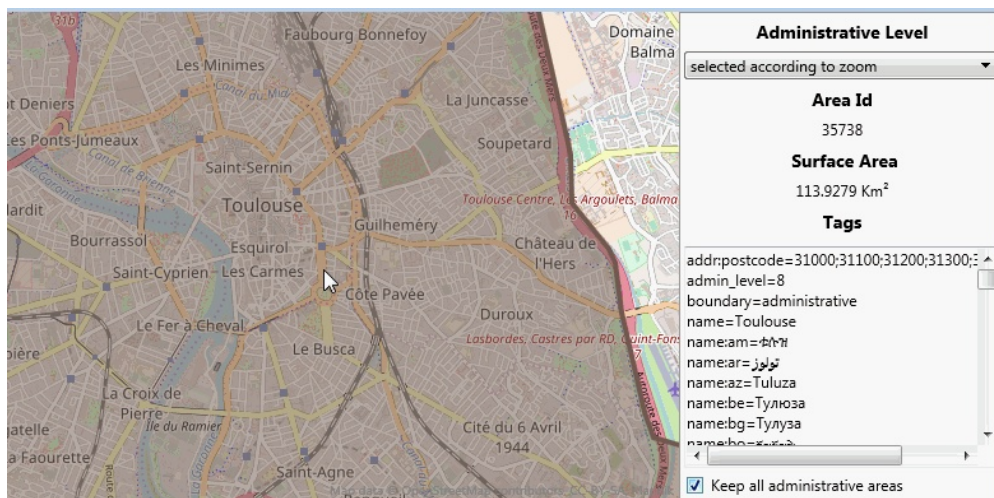


Fig. 83 DemoNativeBoundaryArea

GeoHash

Geohash is a public domain geocoding system invented by Gustavo

Niemeyer, which encodes a geographic location into a short string of letters and digits..

More information on [wikipedia](#).

procedure **TNativeMapControl.GeoHash.Decode**(const geohash: string; var latitude, longitude: double);

```
var lat, lng : double;

map.geoHash.Decode( '9xj5smj4w40m', lat, lng );
```

function **TNativeMapControl.GeoHash.Encode**(const latitude, longitude: double; const precision: Integer=12): string

```
// return the geoHash for the center of map
s := map.geoHash.encode(map.latitude, map.longitude);
```

function **TNativeMapControl.GeoHash.CardinalPoints**
(const geohash: string): TCardinalGeoHash;

```
// get geoHashs located at the 8 cardinal points around a geoHash

var cgh : TCardinalGeoHash;

cgh := map.geoHash.CardinalPoints( '9xj5smj4w40m' );

cgh.North;
cgh.NorthEast;
cgh.East;
cgh.SouthEast;
cgh.South;
cgh.SouthWest;
cgh.West;
cgh.NorthWest;
```

procedure **TNativeMapControl.GeoHash.MoveTo**
(const geohash: string);

```
map.GeoHash.Moveto('9xj5smj4w40m');
```

Open Location Code

Open Location Code (OLC) is a system of geocoding identifying an area anywhere on the Earth. It was developed at Google's Zurich Office.

More information on [wikipedia](https://en.wikipedia.org/wiki/Open_Location_Code).

function **TNativeMapControl.OpenLocationCode.Encode**(const latitude,longitude : Double;const codeLength:Integer=10):string;

```
// Provides a normal precision code, approximately 14x14 meters.
olc := map.OpenLocationCode.encode(lat,lng);

// Provides an extra precision code, approximately 2x3 meters.
olc := map.OpenLocationCode.encode(lat,lng,11);
```

function **TNativeMapControl.OpenLocationCode.Decode**
(const code:string):TecOLC_CODEAREA;

```
var olcArea : TecOLC_CODEAREA;

olcArea := map.OpenLocationCode.Decode('8FM263JF+PM');

olcArea.latitudeLo;
olcArea.longitudeLo;
olcArea.latitudeHi;
olcArea.longitudeHi;
olcArea.latitudeCenter;
olcArea.longitudeCenter;
olcArea.codeLength;

// zoom to the area
map.fitbounds(olcArea)
// draw polygone
map.AddPolygone(olcArea);
```

procedure **TNativeMapControl.OpenLocationCode.MoveTo**
(const OpenLocationCode:string);

```
map.OpenLocationCode.MoveTo( '8FM263JF+PM' );
```

*In the **uecOpenLocationCode.pas** unit you will find a class to use **OpenLocationCode** regardless of **TECNativeMap**, this is my translation of the **openlocationcode.js** of Google*

TECNativeMap allows to save and reload all of its data in a simple text file, this includes not only the component settings and views but also the map data

Alternatively, you can import/export your data in formats [GPX](#) , [GeoJSON](#) and [KML](#)

Backup/restore

function **SaveToFile**(const filename:string):boolean;

Saves the map to a text file.

Use extensions .gpx, .json, .kml to specify a format, otherwise it is the internal format of ECNativeMap that will be used

function **LoadFromFile**(const filename:string):boolean;

Charge la carte avec un fichier texte

Use extensions .gpx .kml to specify a format, otherwise it is the internal format of ECNativeMap that will be used

LoadFromFile can download files on internet

property **toGPX** : string;

Property **read/write** which gives access to the data of the card in GPX format.

property toTxt : string;

Property **read/write** which gives access to the data of the card in a text format.

property **ToKml** : string;

Property in **reading / writing** which returns the geographical elements of the map in [Kml format](#)

property **toGeoJson** : string;

Property **read/write** which gives access to the data of the card in the GeoJson format.

You access to the "properties" with **TECShape.Properties** field data, see chapter on the [vector tiles](#) to learn how to styling your items.

This property is used by **SaveToFile**, **LoadFromFile**.

Events

During loading the map the event **OnLoadShapes**(sender: TObject; const ShapeType:string; const index, max: integer; var cancel: boolean) fires for each item added.

ShapeType can have the values '[TECShapePOI](#)', '[TECShapeMarker](#)', '[TECShapeLine](#)', '[TECShapePolygone](#)' and '[TECShapeInfoWindow](#)'.

Index indicates the number of the currently loaded element and **Max** the total number of items to be loaded.

You can cancel the load switching **cancel** to true

When the file was loaded event **OnLoad**(sender: TObject;const GroupName:string;const FinishLoading : boolean) is triggered

GroupName contains [the name of the Group](#) that has been loaded,

empty for direct loading of the map (default group)

FinishLoading true if the file has been loaded in totality and false if it was abandoned by switching **cancel** to true

OSM XML

You can directly load a file to the OSM XML format or olt with `LoadFromFile`.

The groups also have the functions `LoadFromOSMStream` and `LoadFromOSMString`.

TOSMFile (uecOSM unit) allows you to work with files in the [OSM XML format](#) (base format of OpenStreetMap)

You can download parts of the world in this format on [geofabrik.de](#), take **.osm.bz2** and unzip them

But you can also get these data directly from TECNativeMap using OverPass API

OverPassApi

The OverPassApi property allows you to use the [Overpass API](#) to get data from OpenStreetMap.

By default the server used is **<https://overpass-api.de/api/interpreter>**,

use the `Url` property to change it

```
map.OverPassApi.Url :=
  'https://overpass.kumi.systems/api/interpreter';
```

Queries are nonblocking, plug on the event **onData** to recover data when available.

Use **Query** to run your request.

```
map.OverPassApi.OnData := doOverPass;
// get highway
map.OverPassApi.Query(
  'way({{bbox}})[highway][name];(._>);out;');

// the event is triggered when data is available
procedure TForm.doOverPass(const value:TECOverPassData);
begin
  // load OSM data in your map
  map.Shapes.LoadFromOSMString(value.Data);
end;
```

*You can use **{{bbox}}** to limit the search to the visible area of your map, and **{{center}}** to indicate the focal point of your map.*



See for more information on the scripting language of OverPassAPI :

wiki.openstreetmap.org/wiki/Overpass_API/Language_Guide

wiki.openstreetmap.org/wiki/Overpass_API/Overpass_QL

Some queries are available as standard

StreetNames returns just a list containing the name of the streets

```
procedure StreetNames(const bbox:string="";const
OnDataEvent:TECOnOverPass=nil); overload;
```

Use this version to set a different search in the current view box

```
procedure StreetNames(const SWLatitude,SWLongitude,NELatitude,NELongitude:double;const
OnDataEvent:TECOnOverPass=nil); overload;
```

```
map.OverPassApi.StreetNames;
```

Streets Returns the OSM data for the chosen area streets (it is equivalent to the example above)

```
procedure Streets(const bbox:string="";const OnDataEvent:TECOnOverPass=nil); overload;
procedure Streets(const SWLatitude,SWLongitude,NELatitude,NELongitude:double;const
OnDataEvent:TECOnOverPass=nil); overload;
```

Amenity to find specific points

```
procedure Amenity(const amenity_value:string="";const bbox:string="";const
OnDataEvent:TECOnOverPass=nil); overload;
procedure Amenity(const SWLatitude,SWLongitude,NELatitude,NELongitude:double;const
amenity_value:string="";const OnDataEvent:TECOnOverPass=nil); overload;
```

```
// find all 'restaurant' in visible area
map.OverPassApi.Amenity('restaurant');
```

Data Returns the set of OSM data for the specified area

```
procedure Data(const bbox:string="";const OnDataEvent:TECOnOverPass=nil); overload;
procedure Data(const SWLatitude,SWLongitude,NELatitude,NELongitude:double;const
OnDataEvent:TECOnOverPass=nil); overload;
```

```
map.OverPassApi.data;
```

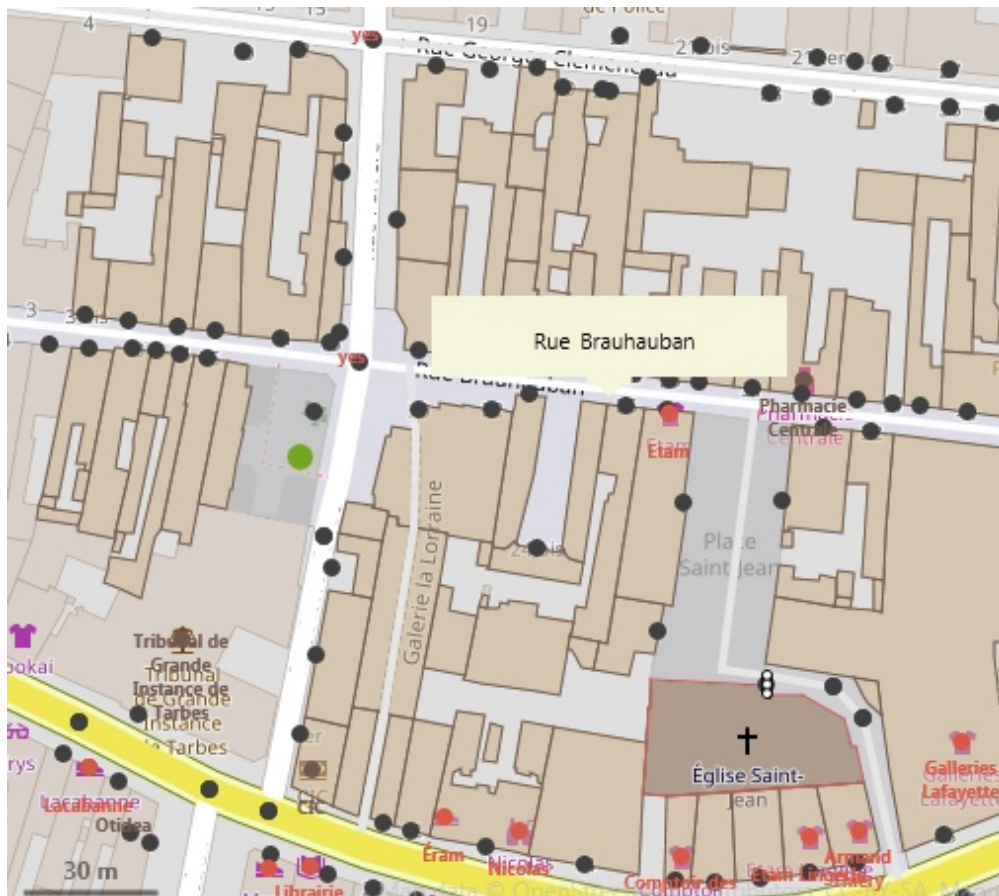


Fig. 85 OverPassApi.Data

OSM XML is large, some files are several Giga, TOSMFile can use an internal format (.olt) up to 19 times lighter, 10 times faster to process.

To convert a .osm file in .olt use the procedure **OSMFileToOLT(const**

OSMFilename:string;const Notify:TNotifyEvent=nil);

The conversion is performed in a thread and Notify is called when the work is complete.

```

procedure TForm2.SaveOLTClick(Sender : TObject);
begin
    // doEndSaveToOLT is call when file is saved
    OSMFileToOLT(FOSMFile.filename, doEndSaveToOLT);

end;

procedure TForm2.doEndSaveToOLT(Sender : TObject);
begin
    // here your osm file is saved in .olt

end;

```

Use TOSMFile to convert portions of OSM files

TOMSFile

function **LoadFromFile**(const sFilename : string) : boolean;

function **LoadFromStream**(const OSMStream : TStream) : boolean;

function **LoadFromString**(const OSMDData : string) : boolean;

Loads a OSM or OLT file, the treatment takes place in a Thread so it does not block your program

property **FilterPrimitive** : TSetPrimitiveOSM ;

There are [nodes](#), [ways](#) and [relations](#).

```

// handle all primitive
FOSMFile.FilterPrimitive := [poNode,poWay,poRelation];

```

If you want to know simply the area geographical covered by the file without loading the data, filter all !

```
FOSMFile.FilterPrimitive := [];
```

property **FilterBounds** : TFilterBounds;

Determines the geographical areas that interest us, data outside are ignored.

```
FOSMFile.FilterBounds.Clear;
FOSMFile.FilterBounds.Add(Areal_NorthEastLat,
Areal_NorthEastLng,
Areal_SouthWestLat, Areal_SouthWestLng);
FOSMFile.FilterBounds.Add(Area2_NorthEastLat,
Area2_NorthEastLng,
Area2_SouthWestLat, Area2_SouthWestLng);
```

property **FilterNode** : TListFilterOSM;

property **FilterWay** : TListFilterOSM;

property **FilterRelation** : TListFilterOSM;

Filters items based on their keys and values, filters down from the class
TFilterOSM

ExcludeKeyValue(*const Key, Value : string*),
OnlyKeyValue(*const Key, Value : string*),
ExcludelfKeyExist(*const Key:string*) and **OnlyIfKeyExist**(*const Key:string*) *simplifies the creation of filters.*

```
FOSMFile.FilterWay.clear;
FOSMFile.FilterWay.ExcludeIfKeyExist('building');
FOSMFile.FilterWay.ExcludeKeyValue('leisure','park');
FOSMFile.FilterWay.ExcludeKeyValue('natural','coastline');
```

function **ReLoad** : boolean;

Reload a file after changing the filters, the keys and values are not recharged allows to earn a few seconds.

```
procedure saveToFile(const Filename : string; NotifyEvent :
TNotifyEvent = nil);
```

You can save in the proprietary .olt format or in [GeoJSON format](#)

```
// save in geojson
FOSMFile.SaveToFile('path_myfile.geojson');
// save in olt
FOSMFile.SaveToFile('path_myfile.olt');
```

To save in a thread add a NotifyEvent

```
procedure TForm2.SaveOLTClick(Sender : TObject);
begin
// doEndSaveToOLT is call when file is saved
FOSMFile.SaveToFile('path_myfile.olt',doEndSaveToOLT);
end;

procedure TForm2.doEndSaveToOLT(Sender : TObject);
begin
// here your osm file is savec in .olt
end;
```

```
property Nodes : TNodeList read FNodeList;
property Ways : TWayList read FWayList;
property Relations : TRelationList read FRelationList;
```

Lists of the elements contained in the file after filtering.

Each list has functions for performing research on these items.

Find(idElement) Returns an item based on its id

The various **Search** functions fill the SearchResult list with the elements eligible

```
function Search(const Lat, Lng, radiusKM : double) : integer; overload;
```

Find all items located in an area centered around the point Lat, Lng and radius radiusKM

function **Search**(const SWLat, SWLng, NELat, NELng : double) : integer; overload;
 Finds all elements in the area.

function **Search**(const SWLat, SWLng, NELat, NELng : double; const Key, Values : string) : integer; overload;
 Finds all elements in the area with Key containing the specified values.

function **Search**(const Key, Values : string) : integer; overload;
 Find all items whose key contains values.

```
// search all nodes with amenity=car or amenity=bar
or amenity=restaurant
FOSMFile.Nodes.Search('amenity','cafe|bar|restaurant');

// search all ways with highway=residential
or highway=secondary
FOSMFile.Ways.Search('highway','residential|secondary');
```

function **Search**(const Key : string) : integer; overload;
 Find all items with this key, regardless of the value.

procedure **Clear**;

procedure **Abort**;
 Abandons a loading

procedure **ToShapes**
 (Shapes:TECShapes;NotifyEvent:TNotifyEvent=nil);
 Loads data into a group.

```
// load in default group
FOSMFile.ToShapes(map.Shapes);
```

procedure **FindToShapes**
 (Shapes:TECShapes;NotifyEvent:TNotifyEvent=nil);
 Loads the search data in a group.

```
// search all nodes with amenity=car or amenity=bar
or amenity=restaurant
FOSMFile.Nodes.Search('amenity','cafe|bar|restaurant');

// search all ways with highway=residential
or highway=secondary
FOSMFile.Ways.Search('highway','residential|secondary');

// load in default group
FOSMFile.FindToShapes(map.Shapes);
```

function **CountKey**(Node: TBaseOSM):integer;

Returns the number of **tag** for an element

function **getKey**(Node:TBaseOSM;const index:integer):string;

Returns the key which is the index

function **ReadKey**(Node : TBaseOSM; const KeyName : string) :
string; overload;

Returns the value of the Key

function **ReadKey**(Node : TBaseOSM) : string; overload;

Returns the set of key/value pairs in the form:

Key1=value1#13#10Key2=value2#13#10Keyx=ValueX

function **ReadValuesForKey**(const KeyName : string; valuesList :
TStringList): integer;

ValuesList filled with all of the values for a Key

property **Aborted** : boolean;

Indicates if loading has been interrupted by **Abort**

property **Bounds** : TBoundsLatLng ;

Contains the geographical area covered by the file

property **FileFormat**: TFileTypeOSM;

Returns the file type (ftOSM ou ftOLT)

property **Filename** : string ;
property **FileSize** : int64;

property **Keys** : TUniqueStringList;
List of all of the Key from the file

property **Values** : TUniqueStringList;
List of all the Value of the file

property Role[Member : TMemberOSM]:string;
Returns the [role](#) of a member of a [relation](#)

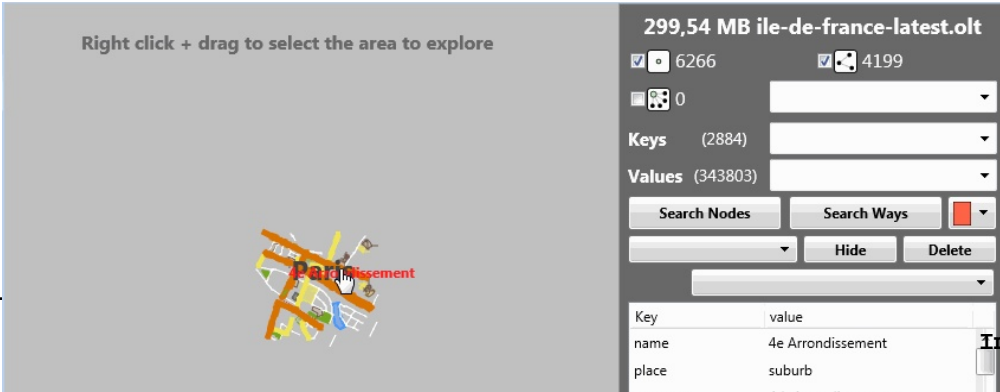
property **Timestamp** : string ;
Returns the date of the last update of the file

property **OnRead** : TOnOSMBlockRead;
To be able to process files of several Giga reading is made by block, this event is raised every time a block is read.

property **OnLoaded** : TNotifyEvent;
Raised when the entire file has been processed.

Check out **DemoOSMViewer** to see how to use *TOSMFile*

OSMViewer



Convert OSM in OLT

Open the osm file in OSMviewer, the area covered is displayed on the map

Check the primitives that you want to export (nodes, ways and relations), and then click **Save OLT** to start the conversion

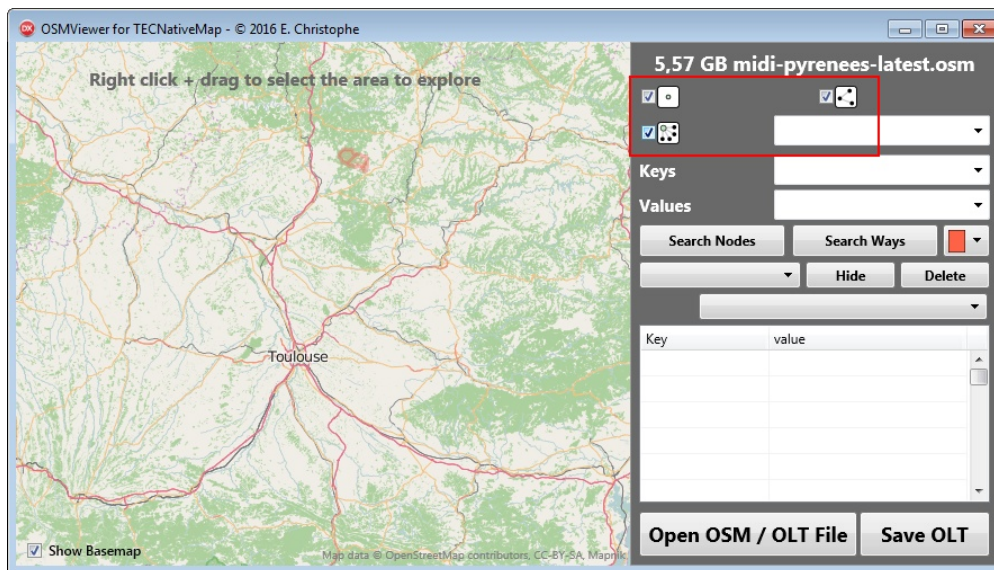


Fig. 87 Select nodes

Depending on the size of the OSM file work may take several minutes.

To export the ways all the nodes are kept in memory , you

need 1 MB of memory per Giga of file.

If you convert only the nodes you can work with 100 Giga files even if you have only 8 MB of memory.

Click right and while holding the button move the mouse to define an area to explore.

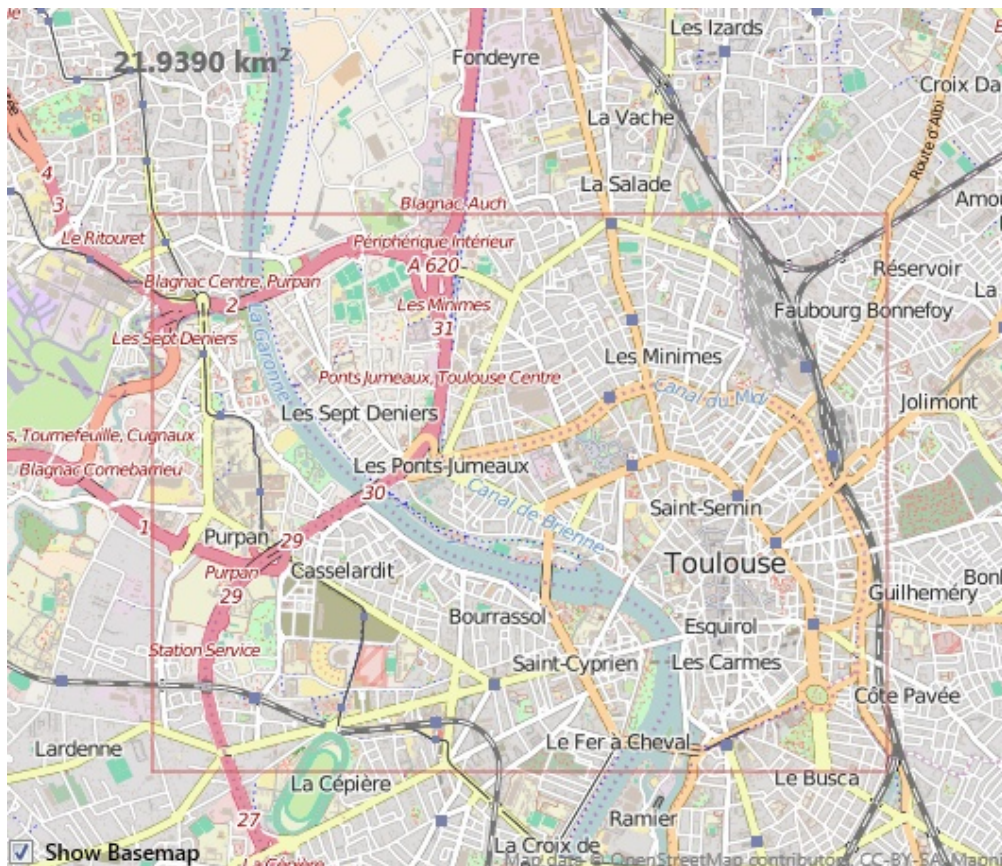


Fig. 88 Selecting an area

Once the loaded zone you will have the choice between displayed data on the map or save them in .olt

By loading data you can search according to key/value pairs

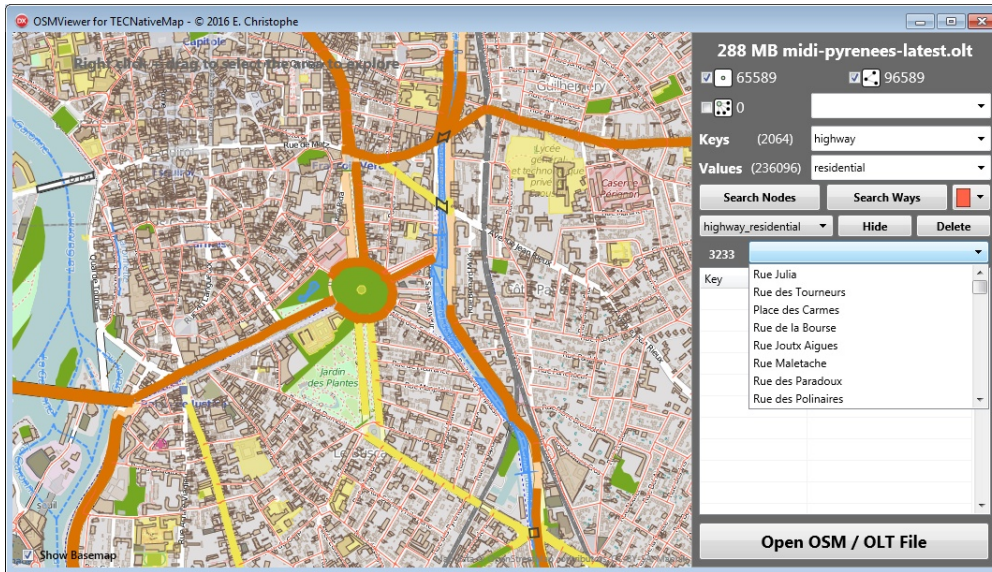


Fig. 89 search highway=residential

Download OSMViewer

You can [download an apk for android](#) to a mini demo that displays a small area OSM data.



Image capture

You can save the viewable area of the screen with the **ScreenShot** property that is an TBitmap

```
// Delphi map component TECNativeMap

// save map to bmp file
map.Screenshot.SaveToFile('c:\mymap.bmp');
```

Alternatively, you can capture any geographic area, even off screen, with the property **ScreenShots**

```
// procedure call when bitmap ready
map.ScreenShots.OnScreenShot := doScreenShot ;

// you can change the size of the bitmap between each
capture (default 800x600)
map.ScreenShots.Width := 2000;
map.ScreenShots.Height:= 2000;

// capture area centered on a point with a specified zoom
map.ScreenShots.Screenshot(latitude,longitude,16,
'optional_name_of_capture');

// you can take screenshots even if the previous is not
over yet

// Take the best zoom to a specific area
map.ScreenShots.Screenshot(NorthEastLat, NorthEastELng,
SouthWestWLat, SouthWestLng,'optional_name_of_capture');

// Take the best zoom to an area defined by its center
and radius in km
// CAUTION use double for the radius, here 1.0 km,
otherwise it will be mistaken for a simple zoom

map.ScreenShots.Screenshot(Latitude, Longitude, 1.0 ,
'optional_name_of_capture');

// when capture is ready, you go here
procedure TForm.doScreenShot(const name:string;const
```


The Photographer demo for Firemonkey shows you an example of using, you run 8 "drones" to photograph the ground every x meters.



With the VCL or Firemonkey you can work in high resolution by adjusting the property **ScaleFactor**

```
// "retina" mode
map.ScaleFactor := 2.0
```

If you work in a doubled resolution, also double the size of the images of your markers and set their Scale property to 0 so that they are automatically updated in the correct scale.

```
map.ScaleFactor := 2.0;

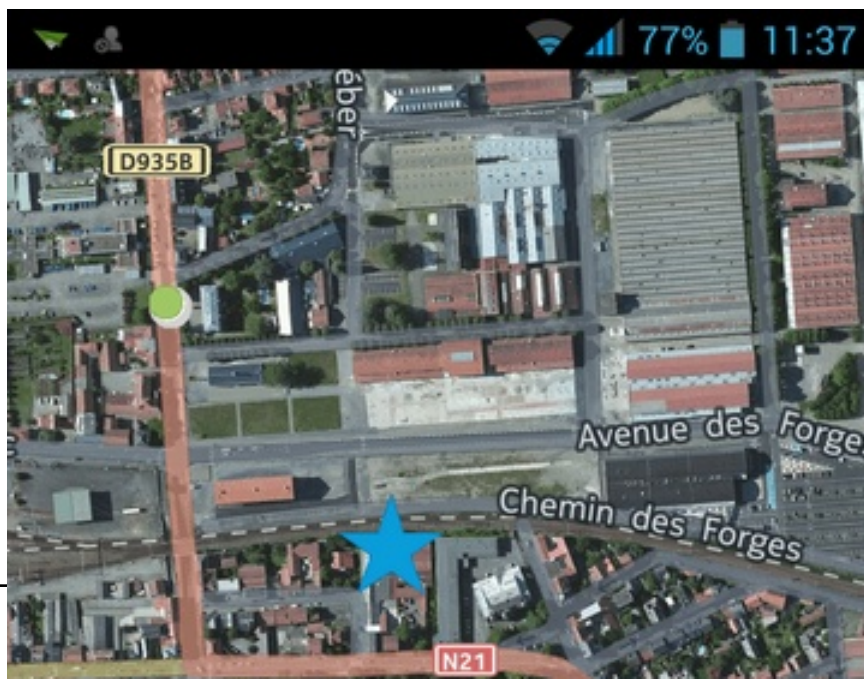
marker := map.AddMarker(latitude,longitude);
marker.Filename := 'image-64x64.png';
marker.Scale     := 0;
```

Firemonkey version allows you to activate the "High resolution" mode

```
map.HiRes := true;
```

For a better visibility of your maps it is preferable to use 512 x 512 tiles

```
map.HiRes := true;
map.TileServer := tsHereHybrid;
map.TileSize := 512;
```



If you have only 256 pixels tiles you can still force the size in 512 pixels, they will be automatically doubled, but the result is not optimal.

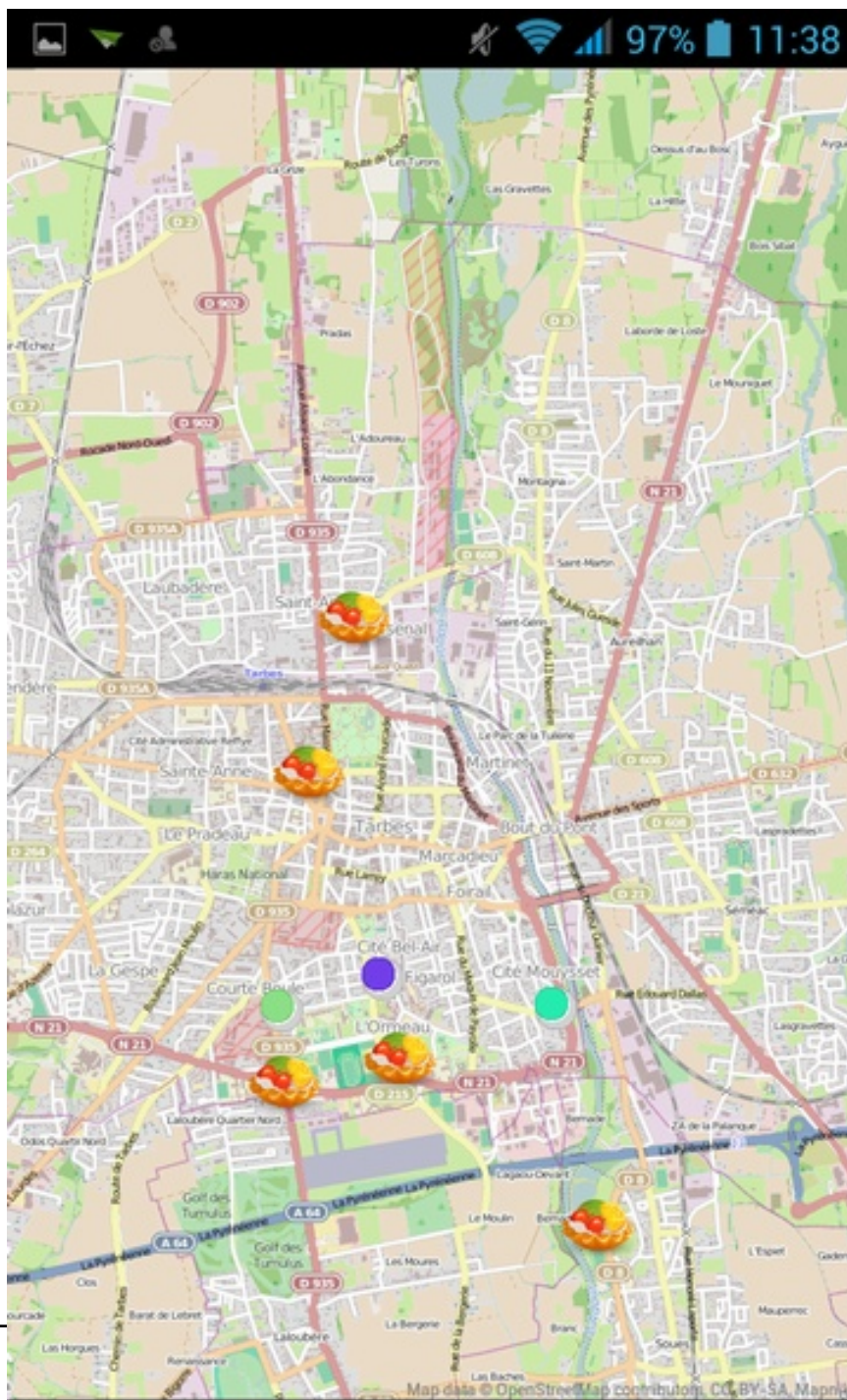
```
map.HiRes      := true;  
map.TileSize   := 512;  
map.TileServer := tsOSM;
```



If you keep tiles 256 pixels in high mode resolution your card will be finer but less readable.

The advantage is that a larger area will be visible.

```
map.HiRes := true;  
map.TileSize := 256;  
map.TileServer := tsOSM;
```



The [markers](#) without images, the [TECShapePois](#) defined in pixels, the [InfoWindos](#) and the [layer Panoramio](#) are automatically scaled.

The [layer TECNativePlaceLayer](#) has properties to suit the image resolution

For the Markers with image need to adapt the properties **XAnchor**, **YAnchor** , and **Filename** resolution (at least it takes double the size of the image in high resolution)

To use the shapes you need to add the unit

uecNativeShape

All the elements positioned on the map are accessible through the property **Shapes** type **TECShapes**, they descend all the class **TECShape**

*when you add several items, use **BeginUpdate / EndUpdate**, otherwise the map is redrawn after each addition.*

```
// for optimisation
map.shapes.BeginUpdate;

map.Shapes.Markers.add(lat1,lng1);
map.shapes.Markers.add(lat2,lng2);
...
map.shapes.EndUpdate;
```

Styles

Instead of manually decorate each element you can define [rules to apply a style](#)

Groups (TECShapes)

A group maintains a set of elements, the Shapes property is the default group, you go to a group by the Group ['name'] property

You can also be accessed by an index through

the property Groups

```

for i:=0 to map.groups.count-1 do
  map.groups[i].Clear;

  // or by iterator

  Group : TECShapes;

for Group in map.Groups do
  Group.Clear;

```

procedure **Clear**

empty group

procedure **BeginUpdate**

blocks the drawing of the map, ensures best performance

procedure **EndUpdate**

Authorizes a repaint of the map

```

  // for optimisation
  map.Group[ 'group1' ].BeginUpdate;

  map.Group[ 'group1' ].Markers.add(lat1,lng1);
  map.Group[ 'group1' ].Markers.add(lat2,lng2);
  ...
  map.Group[ 'group1' ].EndUpdate;

```

procedure **fitBounds**

Displays the map so that all the elements of the group are visible in the best zoom

Pois, markers, lines and polygons *properties also have a fitBounds procedure*

procedure **SaveToFile**(const Filename: string)

Saves the group to a file, the format is determined by the extension .gpx, .kml or own to

TECNativeMap text format

function **LoadFromFile**(const Filename: string): boolean

Charge the group with the contents of a file, the format is determined by the extension [.osm](#), [.olt](#), [.gpx](#), [.kml](#) or own to TECNativeMap text format

The OnLoad event is raised when the content is fully available

function **LoadFromOSMStream**(const Stream: TStream): boolean;

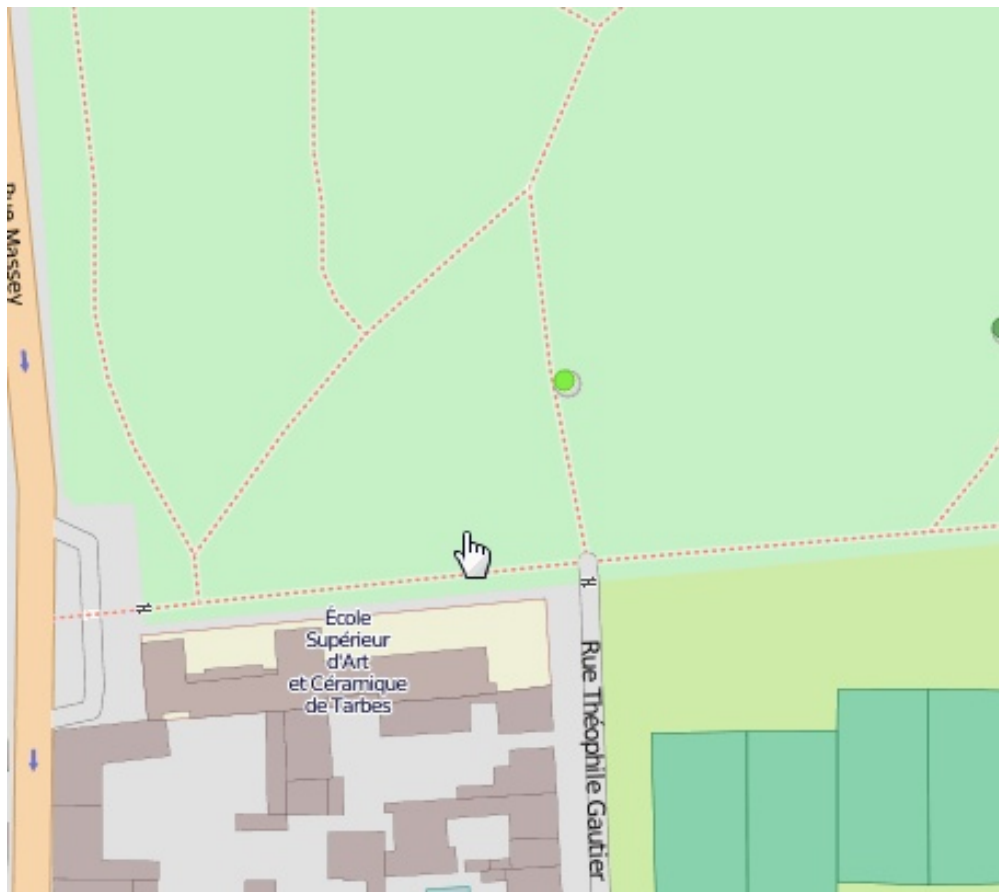
Charge the group with a stream that contains data in the OSM XML format or olt

function **LoadFromOSMString**(const data: string): boolean;

Charge the group with a string containing data at the OSM XML format or olt

property **Clusterable** : boolean

Active support of [Clusters](#), This allows you to group markers and pois who are near for better visibility



property **ClusterManager** : TECClusterManager

Managing clusters

property **Pois**: TECShapePOIList

List of elements of type [TECShapePOI](#)

property **Markers**: TECShapeMarkerList

List of elements of type [TECShapeMarker](#)

property **Lines**: TECShapeLineList

List of elements of type [TECShapeLine](#)

property **Polygones**: TECShapePolygoneList

List of elements of type [TECShapePolygone](#)

Liste des éléments de type [TECShapePolygone](#)

property **InfoWindows**: TECShapeInfoWindowList

Liste des éléments de type [TECShapeInfoWindow](#)

The lists have an iterator as groups

```
// Iterator for all list of shapes (markers, pois, lines,
polygones and infowindows )

var poly : TECShapePolygone;

for poly in map.shapes.polygones do
begin
    poly.color := clRed;
end;
```

property **MaxZoom**: byte

Maximum zoom level for which the items are displayed, above the group is not displayed

property **MinZoom**: byte

Minimum zoom level for which the items are displayed below the group does not appear

property **Zindex** : integer

Groups are displayed in ascending their ZIndex

function **TopZindex** : longint;

Returns the highest ZIndex of all the elements in the Group

TECNativeMap.TopGroupZindex *Returns the highest ZIndex of all groups*

property **Clickable** : boolean

Indicates whether all the elements meet the click of the mouse

The elements also have a clickable property

property **Serialize** : boolean

Indicates whether the Group should be saved when you save the entire map

Does not apply if we directly backup group

```

'''
'''
map.Group['group1'].Serialize := true;
map.Group['group2'].Serialize := false;
'''
'''
map.SaveToFile(filename);    // group2 not save
'''
'''

```

property **Visible**: boolean

Display/Hide group

property **Show**: boolean

Show/Hide group but unlike visible items remain accessible to the mouse

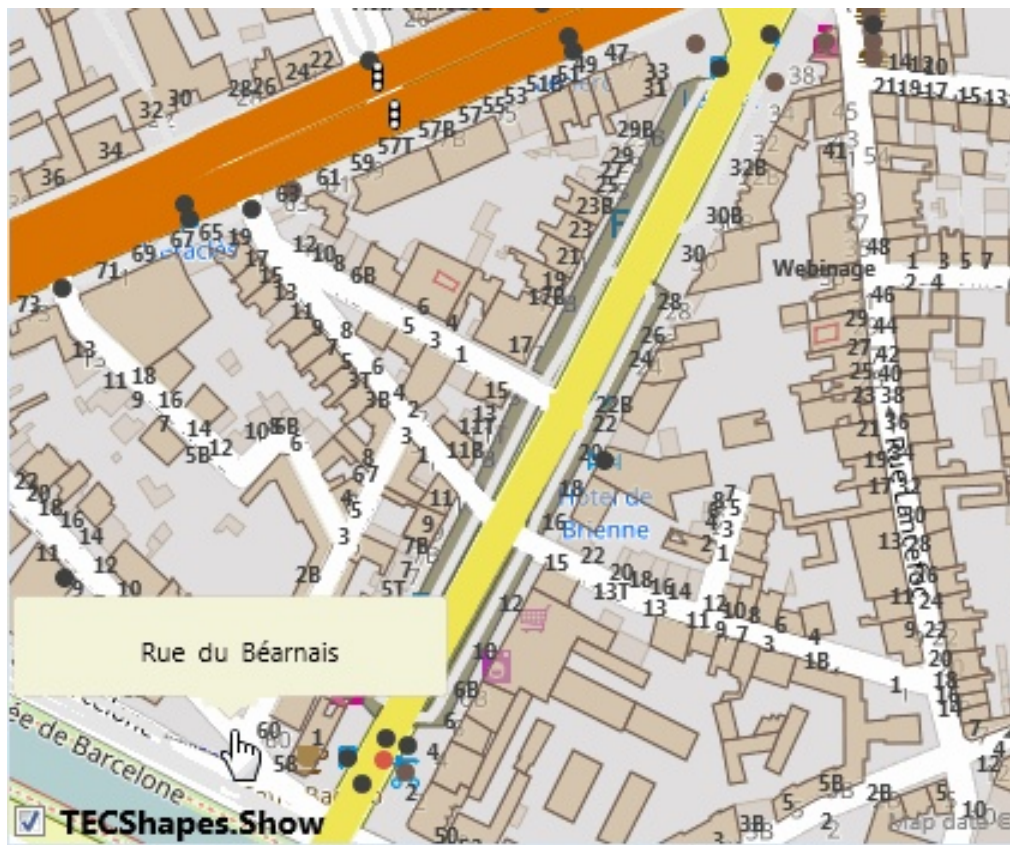


Fig. 96 TECShapes.Show

property **ShowHint**: boolean

Allows the display of the property **Hint** of the elements in the form of a bubble when hovering the mouse over

property **Name**: string

name of the group read-only

property **ToTxt**: string;

Import/Export in native format

property **ToGpx**: string;

Import/Export to Gpx format

property **ToKml**: string;

Import/Export to KML

Add Shapes

You can use the following shortcuts to add your items

```
function TNativeMapControl.AddPOI(const Lat, Lng: double; const
GroupName: string = "): TECShapePOI;
```

```
function TNativeMapControl.AddMarker(const Lat, Lng: double; const
GroupName: string = "): TECShapeMarker;
```

```
function TNativeMapControl.AddLine(const Lat, Lng: double; const
GroupName: string = "): TECShapeLine;
```

```
var line : TECShapeLine;
...
// create line
line := map.addLine(lat,lng);
// use this syntax for add line in group
// line := map.addLine(lat,lng,'your group');

// don't free ! or use remove
// line.remove

// to find your item in the list use indexof
// map.shapes.lines[line.indexof]
// map.group['your group'].lines[line.indexof]

// block repaint
line.beginupdate;
// add more points in line
line.add(lat2,lng2);
line.add(lat3,lng3);
// now of for repaint
line.endupdate;
```

```
function TNativeMapControl.AddLine(const dLatLngs: array of double;
```

const GroupName: string = "): TECShapeLine;

```
var line : TECShapeLine;
...
// create line
line := map.addLine([lat,lng,lat2,lng2,lat3,lng3]);
// use this syntax for add line in group
// line := map.addLine([lat,lng,lat2,lng2],'your group');

// block repaint, no need with addline but now yes
line.beginupdate;
// add more points in line
line.add(lat4,lng4);
line.add(lat5,lng5);
// now of for repaint
line.endupdate;
```

function **TNativeMapControl.AddPolygone**(const Lat, Lng: double;
const GroupName: string = "): TECShapePolygone;

function **TNativeMapControl.AddPolygone**(const dLatLngs: array of
double; const GroupName: string = "): TECShapePolygone;

function **TNativeMapControl.AddInfoWindow**(const Lat, Lng: double;
const GroupName: string = "): TECShapeInfoWindow;

*Do not destroy this shapes, they are kept in their
respective group lists*

TECClusterManager

The following properties allow you to change the display of the clusters

property **Color** : TColor

property **TextColor** : TColor

property **BorderColor** : TColor

property **BorderSize** : integer

property **FontSize** : integer

property **Opacity** : byte

property **WidthHeight** : integer

property **MaxPixelDistance** : integer

Items that are less than MaxPixelDistance of a cluster are grouped, 60 pixels by default

property **DrawWhenMoving** : boolean

Displays the clusters while moving the map defaults to true

property **MaxZoom** : byte

If the zoom is greater than **MaxZoom** elements are not grouped, default 18

property **OnAddShapeToCluster** : TOnAddShapeToCluster (sender : TECCluster; const Shape:TECShape;var cancel:boolean)

Raised when an item is added in a cluster, you can refuse by toggling cancel to true, default false

property **OnColorSizeCluster** : TOnColorSizeCluster (const Cluster : TECCluster;

var Color:TColor;var BorderColor:TColor;var TextColor:TColor,
var WidthHeight,FontSize:integer)

Raised before the display to allow you to adjust properties

property **OnDrawCluster** : TOnDrawCluster (const Canvas : TECCanvas; var rect : TRect; Cluster : TECCluster)

If you connect on this event you are supporting fully the cluster design

property **OnMouseOverCluster** : TOnNotifyEventCluster (const Cluster : TECCluster)

Triggered by the entry of the mouse on the cluster

property **OnMouseOutCluster** : TOnNotifyEventCluster (const Cluster : TECCluster)

Triggered by the release of the mouse in the cluster

example, change color depending on the number of elements contained by the cluster

```
map.Shapes.Clusterable := true;
map.Shapes.ClusterManager.OnColorSizeCluster
:= doOnColorSizeCluster;

procedure TForm.doOnColorSizeCluster(const Cluster
: TECCluster;
                                     var
Color:TColor;var BorderColor:TColor;var TextColor:TColor;
                                     var
WidthHeight,FontSize:integer
                                     );
begin

    if Cluster.Count< 10 then
    begin
        Color := clGreen;
    end

    else

    if Cluster.Count< 100 then
    begin
        Color := clBlue;
    end

    else

        Color := clRed;

    end;
```

Element TECShape

All displayable elements are descended from **TECShape**, they share the following properties

property **Clusterable**

allows not group this elements, true by default

```
map.shapes.markers[10].Clusterable := false;
```

property **Address**: string

Returns the address of the element, research is blocking

procedure **Location**

Geolocated the element, equivalent to **Address** but is not blocking

Raises **OnShapeLocation**

(item:TECShape,GeoResult:TECGeoResult;const Valid:boolean);

If the TECShape is not OnShapeLocation event is the OnShapeLocation of its [Group](#) that is raised.

Valid is true if the current position of the element is the same as that at the research.

```
map.shapes.OnShapeLocation := doShapeLocation;
// Asynchronous search address for Markers[0],

// when it is found map.shapes.OnShapeLocation is triggered
map.shapes.Markers[0].Location;
//
procedure TForm1.doShapeLocation(item:TECShape;const
GeoResult:TECGeoResult;const Valid:boolean);
begin
```



```

    if assigned(item) and Valid then
    begin
        // address
        GeoResult.address;
    end;

end;

```

function **DistanceTo**(Shape:TECShape)

Returns the distance in kilometres between the two elements

procedure **InfoWindow**(const content:string);

procedure **InfoWindow**(window:TECShapeInfoWindow);

Add an [infoWindow](#) to the element, it appears when you click on the element.

```

my_marker.InfoWindow('content for my marker');
// you can also use one infoWindow for many shapes
id := map.shapes.infoWindows.add(0,0,
'the same for all');
my_marker1.InfoWindow(map.shapes.infoWindows[id]);
my_marker2.InfoWindow(map.shapes.infoWindows[id]);
// for delete use nil
my_marker3.InfoWindow(nil);

```

procedure **Remove**

You can directly delete element

procedure **SetPosition**(const dLat, dLng: double)

Moves the item to latitude and longitude

procedure **SetDirection**(const dLat, dLng: double)

Moves the item to latitude and longitude, changes its angle of rotation so that it rotates in the

right direction

procedure **CenterOnMap**

Move the map so that the element is at the centre

function **IndexOf**: integer

Index of the item in its list (Pois, Markers, Lines ou Polygones)

function **ShowOnMap**: boolean

Indicates whether the item is in the visible part of the map

property **Animation** : TECShapeAnimation

Lets animate shape see [Animation](#)

property **Latitude**: double

Latitude of the element

property **Longitude**: double

Longitude of the element

property **Altitude**: double

Altitude of the element, you must have plugged a TECGeolocalise for your
TECNativeMap component

property **Color**: TColor

Color of the item

property **HoverColor**: TColor

Color on hover

property **Tag**: longint

You can use it freely

property **Visible** : boolean

Displays or hides the element

property **TrackLine** : **TECShapeLine**

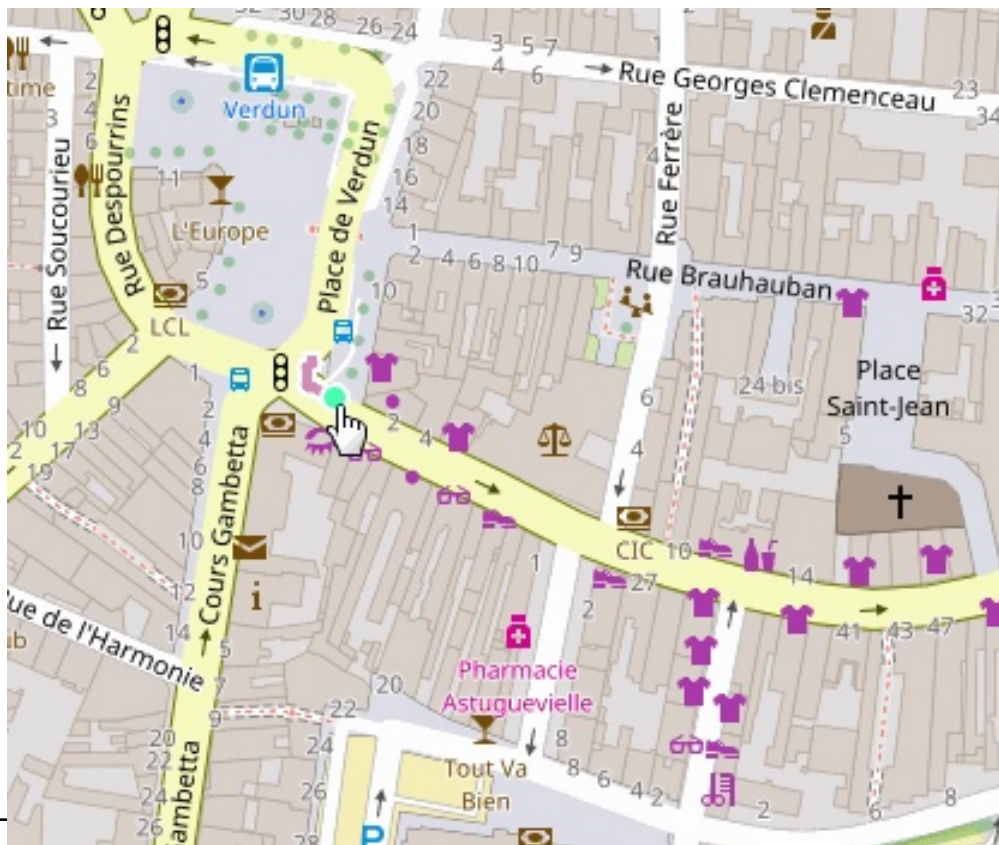
If TrackLine is set the track of the movement of the element will be generated

To activate it, simply use the property, you can also directly assign a line

```
FTracker := map.AddMarker(Latitude,Longitude) ;
// To activate the trace just access TrackLine
// line is created in the same group as FTracker
FTracker.TrackLine.visible := true;

// You can also pass a line
FTracker.TrackLine := your_line;

// for stop tracking set to nil , the line is free
FTracker.TrackLine := nil
```



property **isTrackLineActive** : boolean

Lets you know if a TrackLine is enabled.

If you test directly TrackLine it is automatically created if it does not exist !

property **Serialize** : boolean

Specify whether the item can be saved to a file (default yes)

property **Clickable**

Makes the reactive element to the mouse

property **ZIndex**

The items are displayed in ascending their ZIndex

BringToFront

Show element in foreground regardless of ZIndex

A single item at a time may be in the foreground, BringToFront does not change the display order which remains dependent of the ZIndex

SendToBack

Return to the normal display order

property **Draggable**

Makes a draggable element with the mouse

property **XAnchor** and **YAnchor**

Allows you to set the exact point inside the area of the element representing the Latitude and Longitude

property **Angle**

Angle of rotation in degrees (0-360)

Available for `TECShapeMarker` and `TECShapePOI`

property **Hint**: string

The text displayed in the bubble when the mouse hovers over the element, the parent group ShowHint property must be tripped to true

The text may be enriched with some html tags see

property **EnabledHint**

Enables / disables the display of the ToolTip information for the item

property **MaxShowHint**

indicates the number of times the information bubble will be displayed, you can reactivate it with EnabledHint or reloading MaxShowHint

property **PropertyValue[Name]**

allows you to access the value of the property **Name**

You can use `PropertyValue` to store your own data, this property is also used by vector tiles, the OpenStreetMap data is saved.

property **Properties**

List of Property/Value in the format "name: value", each pair is separated by a carriage return.

property **Group:** [TECShapes](#)

Group to which the item belongs

property **Selected**

Selects item, selected item appears as if it was over

The Selected property of the map contains all selected items

```
// you can use Iterator

var shape:TECShape

for shape in map.Selected do
begin
  ..
end;
```

TECSelectedShapesList

Type of the list of selected items, gives you access to

```
function ByArea(const
NEALat,NEALng,SWALat,SWALng :double):integer;

    Selects elements within the area bounded by NEALat,
    NEALng (top-right corner) and SWALat, SWALng
    (bottom-left corner)

function ByKMDistance(const
FLat,FLng,FKMDistance :double):integer;

    Selects elements within KMDistance (distance in km)
    to the point FLat,FLng

procedure UnSelectedAll

    Cleared all elements without deleting

procedure Clear

    Clears all selected items of the map

function count : integer;

procedure SaveToFile(const filename:string);

property Item[index: integer]: TECShape

    Returns the element whose is the index
```

property **ToTxt**:string

Returns the selected items in the internal text format of TECNativeMap (read-only)

property **GroupFilter** : TStringList

Allows you to filter based on the [groups of elements](#), leave empty to select items regardless of his group

property **OnChange**:TNotifyEvent ;

Triggered by adding/removing an item to the list of selected

Research

You have several functions to search in your items

function **FindShapeByArea**(const SWALat, SWALng, NEALat, NEALng: double;

const ShapeList: TList<TECShape>;

const FilterShapes :

TNativeShapes = [];

Filter:TOnShapeFilter=nil): integer;

ShapeList filled with items in a rectangular area which indicates the Southwest corners and Northeast

FilterShapes allows you to filter the wanted elements, by default all, example **[nsMarker,nsPoi]** to search only for TECShapeMarker and TECShapePOI

Filter allows to add a procedure to filter the search

```
// find the cafes located less than 2km

2mâp.FindShapeByKMDistance(map.Latitude,map.Longitude,
,liste,[nsMarker,nsPoi],FilterCafe);
...

procedure FilterCafe(const
  Shape: TECShape; var
  cancel: boolean);
begin
  cancel := shape.PropertyValue[
'kind']<>'cafe' ;
end;
```

```

function FindShapeByKMDistance(const FLat, FLng,
FKMDistance: double;
                                const
ShapeList: TList<TECShape>;
                                const FilterShapes :
TNativeShapes = [];
                                Filter:TOnShapeFilter=nil):
integer;
    ShapeList filled with items that are less than
    FKMDistance km from the point FLat,FLng

function FindShapeByFilter(const
ShapeList: TList<TECShape>;
                                const FilterShapes
: TNativeShapes;
                                Filter:TOnShapeFilter): integer;

```

All these functions return the number of items found.

For Delphi versions that do not support generics ShapeList is a simple TList

Events

Forms trigger the following events :

property **OnShapeMove** : TOnShapeMove

property **OnShapeDrag** : TOnShapeMove

property **OnShapeDragEnd** : TNotifyEvent

property **OnShapeMouseOver** : TOnShapeMouseEvent

property **OnShapeMouseOut** : TOnShapeMouseEvent

property **OnShapeMouseDown**
: TOnShapeMouseEvent

property **OnShapeMouseup** : TOnShapeMouseEvent

property **OnShapeClick** : TOnShapeMouseEvent

property **OnShapeRightClick**: TOnShapeMouseEvent

property **OnShapeDbIClick** : TOnShapeMouseEvent

property **OnShapeDbIClick** : TNotifyEvent

property **OnShapeLocation** : TOnShapeLocation

Events can be assigned individually to each element, in this case they are not passed to the level of TECNativeMap

```
map.shapes.Markers.add(lat1,lng1);
map.shapes.Markers.add(lat2,lng2);

// connect directly to the event
// OnshapeClick of TECNativeMap will not
be triggered for marker 0
map.shapes.markers[0].OnShapeClick
:= MarkerShapeclick;
```

Snap markers on a line or a polygon

You can set the [TECShapeMarker](#) and [TECShapePOI](#) which will automatically gravitate towards a [TECShapeLine](#) or a [TECShapePolygone](#) if the drop has some distance from these.

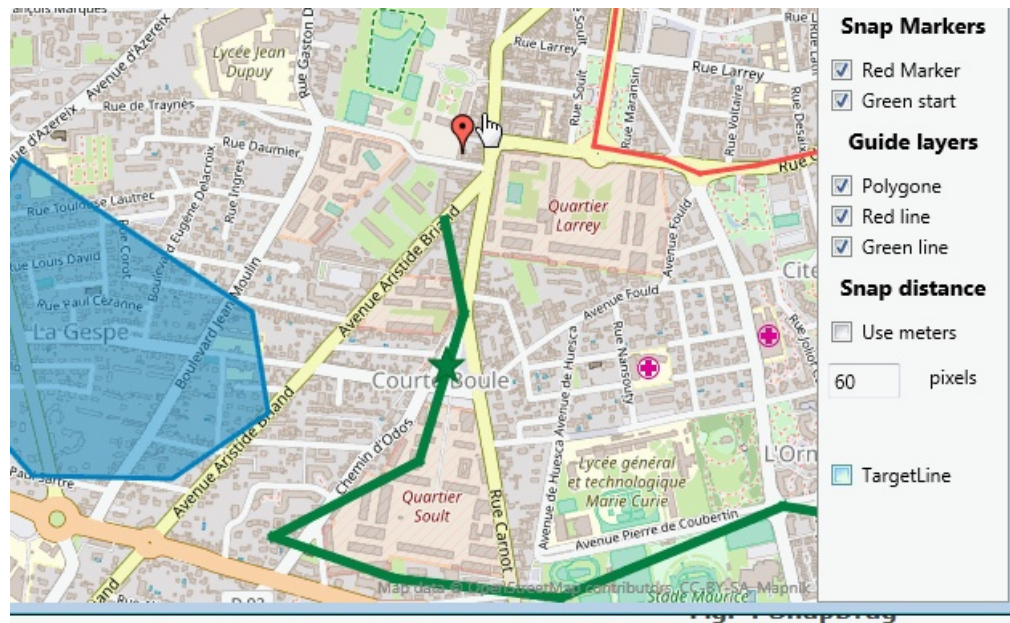


Fig. 98 SnapDrag

Manage you this through the **SnapDrag** of type **TECSnapDrag** of the mapping component **TECNativeMap** property.

procedure **ClearMarker**;

Removes all the snapping markers

function **AddMarker(const Marker:**
TECShape): boolean;

Adds a **TECShapeMarker** or **TECShapePOI** in the list of the snapping markers.

Returns true if the element is added.

procedure **RemoveMarker(const Marker: TECShape);**

Deletes the marker from the list of the snapping markers

procedure **ClearGuide;**

Remove all the TECShapeLine and TECShapePolygone of the list of guides

function **AddGuide(const Guide: TECShapeLine): boolean;**

Adds a TECShapeLine or a TECShapePolygone in the list of guides.
Returns true if the element is added.

procedure **RemoveGuide(const Guide: TECShapeLine);**

Removes the element from the list of guides

procedure **CancelSnap;**

Cancel the move and puts the marker to its point of origin

property **MeterDistance: boolean ;**

Selects the unit for distance triggering the magnetization.
true to meters, false (default) to pixels.

property **SnapDistance: integer ;**

Distance of attraction of the markers to a guide

property **SnapShape: TECShape ;**

Returns the marker that comes to be attracted.

property **SnapGuide: TECShape;**

Returns the guide coming to snap a marker

property **TargetLine: boolean;**

If true draws a line between the marker and the point target on the guide

property **OnSnap: TNotifyEvent;**

The event is raised when the a marker is attracted by a guide.

property **OnNoSnap: TNotifyEvent;**

The event is raised when the a marker is drop outside the catchment area of a guide.

```
map.SnapDrag.addMarker(mrk);

map.SnapDrag.AddGuide(poly);
map.SnapDrag.AddGuide(red_line);

map.SnapDrag.MeterDistance := true;
map.SnapDrag.SnapDistance  := 50;
// 50 meters

map.SnapDrag.MeterDistance := false;
map.SnapDrag.SnapDistance  := 50;
// 50 pixels
```

The property `_snap_` of the guides is set to true when it attracts a marker, you can use to define a style that will visually report the status.

```
// double the size of line when targeted
map.Styles.addRule(
'.line._snap_:true {scale:2}') ;
map.Styles.addRule(
'.polygone._snap_:true {scale:2}') ;
// default size
map.Styles.addRule(
'.line._snap_:false {scale:1}') ;
```

```
map.Styles.addRule(
  '.polygone._snap_:false {scale:1}') ;
// dotted target line
map.Styles.addRule(
  '#TECSnapDrag.line{penStyle:Dot}');
```

*If you want that markers are attracted to specific lines, you can create your own variable of type **TECSnapDrag***

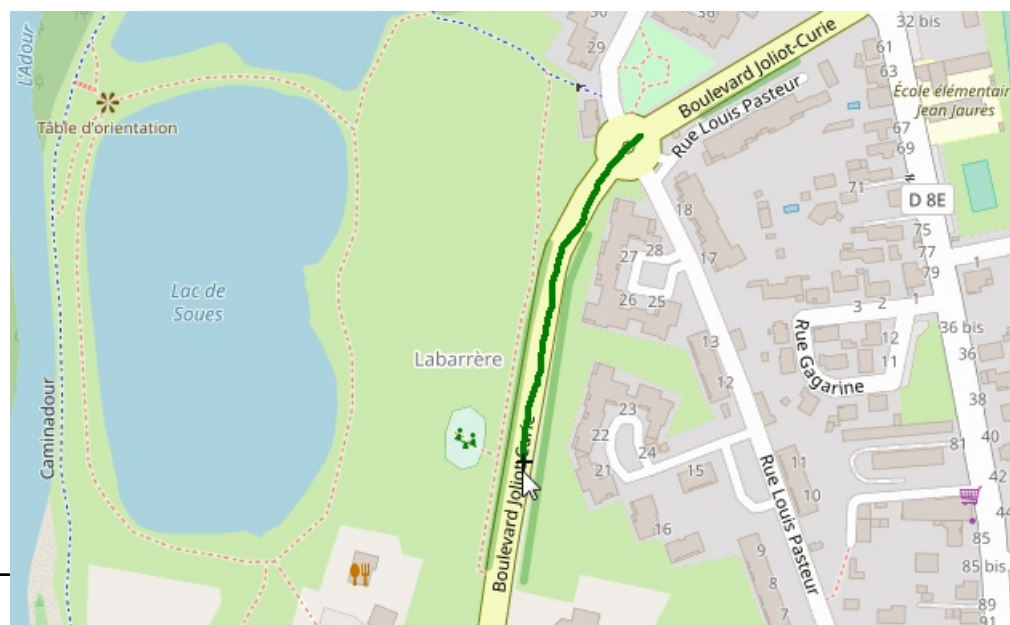
```
var MySnapDrag : TECSnapDrag;
...
MySnapDrag := TECSnapDrag.create;

MySnapDrag.addMarker(mrk);

MySnapDrag.AddGuide(poly);
MySnapDrag.AddGuide(red_line);
...
MySnapDrag.free
```

Drawing and free hand selection

FreeHand allows you to define lines, which can become polygons, to mouse, for it simply activate **FreeHand.Draw**



```

    // event triggered when the drawing is finished
    map.FreeHand.OnDraw := doFreeHandDraw;
    // activate the hands-free drawing
    map.FreeHand.Draw := true;
    ...
procedure
    TForm.doFreeHandDraw(Sender: TObject);
begin
    // save the line to your map

    map.addLine(TECNativeFreeHand(sender).line)
    // you can use also this syntax
    map.FreeHand.addLine;
    // convert the drawn line into a polygon
    and add it to the map

    map.addPolygone(TECNativeFreeHand(sender).line);
    // you can use also this syntax
    map.freeHand.addPolygone;
end;

```

You can also activate the free hand selection tool, allows you to choose items simply by surrounding

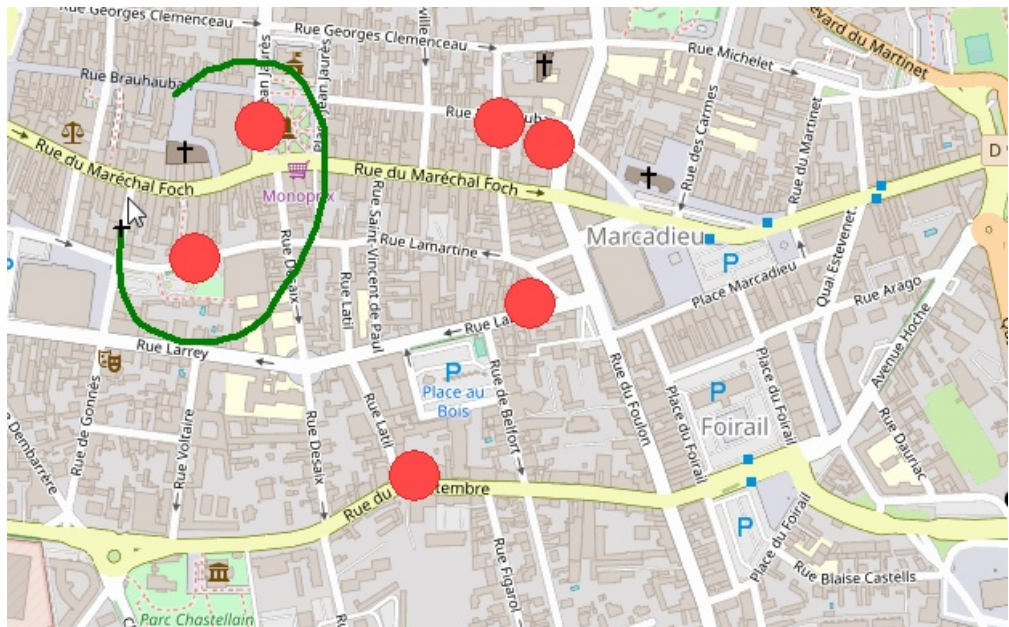


Fig. 100 Sélection main libre

```

// trigger after selection
map.FreeHand.OnSelection := doOnSelection;
// start sélection mode
map.Selection := true;
...
procedure
  TForm21.doOnSelection(Sender: TObject);
begin
  ' ShowMessage(' + Integer(TECNativeFreeHand(sender).SelectionList.count) + ' selected');
end;

```

property **AutomaticMapMovement** : boolean ;

Allows the automatic card moving when the drawing cursor approaches an edge, default True

property **Draw**: boolean ;

The drawing mode.

property **Selection**: boolean ;

The mode Selection.

Do not disable Draw after enabling Selection, Selection also active Draw !

property **MouseButton**: TMouseButton ;

The mouse button used to draw, by default mbRight.

*If you use the left button, it map may be moved by holding down the **CTRL** key.*

property **Cursor**: TECShape ;

Drawing slider, by default a TECShapePOI in the shape of a cross.

property **Line**: TECShapeLine;

The line that represents your drawing, you can adjust its properties either directly or by style.

```
// style for draw line
map.Styles.addRule(
'#TECNativeFreeHand.Line {color : green;penStyle:Dash;weight:2}');
```



Fig. 101 Style FreeHand Line

property **PolygoneSelection**: TECShapePolygone ;

Accessible in the OnSelection event, it represents the area of your selection.

property **SelectionList**: TECShapesList;

Accessible in the OnSelection event, this list contains selected items.

property **SelectionFilter**: TOnShapeFilter

Can set a filter for selection.

```
map.FreeHand.SelectionFilter := FilterCafe;  
...  
    // select only cafe  
procedure FilterCafe(const Shape: TECShape; var cancel: boolean);  
begin  
    cancel := shape.PropertyValue['kind'] <> 'cafe' ;  
end;
```

AddLine(const GroupName:string=""):TECShapeLine;

Add your trace permanently as a Line to your map

AddPolygone(const GroupName:string=""):TECShapePolygone;

Add your trace permanently as a Polygone to your map

TECShapeMarker

The TECShapeMarker are represented by an image markers, You can load a picture in 3 ways

Specifying a .png through the property file **Filename**

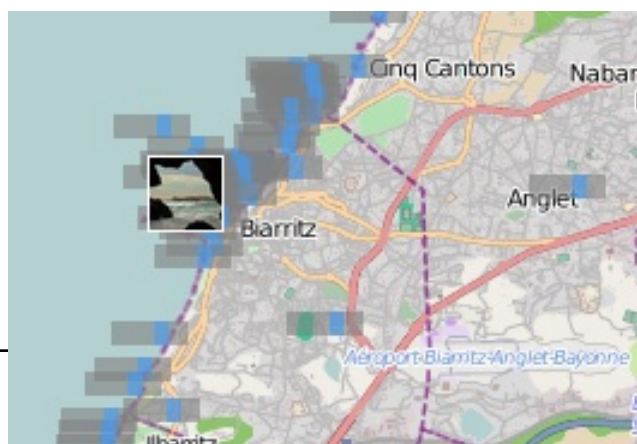
```
map.Shapes.Markers[0].filename :=  
;http://google-maps-icons.googlecode.com/files/restaurant.png'
```

Filename also accepts local files

Alternatively, you can directly specify a Base64-encoded image (Data URI)

```
const Data_Uri =  
'data:image/png;base64,iVBORw0KGgoAAAANSUHEUgAAAAUA  
AAAFCAyAAACNbyblAAAAHE1EQVQI12P4//8/w38GIAXDIBKE0DHxgljNBAAO  
9TXL0Y4OHwAAAABJRU5ErkJggg==';  
  
map.Shapes.Markers[0].filename := Data_Uri;
```

By default an animation indicates that the image is to be downloaded, you can delete this effect by switching the property **WaitAnimation** to false



In the **VCL version** you can assign a **TImageList** with PNGs to the **Icons** property of the **TECNativeMap**, then indicate the index of the image in the **Icon** of the **TECShapeMarker** property.

```
map.Icons := myImageList;  
// use icon number 1  
map.Shapes.Markers[0].Icon := 1;
```

You can also directly assign a **TPngImage** to the **Graphic** property from the **TECShapeMaker**

```
map.Shapes.Markers[0].Graphic := MyPngImage;
```

Images are shared between markers, if you assign the same file to several markers, there will only be one instance of the image, this limits memory consumption.

By default when you directly assign a TGraphic to the Graphic property it will not be automatically released when the marker is destroyed, you will have to do it yourself !

If the property OwnsGraphic of the marker is True then your TGraphic will be released when destroying the marker, but you will have to make

sure that do not share this image with other markers who also have OwnsGraphic at true !

You can specify the clickable area with the function **SetHitBox** (x,y,w,h)

X,Y indicates the top-left corner of the clickable area

W the width

H the height

The top-left corner of the marker is 0,0

Use the properties **XAnchor** and **YAnchor** to determine the specific point of the image corresponding to the latitude and longitude

Example to center the image on the geographical coordinates

```
map.Shapes.Markers[0].Graphic := MyPngImage;

// center on latitude,longitude
map.Shapes.Markers[0].XAnchor := map.Shapes.Markers[0].width div 2;
map.Shapes.Markers[0].YAnchor := map.Shapes.Markers[0].height div 2;
```

Default image

If you let **Filename** and **Graphic** empty, the marker is drawn using StyleIcon and the Color property.

StyleIcon

There are 6 styles for the markers without images



Fig. 103 StyleIcon si3d - siFlat - siFlatNoBorder - siDirection

If you have several tens of thousands of items to display, use the siFlatNoBorder style, it is the fastest to display

```
var mrk3d, mrkFlat, mrkFlatNB, mrkDirection : TECShapeMarker;

mrk3d := map.addMarker(lat1, lng1);
mrk3d.StyleIcon := si3D;
mrk3d.color      := claRed;

mrkFlat := map.addMarker(lat1, lng1);
mrkFlat.StyleIcon := siFlat;
mrkFlat.color     := claBlue;

mrkFlatNB := map.addMarker(lat1, lng1);
mrkFlatNB.StyleIcon := siFlat;
mrkFlatNB.color    := clagreen;

mrkDirection := map.addMarker(lat1, lng1);
mrkDirection.StyleIcon := siDirection;
mrkDirection.color    := clagreen;
// angle indicates the direction ( 0 = North, 180 South )
mrkDirection.angle    := 30;
```

*You can change its color on hover of the mouse with the property **HoverColor***

SVG

With Firemonkey you can also use SVG images, only the most common ones, but it is enough to display the icons of the [MAKI](#) project for example.



Fig. 104 Maki icons

```
mrk.Filename := 'local_path_or_url\bicycle-15.svg';

// You can also directly inject the SVG data

mrk.StyleIcon := siSVG;
mrk.Filename :=
'M7.49,15C4.5288,14.827,2.1676,12.4615,2,9.5C2,6.6,6.25'+
',1.66,7.49,0c1.24,1.66,5,6.59,5,9.49S10.17,15,7.49,15z';

// you can style like this
map.styles.addRule(
);marker {Graphic:HERE-SVG-DATA;StyleIcon:siSVG;color:red}'
```

*You can also use **siOwnerDraw** to draw your self your marker*

```
marker.StyleIcon := siOwnerDraw
marker.OnAfterDraw := doOwnerDraw;
..
procedure TForm1.doOwnerDraw(const canvas: TECCanvas; var rect:
TRect; item: TECShape) ;
var s:string;
begin
    // draw hint
```

```
canvas.TextRect(rect,0,0,item.hint);  
end;
```

Field of view

If the **Fov** (Field of view) property is greater than 0, a cone appears in the direction defined by the angle of the marker.

An angle of 0 corresponds to the North.

You can change the length of the cone with the **FovRadius** property.

```
mrk.fov := 40;  
mrk.FovRadius := 30;  
mrk.Angle := 180;
```



Fig. 105 field of view

Scale

The **Scale** property change the size of the marker.

```
mrk.scale := 1.5; // increases by 50%
mrk.scale := 0.5; // decrease of 50%
```

You can use `mrk.scale` to increase the size of the markers when the mouse passes over

```
map.styles.addRule('.marker:hover {scale:1.5;}');
```



Fig. 106 automatic scale

Adjust size to Zoom

Use the **ScaleMarkerToZoom** property to set the marker size changes depending on the Zoom.




```
map.ScaleMarkerToZoom := true;
```

OnBeforeDraw

This event is raised before the drawing of your marker, you can use it to add a background

```
// sample, add a circle in the background of the marker
// for all markers of default group
map.shapes.markers.OnBeforeDraw := doCircleMarker;
...
procedure TForm1.doCircleMarker(const canvas: TECCanvas;
var rect: TRect; item: TECShape) ;
begin
    // size border
    Canvas.PenSize := 3;
    // border color
    canvas.pen.Color := $FFEDED;

    if item.Hover or item.Selected then
        canvas.Brush.Color := item.Color
    else
        canvas.Brush.Color := item.HoverColor;

    canvas.Ellipse(rect.left - 8,rect.Top - 8,rect.Right+8
,rect.Bottom+8);
end;
```

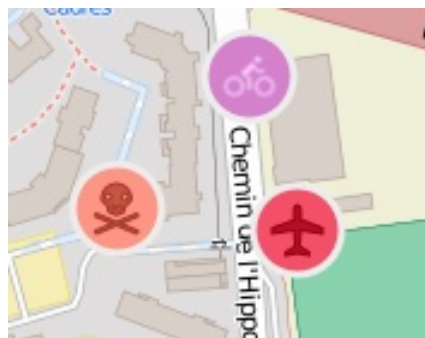


Fig. 108 OnBeforeDraw

GroundOverlay

You can constrain an image to cover a certain area

SetBounds(NorthEastLatitude,NorthEastLongitude,SouthWestLatitude,SouthWestLongitude)

You enable the recovery by flipping the property **fitBounds** to true;

```
// groundoverlay
i := map.Shapes.Markers.Add(40.712216,-74.12544);
map.Shapes.Markers[i].setBounds(40.773941,-74.12544,
40.712216,-74.22655);

map.Shapes.Markers[i].Filename      :=
';https://www.lib.utexas.edu/maps/historical/newark_nj_1922.jpg'
map.Shapes.Markers[i].fitbounds     := true;
```

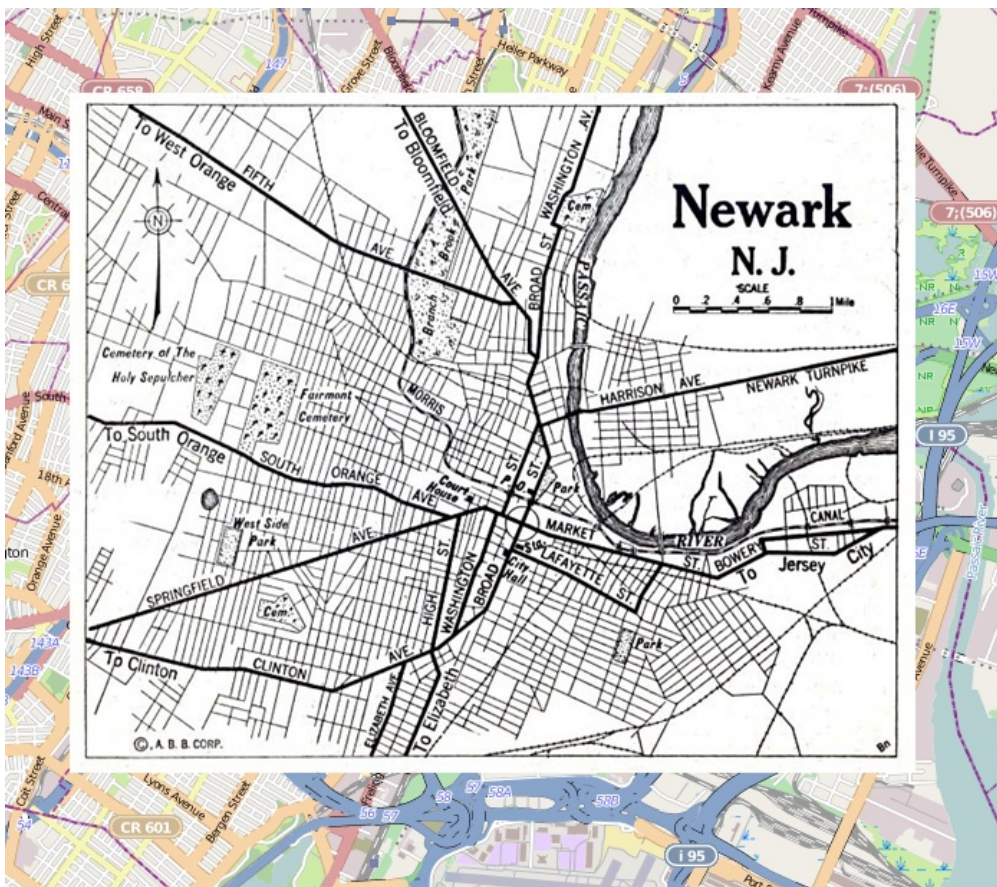


Fig. 109 GroundOverlay

Pois (point of interest) are equivalent to the [markers](#), the difference is that you have 9 predefined shapes (Text, Ellipse, rectangle, triangle, arrow, arrowhead, diamond, cross, diagCross, star and hexagon) but you can also take in charge you even their design.

By default the elements sizes are in pixels but you can also use meters by using the property **POIUnit**.

```
// Delphi map component EMap

// add POI at center of map
id := map.shapes.Pois.add(map.latitude,map.longitude);

// random form

case random(10) of
  0 : map.shapes.pois[id].POIShape := poiEllipse;
  1 : map.shapes.pois[id].POIShape := poiStar;
  2 : map.shapes.pois[id].POIShape := poiRect;
  3 : map.shapes.pois[id].POIShape := poiTriangle;
  4 : map.shapes.pois[id].POIShape := poiDiamond;
  5 : map.shapes.pois[id].POIShape := poiHexagon;
  6 : map.shapes.pois[id].POIShape := poiArrow;
  7 : map.shapes.pois[id].POIShape := poiArrowHead;
  8 : map.shapes.pois[id].POIShape := poiCross;
  9 : map.shapes.pois[id].POIShape := poiDiagCross;
 10 : begin
map.shapes.pois[id].POIShape := poiText;
        map.shapes.pois[id].Description :=
'Poi n°'+inttostr(id);
        end;
end;

map.shapes.pois[id].Draggable := true;

map.shapes.pois[id].Color := RGB(random(255),random(255)
),random(255))    ;

// set hint to this POI
map.shapes.Pois[id].Hint := 'my first POI!';

// by default the size of TECShapePOI is in pixel
// you cant use meter also

// map.shapes.pois[id].POIUnit := puMeter

map.shapes.pois[id].POIUnit := puPixel;
```

```
map.shapes.pois[id].width := 32;
map.shapes.pois[id].height:= 32;
```



Fig. 110 PoiShape

*By default the color of the border is derived from the main color, if you want to change this use **BorderColor** after **Color** !*

You can change the size of the shapes by flipping their **Editable** property to true

```

// Delphi map component EMap

// add POI at center of map
id := map.shapes.Pois.add(map.latitude,map.longitude);

map.shapes.pois[id].POIShape := poiStar;

map.shapes.pois[id].editable := true;

// map.shapes.pois[id].POIUnit := puMeter

map.shapes.pois[id].POIUnit := puPixel;

map.shapes.pois[id].width := 32;
map.shapes.pois[id].height:= 32;

```



Fig. 111 TECShapePOI in edit mode - resize handle displayed

TECMapPois

Pois are managed by an **TECShapePois** list accessible through the **pois** of the groups property **TECShapes**

the property **OnOwnerDrawPOI** : TOnOwnerDrawPOI specify a procedure to support the design of type **poiOwnerDraw**

```

// Delphi map component EMap

```

```

map.Shapes.Pois.OnOwnerDraw := doOwnerDrawPOI;
...

// owner draw poi, here transparency text
procedure TFormPoi.doOwnerDrawPOI(const canvas:TCanvas;var
Rect:TRect;item:TECMapPoi) ;
var x,y,w,h:integer;
begin

    canvas.brush.Style := bsClear;

    if item.Hover then
        canvas.font.color := item.HoverColor
    else
        canvas.font.color := item.color;

    canvas.font.Style := [fsBold];

    w := canvas.TextWidth(item.hint) ;
    h := canvas.TextHeight(item.hint);

    x := rect.Left+((rect.Right-rect.Left-w) div 2);
    y := rect.top+((rect.bottom-rect.top-h) div 2);

    canvas.TextOut(x,y,item.Hint);

end;

```

Each TECShapePOI element has its own property OnOwnerDrawPOI

You also have a property OnAfterDraw which allows to draw on a figure, example to include his number.

```

// Delphi map component EMap

map.Shapes.Pois.OnAfterDraw := doAfterDrawPOI;
...

// after draw poi, here write is number
procedure TFormPoi.doAfterDrawPOI(const canvas:TCanvas;var
Rect:TRect;item:TECMapPoi) ;

```

```
var x,y,w,h : integer;
    s : string;
begin

    canvas.font.style := [fsBold];

    s := inttostr(item.IndexOf);

    w := canvas.TextWidth(s) ;
    h:= canvas.TextHeight(s) ;

    x := 1+((r.Left + r.Right) - w) DIV 2 ;
    y := 1+((r.Top + r.Bottom) - h) DIV 2 ;

    canvas.brush.Style := bsClear;

    canvas.font.color := clWhite;

    canvas.TextRect(r,x,y,s);
end;
```

As for the [TECShapeMarker](#) the [Scale](#) property allows you to adjust the size and `ScaleMarkerToZoom` is also functional.

TECShapeLine allows you to plot a line consisting of a set of points on your card.

The lines are managed by an **TECShapeLines** list accessible through the property **Lines** of the [TECShapes](#) groups

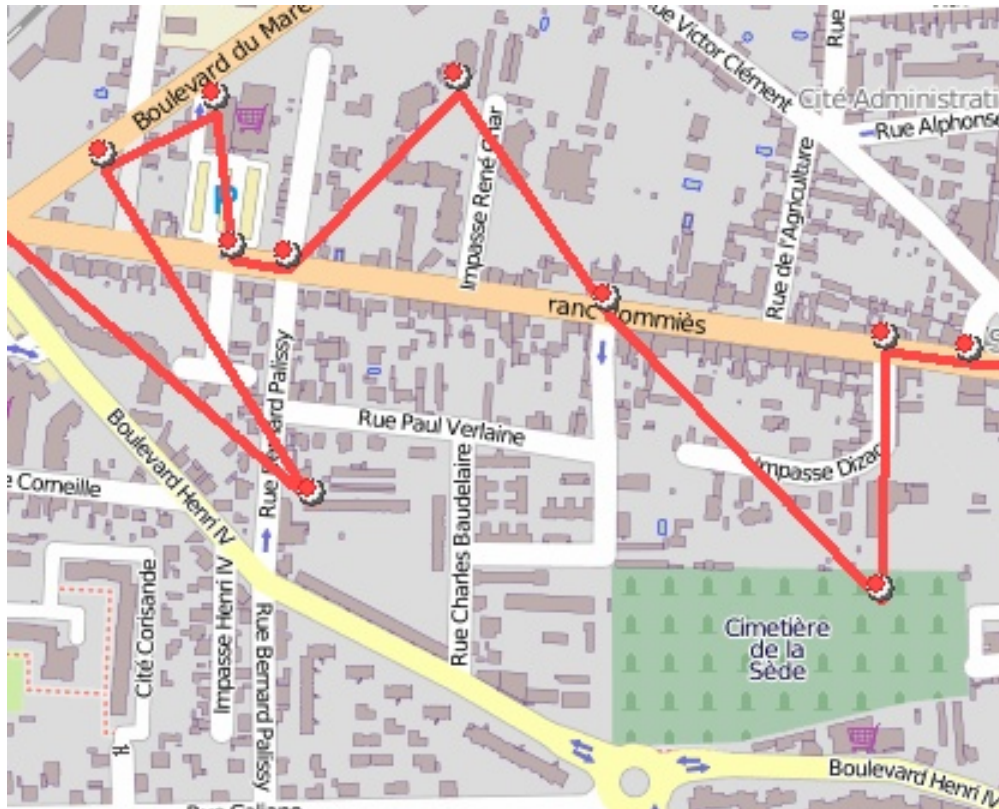


Fig. 112 A polyline in edit mode

TECShapeline

This class descends from [TECShape](#)

It additionally has the following properties

function **Add**(const Lat, Lng: double; const Alt: double =

_ErrorAltitude): integer;

Add a point with potentially altitude

procedure **Insert**(const index: integer; const Lat, Lng : double; const Alt: double = _ErrorAltitude);

function **Add**(const dLatLngs: array of double): integer;

Adds an array of points, consisting of a suite of latitudes and longitudes

```
line.add([lat1,lng1,lat2,lng2,...,latx,lngx]);
```

Unlike adding a simple point, you do need to manage the addition of your table by BeginUpdate / EndUpdate, is done automatically.

Inserts a point with potentially altitude

procedure **Reverse**

Reverse the direction of the line

property **ShowDirection** : boolean

Add arrows to each segment to show direction



Fig. 113 ShowDirection

```
procedure Slice(const StartIndex,EndIndex : integer;Line :
TECShapeLine); overload;
```

Copy the part bounded by the StartIndex and EndIndex segments in Line

```
function Slice(const
```

```
StartIndex,EndIndex:integer;GroupName:string=""):TECShapeLine; overload;
```

Returns a line composed of the portion bounded by the StartIndex and EndIndex segments

```
procedure Slice(const StartKm,EndKm : double;Line :
TECShapeLine); overload;
```

Copy the part starting at the StartKM kilometre and ending at the EndKM kilometre in Line

```
function Slice(const StartKm,EndKm :
double;GroupName:string=""):TECShapeLine; overload;
```

Returns a line consisting of the portion commencing at kilometre StartKM and ending at the EndKM kilometre



```
// red color between 1.2 km and 3.8 kilometer
line := map.shapes.lines[0].Slice(1.2,3.8);
line.Color := claRed;
```

procedure **Clear**;

Clears the points

procedure **Delete**(index: integer);

property **Encoded**: string ;

Encode / Decode the polyline by using the algorithm of google (
developers.google.com/maps/documentation/utilities/polylinealgorithm)

```
line.add([lat1,lng1,lat2,lng2,...,latx,lngx]);
```

property **EncodePrecision**: byte;

Accuracy of encoding, default 5 decimal places (180.00000 to -180.00000)

Removes a segment of the line

function **Count**: integer;

Indicates the number of segment

property **Weight**: byte read FWeight write setWeight;

property **BorderSize** : integer

Border thickness

property **BorderColor** : TColor

Color of the border

property **HoverBorderColor** : TColor

Border color when the mouse hovers over it



Fig. 115 2 pixel black border

Line thickness

property **PenStyle** : TPenStyle

stroke type **psSolid**, **psDash**, **psDot**, **psDashDot** and **psUserStyle** are available

Using **psUserStyle** and **setCustomDash**([len_dash,len_space,...,len_dashx,len_spacex]) you can create your pattern of traits

```
line.penStyle := psUserStyle;
line.SetCustomDash([4,4,2,4,4,4]);
//you can use in styles like this
map.styles.addRule(
) #_DRAGZOOM_.line {weight:4;color:green;penStyle:userStyle;customStyle:4,4,2,
```

You can animate the traits

property **LineType** : TECLineType

Type of line : **ItStraight** straight line between each point, **ItBezier** to draw bezier curves



Fig. 116 Penstyle := psDashDot - LineType := ltBezier

```
procedure getAltitudes(Event: TOnGetAltitude = nil);
```

Calculates the altitudes of the points, you can pass a procedure of type `TOnGetAltitude` to display a progress bar (see `DemoNativeRoute`)

```
// Delphi map component ECGap

// calcul altitude for all point of polyline 0
// you can pass nil if you don't show a progressbar
```

```

map.Shapes.Lines[0].GetAltitudes(doGetAltitude);
...
{ *
    event fired by getAltitudes

    @param Sender TECShapeLine
    @param Total number of altitude's point calculated
    @cancel flag for abort calcul
}
procedure
    TFDemoRoute.doOnGetAltitude(Sender: TECShapeLine;
const Total:integer;var cancel:boolean);
begin
    ProgressAltitude.Position := total;
    // cancel if press button
    cancel                      := btAbortAlt.tag = -1;
end;

```

function **getLatLngFromMeter**(const SensStartEnd: boolean;
const IMeter: longint; var dLatitude, dLongitude: double;
var idPoint: integer; var Heading: integer; var bEnd: boolean): boolean;

Calculates the latitude and longitude of a point on the line/polygon based on its distance in metres, returns **True** if a point was found

SensStartEnd meaning of the course, true for departure -> arrival

IMeter the distance in metres

dLatitude,dLongitude variables of type **double** who will receive the latitude and longitude

idPoint a variable that will contain the index in the array **Path** where is located the point, the calculation returns an approximation because your polyline/polygon has all of the actual points

Heading a variable that will contain the angle of the point relative to the North (from 0 to 360 °)

bEnd indicates whether one has reached or exceeded the end of polyline/polygone(or early depending on the direction)

property **HoverPoint**: integer read FHOverSeg;

Index of the segment pointed by the mouse

property **Path**[index: integer]: TECPointLine

Array containing all the points of the line

function **Distance** : double;

Indicates the total distance (km) of the line

property **Duration** : integer;

If the row represents a route (route) indicates the duration of the trip.

property **NorthEastLatitude**: double

Latitude of the upper right corner of the box enclosing the line

property **NorthEastLongitude**: double

Longitude of the upper-right corner of the box enclosing the line

property **SouthWestLatitude**: double

Latitude of the lower left corner of the box enclosing the line

property **SouthWestLongitude**: double

Longitude Latitude from the lower left of the box enclosing the line

```
// Delphi map component EMap  
  
line := map.Shapes.Lines[0];  
  
// show the entire line  
map.fitBounds(line.NorthEastLatitude, line.NorthEastLongitude, line.SouthWestL
```

property **ShowText** : boolean

If the row represents a road displays the intermediate points

property **Editable** : boolean

property **Shapes** : [TECShapes](#)

If the row represents a road provides access to the forms that indicate the steps

The departure and arrival points are represented by [TECShapePOI](#), the other intermediate points are [TECShapeMarker](#)

The example below enables to change the display of the points of departure and arrival.

```
// Delphi map component EMap
// change default poi start and finish of route
var line : TECShapeLine;
...
line := map.shapes.lines[0];

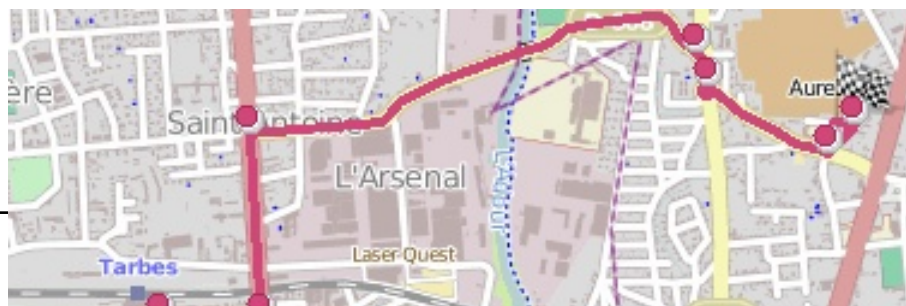
if line.shapes.Pois.count > 1 then
begin
  i := line.shapes.markers.add(line.shapes.pois[0].latitude, line.shapes.pois[0].longitude);
  line.shapes.markers[i].filename :=
    'http://www.helpandweb.com/flag_blue.png';
  line.shapes.markers[i].hint := line.shapes.pois[0].hint;

  i

17. latitude, longitude) := line.shapes.pois[line.shapes.Pois.count - 1].latitude,
line.shapes.pois[line.shapes.Pois.count - 1].longitude);
  line.shapes.markers[i].XAnchor := 0;
  line.shapes.markers[i].filename :=
    'http://www.helpandweb.com/checkered_flag.png';

  line.shapes.markers[i].hint
:= line.shapes.pois[line.shapes.Pois.count - 1].hint;

  // hide default start and finish
  line.shapes.pois[0].visible := false;
  line.shapes.pois[line.shapes.Pois.count - 1].visible
:= false;
end;
```



property **Editable** : boolean

Makes it editable line, removes it a double click on a point, a double click on a segment adds a point, the points are Draggable mouse

This allows that straight paths, to modify a route it takes just 2-3 lines of additional code.

By default the selection points are represented by a black square to the starting point, a white square to the end point and a white circle for the intermediate points.

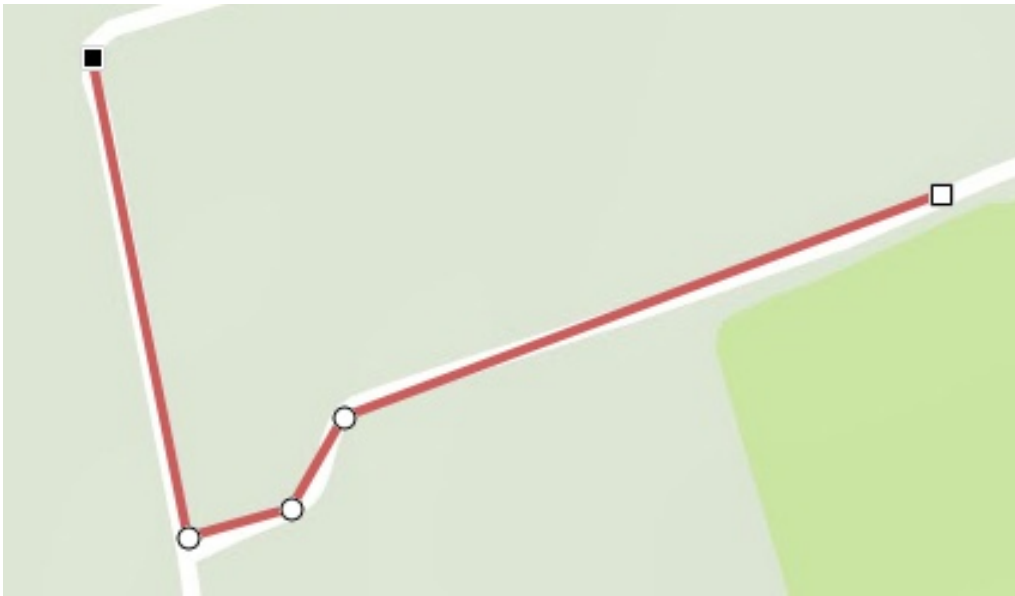


Fig. 118 Edit line

Use the properties **FilenameStartEditLine**, **FilenameEndEditLine**, **FilenamePointEditLine** of your map to change the bitmaps

```
// Reverse the point of departure and arrival
s := map.FilenameStartEditLine;
map.FilenameStartEditLine := map.FilenameEndEditLine;
```

```
map.FilenameEndEditLine := s;
```

You can change the creation of the selection points using the property
OnCreateShapeLinePoint

```
line := map.add(nsLine,Lat,Lng);

// change défaut point
line.OnCreateShapeLinePoint
:= doCreateShapeLinePointEditable;

procedure TForm1.doCreateShapeLinePointEditable(sender:
TObject; const Group:TECShapes;
TECShape;
var ShapeLinePoint:
TECShape;

const Lat,Lng:double;const index:integer);
var i:integer;
poi:TECShapePOI;
begin

    i := Group.Pois.add(lat,Lng);
    Poi := Group.Pois[i];

    ShapeLinePoint := poi;

    poi.Width := 10;
    poi.Height:= 10;

    poi.POIShape := poiRect;

    poi.Color      := claWhite;
    poi HoverColor  := claWhite;
    poi.BorderColor := claBlack;
    poi HoverBorderColor:= claBlack;

end;
```



property **OnShapePathChange** : TNotifyEvent

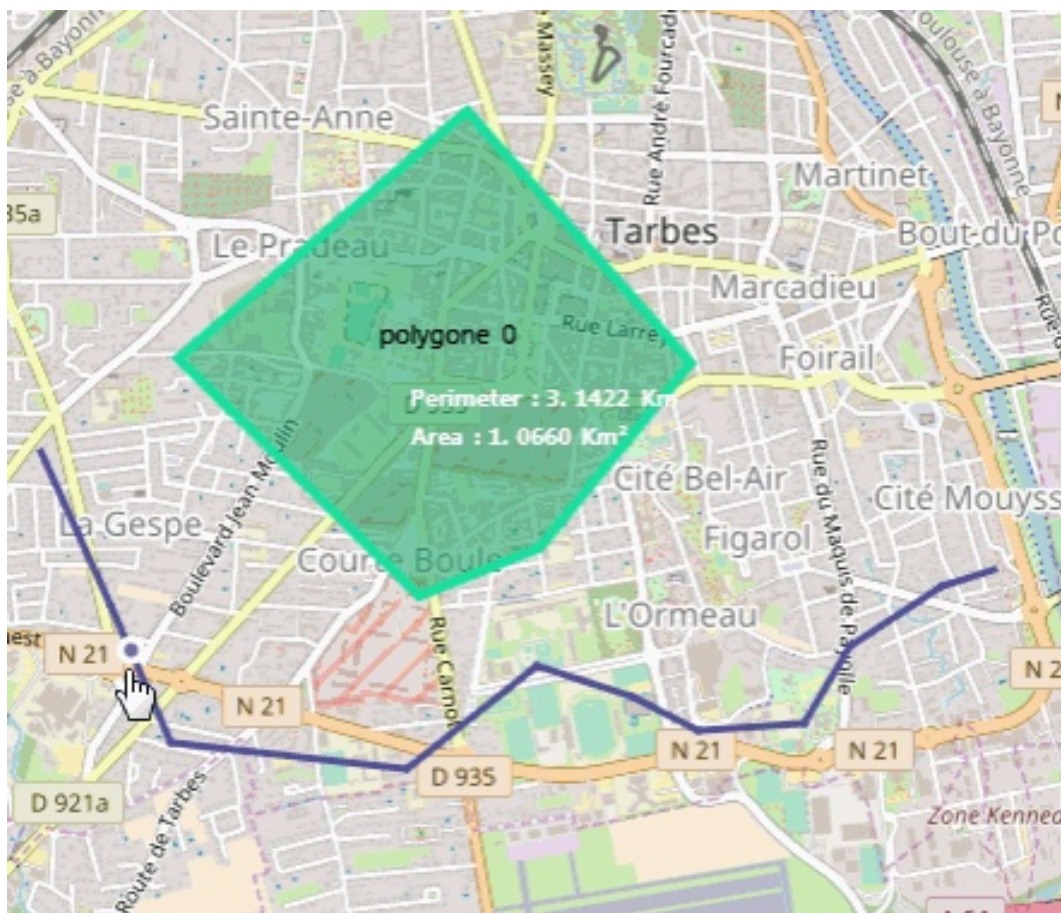
To be informed when the line changes.

Select a portion of the line

To select a portion of your line with the mouse or by code, use the `TecNativeLineSelect` class (unit `uecEditNativeLine`)

```
var SelectLine : TecNativeLineSelect;
...
SelectLine := TecNativeLineSelect.create;
// triggered by selection
SelectLine.OnSelect := doSelect;
// triggered by deselection
SelectLine.OnDeselect := doDeselect;

SelectLine.Line := your_Line;
```



Hover over the line and click on the selection point to validate it, click on it again to remove it, the OnSelect event is triggered as soon as the validation of the second point

You can then create a line or a polygon with SelectionTo, or go directly to the selected with points Selection[x]

```
// triggered by selection
procedure form.doSelect(Sender: TObject);
begin

    // convert selected points to a new line
    if not assigned(NewLine) then
        newLine := map.addLine(0,0);

        SelectLine.SelectionTo(newLine);

    // convert selected points to a new polygone
    if not assigned(NewPoly) then
        newPoly := map.addPolygone(0,0);

        SelectLine.SelectionTo(newPoly);

end;

end;
```

Use SelectionFrom(ALat,ALng,BLat,BLng) for the code segment selection.

```
// select portion by code
SelectLine.SelectionFrom(ALat,ALng,BLat,BLng);

// return number of points in selection
SelectLine.Count;

// return point (TECPointLine) number x in selection
P := SelectLine.Selection[x];

// deselect portion by code
SelectLine.Deselect;
```

Geodesic line

A geodesic is the shortest path between two point, use the **AddGeodesicLine** function to create a.

```
function TNativeMapControl.AddGeodesicLine(const SLat,SLng,ELat,ELng:double;
const GroupName: string = "";
const maxSegmentLength:integer=5000): TECShapeLine;
```

You need at least to indicate the coordinates of two points.

maxSegmentLength set the maximum size, in meters, intermediate segments.

```
var line:TECshapeLine;
PtA,PtB : TECShapeMarker;
...

PtA := map.shapes.Markers[0];
PtB := map.shapes.Markers[1];
line
:= map.AddGeodesicLine(PtA.Latitude,ptA.Longitude,ptB.Latitude,ptB.Longitude)
```



Fig. 121 geodesic line

Save moving

The purpose of this example is to save its movement in a

TECShapeLine, a triangle pointing in the direction of the displacement will indicate the current position.

```

var MyLocationShape : TECShapePOI;
    MyLocationLine   : TECShapeLine;

procedure TForm1.FormCreate;
begin
    // create triangle
    MyLocationShape := map.AddPoi(0, 0);
    MyLocationShape.POIShape := poiTriangle;
    MyLocationShape.Width    := 18;
    MyLocationShape.height   := 24;
    MyLocationShape.YAnchor  := 0 ;
    MyLocationShape.visible  := false;
end;

// Connect this to your TLocationSensor.OnLocationChanged
// save your location in a TECShapeLine
procedure
TForm1.LocationSensorLocationChanged(Sender: TObject;
    const OldLocation, NewLocation: TLocationCoord2D);

begin

    map.beginupdate;

    // create new line if not assigned
    if not assigned(MyLocationLine) then
    begin
        MyLocationLine :=
        map.AddLine(NewLocation.Latitude, NewLocation.Longitude,
            'mylocations');
        MyLocationLine.weight := 4;
        MyLocationLine.Color := GetRandomColor;

        // add border
        MyLocationLine.BorderSize := 2;
        MyLocationLine.BorderColor
        := GetShadowColorBy(MyLocationLine.Color, 32);

    end

    else

        MyLocationLine.add(NewLocation.Latitude, NewLocation.Longitude);

        MyLocationShape.SetDirection(NewLocation.Latitude, NewLocation.Longitude);

```



```
MyLocationShape.visible := true;  
  
    // center on your position  
  
    map.setCenter(NewLocation.Latitude, NewLocation.Longitude);  
  
    map.endupdate;  
  
end;
```

Weather

By using the services of OpenWeatherMap.org you can view the weather along your TECShapeLine.

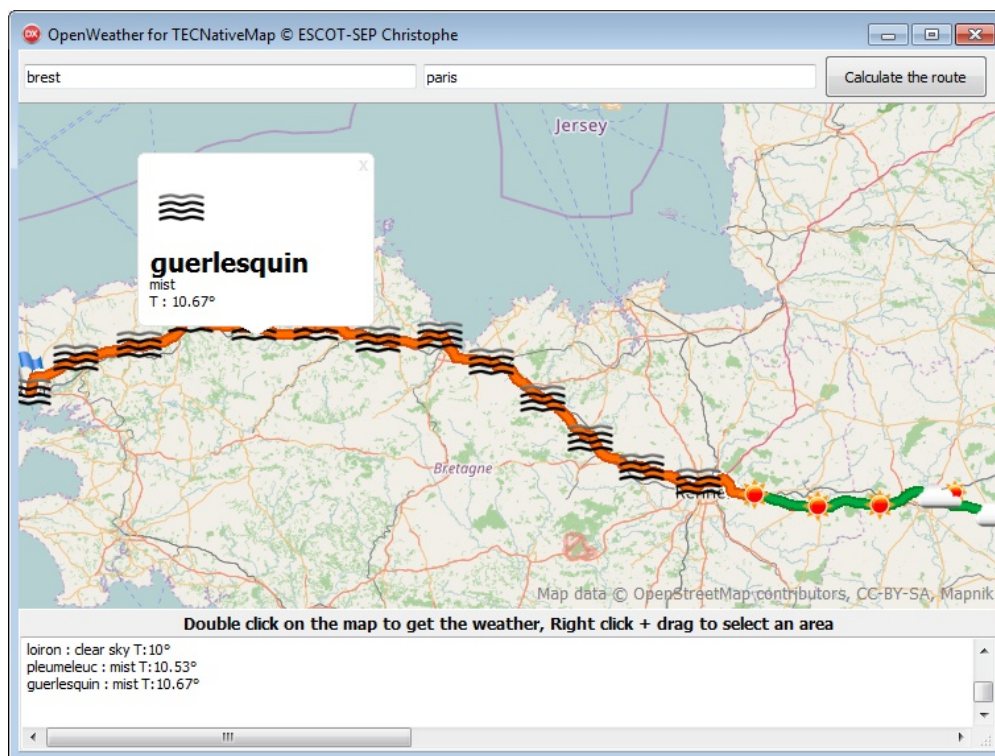


Fig. 122 Weather on the road

You first need to get a [key](http://OpenWeatherMap.org) to use the api of OpenWeatherMap


```
map.OpenWeather.Key := your_key;

// Get the weather along a TECShapeLine
Line.ShowWeather := true;
```

See the RouteWeather demo for more information on how to use the api of OpenWeatherMap.

TECShapePolygone descends from **TECShapeLine** and allows to display on your map polygons.

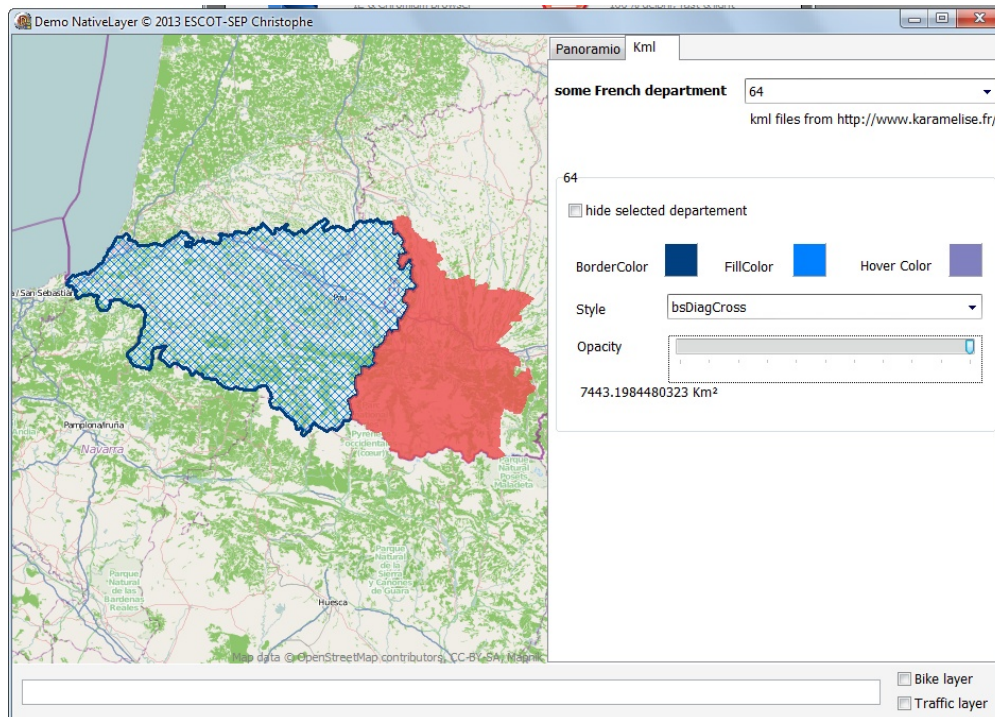


Fig. 123 Polygones

In addition to the properties of **TECShapeLine** you have access to

Property **Area** : double

Surface Km²

property **FillColor** : TColor

Fill color

Property **Color** : TColor

The border color

property **Style** : TBrushStyle

Fill style (bsSolid, bsClear, bsHorizontal, bsVertical, bsFDDiagonal, bsBDDiagonal, bsCross, bsDiagCross)

*Style does not work under Firemonkey, use **BrushFilename** to indicate an image that will fill your polygon, BrushFilename can be either a local path or a url or an encoded as base64 for image, like `Marker.Filename`*

property **Opacity** : byte

Opacity of the polygon (0 = fully transparent, 100 = opaque)

property **Level** : integer

Height in metres of the polygon for a display in pseudo 3D



Fig. 124 Level = 16 - Level = 0

OnAfterDraw

You also have an OnAfterDraw event that allows you to draw on top your polygon, example to include its description.

```

FPoly : TECShapePolygone;

FPoly := TECShapePolygone(map.add(nsPolygon,Lat,Lng)) ;
FPoly.OnAfterDraw := doAfterDraw;
FPoly.Description := 'Polygone '
+inttostr(map.Shapes.Polygones.Count);

procedure TFormNativeLinePolygone.doAfterDraw(const
  canvas:TECCanvas;var Rect:TRect;item:TECshape) ;
var x,y,w,h:integer;
begin
  //transparancy text
  canvas.brush.Style := bsClear;

  canvas.font.color := clBlack;
  canvas.font.Style := [fsBold];

  w := canvas.TextWidth(item.Description) ;
  h := canvas.TextHeight(item.Description);

  x := rect.Left+((rect.Right-rect.Left-w) div 2);
  y := rect.top+((rect.bottom-rect.top-h) div 2);

  canvas.TextOut(x,y,item.Description);

end;

```

TECShapeInfoWindow allows you to display a panel containing text.

Windows are managed by an **TECShapeList** list accessible through the property **InfoWindows** groups [TECShapes](#)



Fig. 125 InfoWindow

TECShapeInfoWindow

The following properties are available

property **Color** : TColor

Color window

property **Style** : TECInfoWindowStyle

The window style : **iwsTransparent**, **iwsRectangle** or **iwsRoundRect** (by default)

property **Visible** : boolean

show / hide window

property **CloseButton** : boolean

show / hide close button

property **ContentCenter** : boolean

Center text horizontally

property **Width** : integer;

Width in pixels

property **Height** : integer;

Maximum height, if the text takes up less space then the height is automatically adjusted

property **Content** : string

Text to display

You can use a subset of HTML to enrich the display

Tags ****, **<h>**, **<a>**, ****, **<i>**, **<u>**, **<s>**, ****, **
, **<tab> and **<PlainText>** are supported

```
map.Shapes.InfoWindows.add(map.latitude,map.longitude,
'content');
// html content
map.Shapes.InfoWindows[0].Content :=
```

```
'<h2>Titre</h2>' +
'<tab="32"><b>Bold</b><br>' +
'<tab="32"><font face="Times New Roman" size=14
bkcolor=FF0000 color=FFFFFF>Font</font><br>' +
'<tab="32"><a href="#16/43.094089/-0.046520">Link
  Lourdes</a> 
```

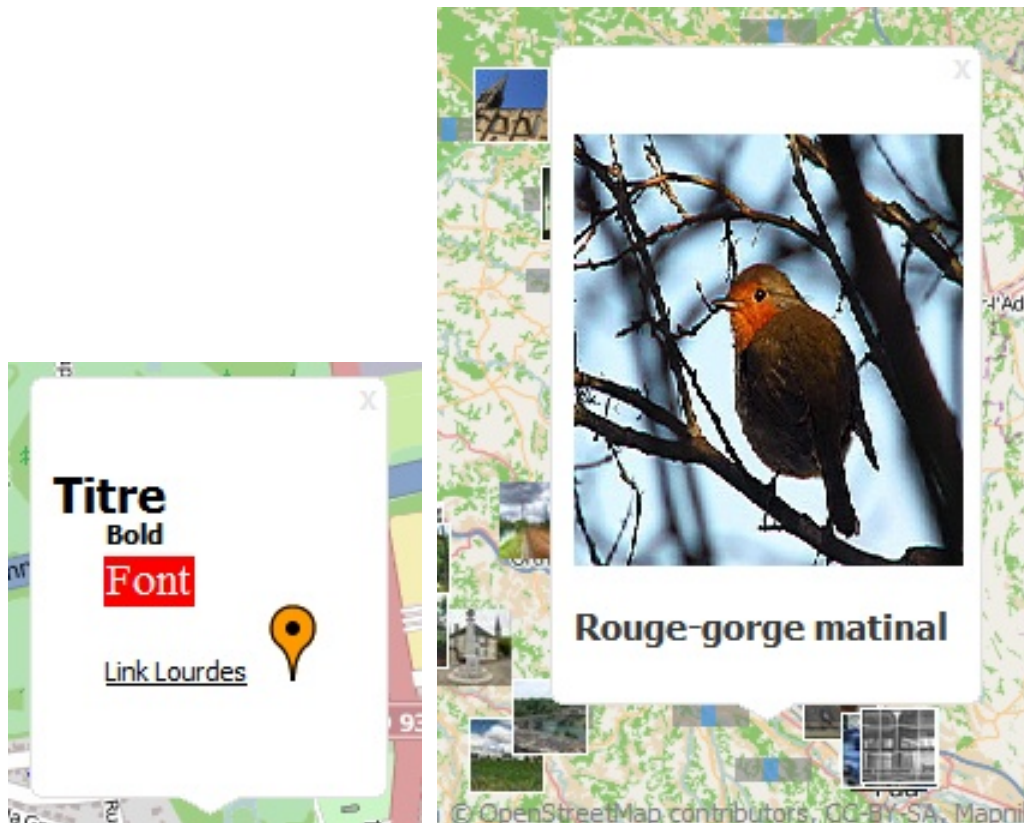


Fig. 126 html Content InfoWindow

property **OnOpen** : TOnInfoWindowOpen;

This event is raised just before the window display, you can change the contents or cancel opening

```
win.OnOpen := doOnOpenWindow;
...
procedure TForm.doOnOpenWindow(const
  infoWindow: TECShapeInfoWindow; var cancel: boolean);
```



```

begin
    // set cancel to true for not open then infowindow
    (default false)
    // cancel := true;

    infoWindow.Content := 'change content here';

end;

```

When a window is closed by clicking on his cross the event **OnCloseInfoWindow** of **TECNativeMap** is raised and a **TECShapeInfoWindow** is passed as a parameter.

OnCloseInfoWindow *does not occur if you close by using the property* **visible**

Automatic closing

Using an **animation** of type **TECAAnimationAutoHide** you can close the window automatically after a few seconds.

```

var win: TECShapeInfowindow;
...
win := map. addInfoWindow (lat, lng);

// automatically close the window after 15 seconds
win.Animation := TECAAnimationAutoHide.Create;
TECAAnimationAutoHide (win.Animation).MaxTiming := 1000 *
15;

```

Link

You can set a link by using the `<a>` tag, to intercept the click Connect

on the **OnBeforeUrl** event of your map.

```
s := '<a href="#your_data"> link </a>';
OpenInfoWindow (s, Item.Latitude, Item.Longitude);
```

...

Procedure TForm.mapBeforeUrl (Sender: TObject; **var** Url: **string**);
Begin

```
// here url = '#your_data'
```

End;

You have the option to set a special link that allows you to move around the map, it is of the form **#zoom/latitude/longitude**

```
'<a href="#16/43.094089/-0.046520"> goto here </a>'
```

Automatic opening by clicking on an element

By switching the property **UseInfoWindowDescription** of your map to true, the field **Description** of the elements will be displayed as an infoWindow when you click on them.

You can access a global infoWindow through the **InfoWindowDescription** property.

```
map.UseInfoWindowDescription := true;
map.InfoWindowDescription.minHeight := 300;
...
marker1.Description := '<h1>Marker 1</h1>';
line1.Description := '<h1>Line 1</h1>';
```

TECShape has a property type Animation TECShapeAnimation to animate.

The following classes are available as standard,
TECAnimationShapeColor , **TECAnimationAutoHide**,
TECAnimationMarkerFilename, **TECAnimationFadePoi**,
TECAnimationDrawPath, **TECAnimationMoveToDirection** and
TECAnimationMarkerZoomFilename

*You can create the your in the class heritant
 TECShapeAnimation and redefining the procedure
 Animation*

TECAnimationShapeColor

This animation will toggle the property **Color** and **HoverColor** the TECShape

```
shape := map.shapes.markers[0];
shape.Animation := TECAnimationShapeColor.create;
// change color every 250ms
shape.Animation.Timing := 250;

shape := map.shapes.markers[1];
// stop animation after 10 cycles
shape.Animation := TECAnimationShapeColor.create(10);
// change color every 250ms
shape.Animation.Timing := 250;
```



TECAAnimationMarkerFilename

This class is used to modify the property **Filename** of [TECShapeMarker](#) and therefore have an animation of images

```
var Animation : TECAAnimationMarkerFilename ;
    Shape      : TECShapeMarker;

shape := map.shapes.markers[0];

Animation := TECAAnimationMarkerFilename.create;

    // importantly other animation will not work
Shape.Filename :=
'http://maps.google.com/mapfiles/ms/icons/orange-dot.png';

    // change image every 200ms
Animation.Timing := 200;
Animation.Filenames.add(
'http://maps.google.com/mapfiles/ms/icons/orange-dot.png'
);
Animation.Filenames.add(
'http://maps.google.com/mapfiles/ms/icons/green-dot.png' );
Animation.Filenames.add(
'http://maps.google.com/mapfiles/ms/icons/blue-dot.png' );

shape.Animation := Animation;
```

TECAAnimationMarkerZoomFilename

This class is used to change the **Filename** property of [TECShapeMarker](#) depending on the zoom level of the map

```
...
shape := map.shapes.markers[0];

    // create animation and fix filename by default
AnimZoom := TECAAnimationMarkerZoomFilename.create(
'http://maps.google.com/mapfiles/ms/icons/red-dot.png' );

    // set animation
shape.Animation := AnimZoom
```

```

// filename for zoom 14
AnimZoom.add(14,
'http://maps.google.com/mapfiles/ms/icons/orange-dot.png'
);

// filename for zoom 18
AnimZoom.add(18,
'http://maps.google.com/mapfiles/ms/icons/green-dot.png' );

// filename for zoom 10
AnimZoom.add(10,
'http://maps.google.com/mapfiles/ms/icons/blue-dot.png' );

```

TECAnimationFadePoi

The size (width et Height) of a [TECShapePOI](#) varies from a minimum to a maximum and at the same time the marker becomes increasingly transparent to disappear.

Transparency for poiEllipse is supported only in version Firemonkey



Fig. 128 TECAnimationFadePoi

property **MaxSize** : integer

Maximum size of the marker (default 20)

property **StartSize** : integer

Minimum size of the marker (default 10)

property **StartOpacity**: byte

Start opacity (default 50), it decreases to 0 (full transparency)

property **Step** : integer

Number of step from Startsize to MaxSize (default 10)

```
anim := TECAAnimationFadePoi.Create;  
anim.MaxSize := 32;  
anim.StartSize := 8 ;  
  
anim.StartOpacity := 90;  
  
ShapePOI.Animation := anim;
```

TECAAnimationDrawPath

Displays a [Polyline](#) step by step

Constructor **Create**(const ValueDuration: cardinal = 5000);

Creation of animation, ValueDuration is the time in milliseconds to display all of the trace

property **Latitude** : Double

Latitude of the last displayed point

property **Longitude**: Double

Longitude of the last displayed point

property **OnDraw** : TNotifyEvent

Event is raised after each display

The OnEndAnimation event is raised when the trace was fully displayed

```
// draw route in 3s
DrawingRoute := TECAAnimationDrawPath.Create(3000);

// call when line is 100% draw
DrawingRoute.OnEndAnimation := doOnDrawAllRoute;

// call every draw
DrawingRoute.OnDraw := doOnDraw

// DrawingRoute will be automatically deleted
map.shapes.Lines[i].Animation := DrawingRoute;
```



Fig. 129 TECAAnimationDrawPath

TECAAnimationMoveOnPath

This class allows to move a **TECShape** along a **TECShapeLine**

property **Distance** : integer ;

Distance in metres from the beginning of TECShapeLine

property **Heading** : boolean;

Adjust the angle of the element according to the movement

property **Speed** : integer;

Movement speed in Km/H

property **Stop** : boolean;

Allow or not moving

property **ComeBack** : boolean;

Indicates the direction of travel, if **ComeBack = true** then the element moves from the end to the beginning of TECShapeLine

property **OnDriveUp** : TNotifyEvent

Event raised when the item arrives at the end or at the beginning of the path

```

shape := map.shapes.markers[0];

    // create animation on Lines[0] , start à 0 meter, Speed
    = 50km/h
moving := TECAAnimationMoveOnPath.create(map.shapes.Lines[0]
,0,50);

shape.animation := moving;

moving.OnDriveUp := doEndAnimationMove;
    // run shape
moving.stop := false;
...
procedure TForm.doEndAnimationMove(sender : TObject);
begin
    // We arrived at the end of the road, reversing the
    direction of movement

```

```

        TECAAnimationMoveOnPath(TECShape(sender).animation).ComeBack
:= not
    TECAAnimationMoveOnPath(TECShape(sender).animation).ComeBack;
    // run shape
    TECAAnimationMoveOnPath(TECShape(sender).animation).Stop
:= false;
end;

```



Fig. 130 TECAAnimationMovOnPath

You do not need to destroy TECShapeAnimation the TECShape does this automatically when you assign a new animation or when he is even released

```

shape := map.shapes.markers[0];
// stop animation after 10 cycles
shape.Animation := TECAAnimationShapeColor.create(10);
// change color every 250ms
shape.Animation.Timing := 250;

...

// free TECAAnimationShapeColor
shape.Animation := nil;

```

The animations have events **OnAnimation**, raised after each cycle and **OnEndAnimation** raised at the end of the animation if you indicated a

maximum number of cycles during the creation of animation.

```

shape := map.shapes.markers[0];
// stop animation after 10 cycles
shape.Animation := TECAAnimationShapeColor.create(10);
// change color every 250ms
shape.Animation.Timing := 250;
//
shape.Animation.OnEndAnimation := doEndAnimation;
...

procedure TForm.doEndAnimation(sender : TObject);
begin
    ShowMessage('End Animation: '
+inttostr(TECShape(sender).Id));
end;

```

TECAAnimationMoveToDirection

This animation allows you to move an element in one direction depending on its speed

property **Direction**: double

de 0 à 360°, 0 indicating the top of the map (North)

property **Heading** : boolean

Rotates the item depending on its direction

property **Speed**: integer

Speed in km/h

procedure **Start**

Starts the animation

property **Stop**: boolean

Stops the animation

property **TimeOut** : cardinal

indicates a time in milliseconds after which the animation is stopped, 0 = unlimited, 1000 = 1s

```

var Direction : integer;
    animD      : TECAnimationMoveToDirection;
    ShapeMarker: TECShapeMarker;
    SpeedKMh   : integer;
begin
    // Optimization, call BeginUpdate before adding elements
    map.BeginUpdate;

    // Create 360 markers that will move in their own direction
    for Direction := 0 to 359 do
    begin
        ShapeMarker :=
            map.AddMarker(map.Latitude,map.longitude) ;
        shapeMarker.Filename :=
            'http://www.helpandweb.com/arrow2.png';
        SpeedKMh := 30 + random(100);

        // create animation
        animD
        := TECAnimationMoveToDirection.Create(SpeedKMh,Direction);
        shapeMarker.Animation := animD;

        // the item points in the direction
        animD.Heading := true;

        // start move
        animD.Start;

    end;

    // We can now allow the map to be updated

    map.EndUpdate;

```



TECAAnimationAutoHide

This animation hides an element by switching its visible property to false, MaxTiming shows the time in milliseconds after which the element is hidden.

```
var win: TECShapeInfowindow;
...
win := map. addInfoWindow (lat, lng);

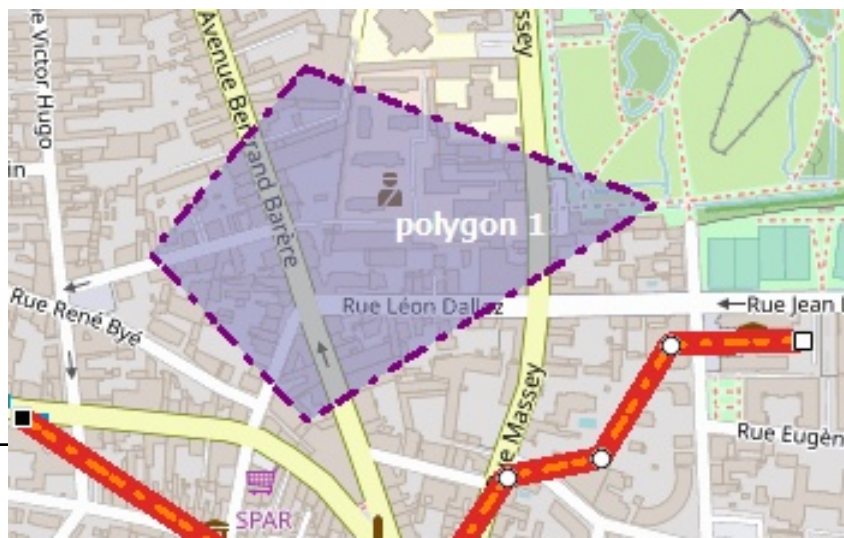
// automatically close the window after 15 seconds
win.Animation := TECAAnimationAutoHide.Create;
TECAAnimationAutoHide (win.Animation).MaxTiming := 1000 *
15;
```

TECAAnimationDash

Allows to scroll the pattern used by PenStyle in TECShapeLine and TECShapePolygone.

```
poly.PenStyle := psuserStyle;
poly.setcustomdash([4,8,2,8,8,4]);
poly.animation := TECAAnimationDash.create;

line.PenStyle := psuserStyle;
line.setcustomdash([4,8,2,8,8,4]);
line.animation := TECAAnimationDash.create;
// reverse the direction
TECAAnimationDash(line.animation).MoveForward := false;
```



API Keys

TECNativeMap uses OpenStreetMap.

If you want to use these services of MapQuest [you must obtain a key from MapQuest](#).

Do the same to use the services of [MapZen](#) and [MapBox](#).

Offline mode

If you specify a directory in the [LocalCache](#) property, all datas will be cached and can be reused in offline mode.

Available in future version 2.7

TECRouting

Managing Routes and turn by turn navigation.

```
procedure Request(const StartAdress, EndAdress: string;const params:  
string = ""); overload;
```

```
procedure Request(const dLatLngs: array of double;const params:  
string = ""); overload;
```

Calculate a route between addresses or points

```
map.Routing.routeType := rtCar;
// get data road between Tarbes and Paris
map.Routing.Request('Tarbes','Paris');
// get data road between Tarbes and Lourdes via Aureilhan
map.Routing.Request('Tarbes','Aureilhan|Lourdes');
// you can also pass a array of coordinate pairs
map.Routing.Request([43.232858,0.0781021,43.2426749,
0.0965226]);
```

Params allows you to pass additional parameters to the routing engine

property **routeType**: TMQRouteType

Type of route (rtCar, rtFastest, rtShortest , rtPedestrian, rtBicycle)

property **GroupName** : string

Name of the group which will own the new road ([TECShapeLine](#)), default empty

property **Color** : TColor

Color of the road

property **Weight**: integer

Thickness of the road

property **StartMarkerFilename**: string

Image used to mark the start, default



property **EndMarkerFilename**: string

Image used to mark the arrival, by default



property **RouteDrawingTime**: integer

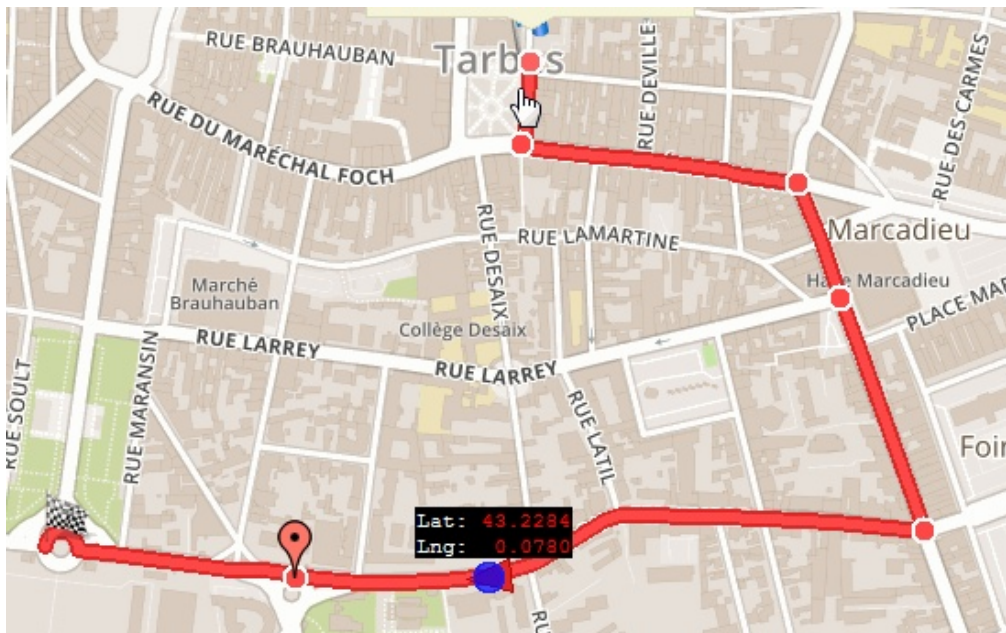
Time (in MS) to display the route, allows to realize an animation for a progressive display.
Default 0, the road appears only once.

property **EditRoute**: TECShapeLine

To edit the route with the mouse

property **EditOnClick** : boolean

By default a click on a route makes switch to edit mode, simply click on the points A or B to exit this mode



Routing engine

procedure **Engine**(value:TRoutingEngine;const ApiKey:string="");

Allows you to choose a routing engine, OpenStreetMap (default choice), [MapQuest](#) (only one engine available before), [MapZen](#), [OSRM](#) or [MapBox](#)

```
map.routing.Engine(reOpenStreetMap);
map.routing.Engine(reMapquest, 'your_key');
map.routing.Engine(reMapZen, 'valhalla-your_key');
map.routing.Engine(reOSRM);
map.routing.Engine(reMapBox, 'your_token');
```

Use **EngineValidUrl** to change the url before is sent to the engine

```

procedure doUrl(sender:TECThreadDataRoute);
begin
    // Here you can change the url
    // ! your are not in main thread
    sender.Url := ...;
end;

procedure TFDemoNativeRoute.FormCreate(Sender: TObject);
begin
    // setup routing
    map.routing.Engine(reOSRM);
    // intercept url before engine
    map.Routing.EngineValidUrl := doUrl;

end;

```

Define its own engine

Use EngineUrl and EngineExcute to edit your own engine.

property **EngineUrl** : TECThreadRoutingUrl

property **EngineExecute** : TECThreadRoutingExecute

```

procedure TForm.FormCreate(Sender: TObject);
begin
    map.Routing.Engine(reCustom);
    map.Routing.EngineName := yourRouting_Name;
    map.Routing.EngineUrl   := YourRoutingUrl;
    map.Routing.EngineExecute := YourRoutingExecute;
end;

    // build url here
procedure YourRoutingUrl(sender:TECThreadDataRoute);
begin

    // sender:TECThreadDataRoute content all necessary info

    sender.url := ...
end;

    // build route here
procedure YourRoutingExecute(sender:TECThreadDataRoute; var

```



```
valid:boolean);  
begin  
    // sender.RequestResult content data  
end;  
    // see source of engine supported in uecNativeMapShape.pas
```

EngineUrl and EngineExecute are executed in another thread !

property **EngineName** : string

Gives a name to the engine, used for local storage

property **EngineKey** : string

Key to use the engine, you get yours from MapQuest, MapZen or MapBox

property **TurnByTurn**: TECTurnByTurn

The turn by turn navigation management

4 events are available

property **OnAddRoute** : TOnAddRoute

Raised when the road has been created

property **OnChangeRoute** : TOnChangeRoute

Raised when the road has changed

property **OnErrorRoute** : TOnErrorRoute

Raised if the road has not been created

property **OnDrawRoute** : TOnDrawRoute

Raised when the road was fully displayed (useful if RouteDrawingTime > 0)

```
map.Routing.OnAddRoute      := doOnAddRoute;
map.routing.OnErrorRoute    := doOnErrorRoute ;
map.Routing.OnChangeRoute:= doOnChangeRoute;

procedure
  TFDemoNativeRoute.doOnAddRoute(sender: TECShapeLine;const
  params:string);
begin
  // sender is the route (TECShapeLine)
  // params this is the URL sent to the routing engine
end;

procedure
  TFDemoNativeRoute.doOnErrorRoute(sender: TObject;const
  params:string);
begin
  // sender is a TECThreadDataRoute
  ShowMessage('Route not found !');
end;

procedure
  TFDemoNativeRoute.doOnChangeRoute(sender: TECShapeLine;
const params:string);
begin
  if not assigned(sender) then exit;

  showMessage ( doubleToStrDigit(sender.Distance, 2) + 'km
- ' +
                                doubleToStrDigit(sender.Duration / 3600, 2
)+ 'h' )  ;
end;
```

TECTurnByTurn

property **line**: TECShapeLine

The way ahead

function **Position**(const Lat, Lng: double):boolean

Specify your GPS position

property **AlertDistance**: integer

By default 300 meters

property **ExecutionDistance**: integer

Default 100 meters

property **ErrorDistance**: integer

Default 30 meters

property **NextInstruction**:string

Next statement to follow

 *Available in the OnInstruction event*

property **NextManeuver**: string ;

List of operations (in the form of a json string containing various parameters depending on the engine used)

 *Available in the OnInstruction event*

property **NextInstructionInKm**: double ;

Distance in KM for the next instruction

 *Available in the OnInstruction event*

property **NextInstructionPosition**: TLatLng ;

Geographical position of the next instruction

Available in the OnInstruction event

available events

property **OnAlert**: TOnTurnByTurnAlert

Triggered when you arrive less than **AlertDistance** meters from the target

property **OnInstruction**: TOnTurnByTurnInstruction

Triggered when you arrive less than **ExecutionDistance** meters from the target

property **OnArrival**: TNotifyEvent

property **OnError**: TOnTurnByTurnError

property **OnAfterInstruction**: TOnTurnByTurnInstruction

property **OnConnectRoute**: TNotifyEvent

Triggered by a road connection

property **OnDisconnectRoute**: TNotifyEvent

Triggered by the disconnection of a road

```
map.Routing.TurnByTurn.OnInstruction
:= doOnTurnByTurnInstruction;
```

```

map.Routing.TurnByTurn.OnAfterInstruction
:= doOnTurnByTurnInstruction;
map.Routing.TurnByTurn.OnAlert
:= doOnTurnByTurnAlert;
map.Routing.TurnByTurn.OnArrival
:= doOnTurnByTurnArrival;
map.Routing.TurnByTurn.OnError
:= doOnTurnByTurnError;

procedure TFDemoNativeRoute.doOnTurnByTurnAlert(sender
: TECTurnByTurn;const Instruction:string;const
Distance:double);
begin
    // alert is fired before OnInstruction
end;

procedure TFDemoNativeRoute.doOnTurnByTurnInstruction(sender
: TECTurnByTurn;const Instruction:string;const
Distance:double);
begin
    // execute instruction in Distance (km)
end;

```

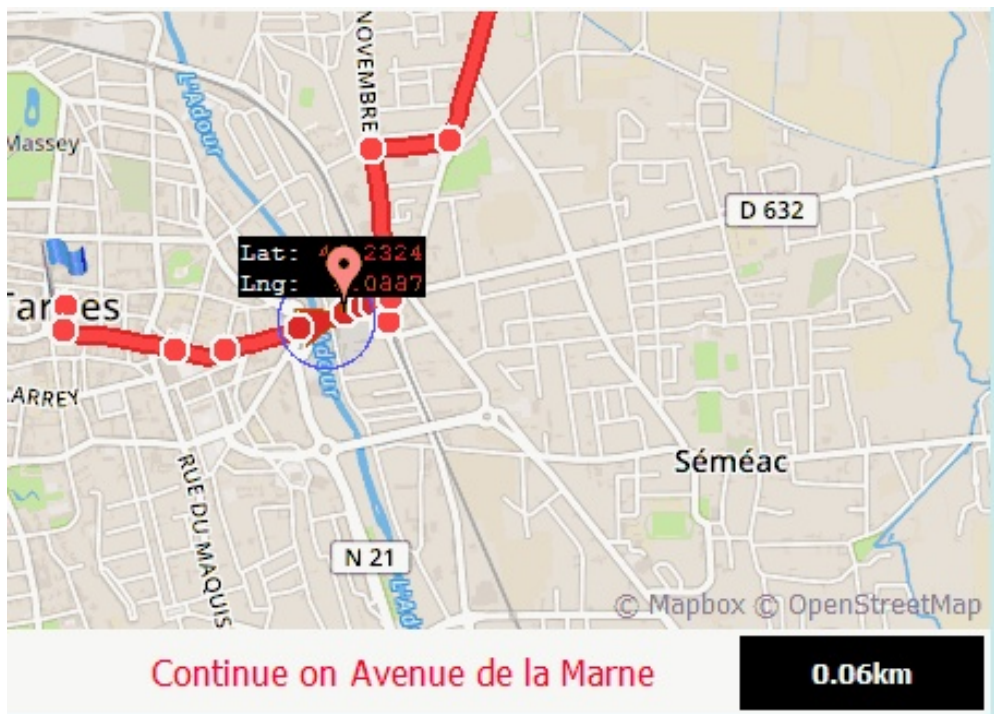


Fig. 133 DemoNativeRoute

The DemoNativeRoute demo has been rewritten to use the new feature.

Old methods, use new features available since version 2.7

Search for routes

You have at your disposal 4 routines (2 synchronous et 2 asynchronous) to calculate a route.

```
function GetRoutePathByAddress(const StartAdress, EndAdress:
string;const routeType:TMQRouteType = rtFastest const params: string
= " ): TECRoutePath
```

```
function GetRoutePathFrom(const dLatLngs: array of double;const
routeType:TMQRouteType = rtFastest ;const params: string = "
): TECRoutePath
```

```
procedure GetASyncRoutePathByAddress(const StartAdress,
EndAdress: string;const routeType:TMQRouteType = rtFastest ;const
params: string = " )
```

```
procedure GetASyncRoutePathFrom(const dLatLngs: array of
double;const routeType:TMQRouteType = rtFastest ;const params:
string = " )
```

These functions allow to obtain information from a route without trace, asynchronous

procedures will run in background and raise the **OnRoutePath** event when the data are available.

See open.mapquestapi.com/directions/ for the parameter Params

*If you do not release the **TECMapRoutePath** obtained, it will be automatically during the destruction of **TECNativeMap***

Display the route

You can create a **TECShapeLine** from one **TECMapRoutePath** with the function **AddRoute**

```
// Delphi map component EMap  
  
var line : TECShapeLine;  
...  
    // ge data road between Tarbes and Lourdes via Aureilhan  
routePath := map.GetRoutePathByAddress('Tarbes',  
    'Aureilhan|Lourdes');  
    // draw polyline from routePath  
if routePath<>nil then  
begin  
  
    Line := map.AddRoute(routePath);  
    // see the entire route  
map.fitBounds(line.NorthEastLatitude,line.NorthEastLongitude,line.SouthWestLa  
  
routePath.free;  
end;
```

You can also directly edit a `TECShapeLine` with the procedure

setRoutePath

```
// Delphi map component EMap

var line : TECShapeLine;
...
// ge data road between Tarbes and Lourdes via Aureilhan
routePath := map.GetRoutePathByAddress('Tarbes',
'Aureilhan|Lourdes');
// draw polyline from routePath
if routePath<>nil then
begin

    // change line 0 with new data
    if map.shapes.lines.count>0 then
        map.shapes.lines[0].setRoutePath(routePath);
    // see the entire route
    map.fitBounds(line.NorthEastLatitude,line.NorthEastLongitude,line.SouthWestLa

routePath.free;
end;
```

Edit the road with the mouse

For this you will need to add the unit **uecEditNativeLine** (FMX.uecEditNativeLine under FireMonkey)

You can then use the **TecNativeLineToRoute** class to be able to dynamically change the route.

property **Line** : TECShapeLine

TECShapeLine to modify

property **RouteType** : TMQRouteType

type of road (rtFastest, rtShortest, rtPedestrian, rtBicycle)

property **Modified** : boolean

Indicates if the road was amended

property **OnClick** : TOnShapeMouseEvent

To respond to the click on the road

property **OnMouseOver** : TOnShapeMouseEvent

Can respond to the overview of the road

property **OnChange** : TNotifyEvent

Triggered event has each modification of the road

Example of use

```
// Delphi map component EMap

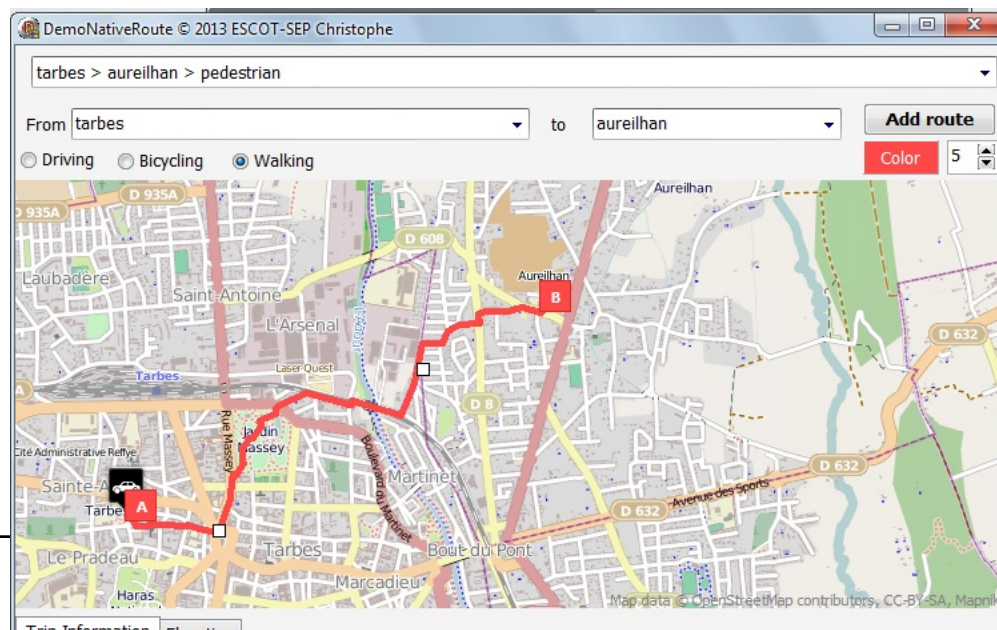
var ecNativeLineToRoute : TecNativeLineToRoute;
...
ecNativeLineToRoute := TecNativeLineToRoute.create;

// edit route
ecNativeLineToRoute.Line := map.shapes.Lines[0];

// end edit route
ecNativeLineToRoute.Line := nil;
...
ecNativeLineToRoute.free;
```

DemoNativeRoute

A demonstration is available to understand the management of roads.



TECNativeLayer is the base class of the layers that allow you to react to the displayed area, and mouse actions.

```
TECNativeLayer = class
  private
    FObserver : TNativeMapObserver;
    FShapes   : TECShapes;
    FMap      : TNativeMapControl;

    FOnShapeRightClick,
    FOnShapeClick : TOnShapeMouseEvent;

    FOnMouseClicked,
    FOnMouseMove : TNotifyEvent;

    function getMinZoom : byte;
    function getMaxZoom : byte;

    procedure setMinZoom(value:byte);
    procedure setMaxZoom(value:byte);

    function getVisible : boolean;

  protected

    procedure doOnMapEndMove(sender : TObject);
  virtual;
    procedure doOnShapeClick(sender : TObject);
  virtual;
    procedure doOnShapeRightClick(sender : TObject);
  virtual;
    procedure doOnMapMouseMove(sender : TObject);
  virtual;
    procedure doOnMapMouseClicked(sender : TObject);
  virtual;
    procedure doOnMapHiResChange(sender : TObject);
  virtual;

  public

    procedure setVisible(const value:boolean); virtual;

    constructor Create(_FMap:TnativeMapControl;const Name:
string); virtual;
    destructor Destroy; override;

    property OnShapeClick      : TOnShapeMouseEvent read FOnShapeClick
```

```

        write FOnShapeClick;
        property OnShapeRightClick : TOnShapeMouseEvent read
FOnShaperightClick write FOnShapeRightClick;

        property OnMouseMove : TNotifyEvent read
FOnMouseMove write FOnMouseMove;
        property OnMouseClicked : TNotifyEvent read
FOnMouseClicked write FOnMouseClicked;

        property Map      : TNativeMapControl   read FMap;
        property Shapes   : TECShapes           read FShapes;
        property Visible   : boolean             read
getVisible write setVisible;

        property MaxZoom   : byte                read
getMaxZoom write setMaxZoom;
        property MinZoom   : byte                read
getMinZoom write setMinZoom;

end;

```

*You can either plug into **OnMapHiResChange** or overload the **doOnMapHiResChange(sender: TObject)** procedure so that your layer adapts to*

Panoramio

Google closed Panoramio November 4, 2016, the service is no longer functional

A **Panoramio** layer is integrated into **TECNativeMap** through the property **PanoramioLayer**

```
// show panoramio layer
```

```
map.PanoramioLayer := true;
```

The event **OnPanoramioClick**(sender: TObject; const Item: TECShape; const ownerId, ownerName, PhotoId, PhotoDate, PhotoTitle, PhotoUrl, copyright: string) is raised when clicking a Panoramio item.



Fig. 135 Click sur un élément Panoramio

TECNativePlaceLayer

Allows you to automatically display the result of [a search](#)

To use this layer you must embed unit

uecNativePlaceLayer *or* **FMX.uecNativePlaceLayer** *depending on whether you use the VCL version or FireMonkey*

Example: a layer that displays restaurants

```
// search Layer

FPlacesLayer
:= TECNativePlaceLayer.create(map, 'PLACES_LAYER');
FPlacesLayer.OnPlaceClick := doOnPlaceClick;

FPlacesLayer.Visible := true;
FPlacesLayer.Search := 'node[amenity=restaurant]';
// standard image 32x32
FPlacesLayer.MarkerFilename :=
'http://www.helpandweb.com/cake_32.png';
// lat,lng on center of image
FPlacesLayer.XAnchor := 16;
FPlacesLayer.YAnchor := 16;
// hi-res image 64*64
FPlacesLayer.MarkerHiResFilename :=
'http://www.helpandweb.com/cake_64.png';
// lat,lng on center of image
FPlacesLayer.HiResXAnchor := 32;
FPlacesLayer.HiResYAnchor := 32;
...

// event click on place shape
procedure TForm1.doOnPlaceClick(sender : TECShape);
var
    pResult : TECPlaceResult;
    r,
    n,
    Content : string;
    i       : integer;
begin
```



```
// here Sender and Sender.Item are allway assigned, and
Sender.Item is TECPlaceResult
```

```
pResult :=TECPlaceResult(Sender.item)
```

```
for i:=0 to pResult.CountResult-1 do
begin
```

```
  n := pResult.NameResult[i];
```

```
  r := pResult.Result[n];
```

```
  if length(n)< 9 then
```

```
    n := n+'<tab="65">';
```

```
  content := content+'<b>'+n+'</b>: '+r+'<br>';
```

```
end;
```

```
if FPlacesLayer.Shapes.InfoWindows.Count=0 then
```

```
begin
```

```
  FPlacesLayer.Shapes.InfoWindows.add(0,0,'');
```

```
  FPlacesLayer.Shapes.InfoWindows[0].zindex := 100;
```

```
end;
```

```
FPlacesLayer.Shapes.InfoWindows[0].content := Content;
```

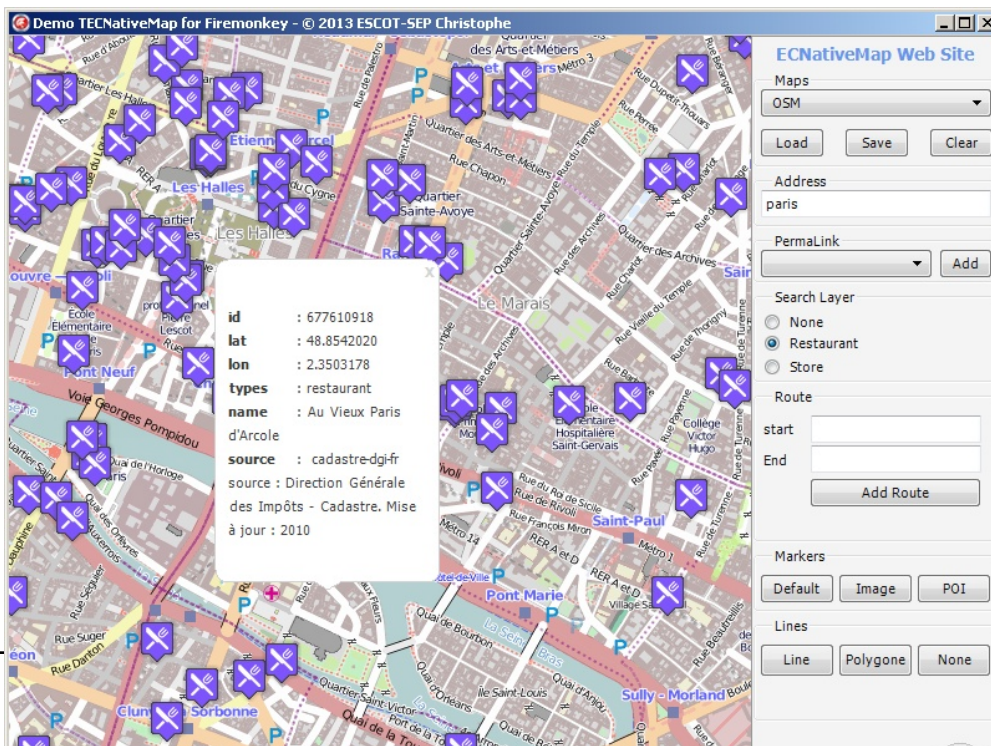
```
FPlacesLayer.Shapes.InfoWindows[0
```

```
].SetPosition(Shape.Latitude,shape.Longitude);
```

```
FPlacesLayer.Shapes.InfoWindows[0].Visible := true;
```

```
end;
```

```
end;
```



By default the elements of layer are [TECShapeMarker](#) but you can change the type of TECShape by redefining

TECNativePlaceLayer.doCreateShape(SearchResult : TECPlaceResult):TECShape;

XapiLayer

XapiLayer uses TECNativePlaceLayer and is directly integrated with TECNativeMap.

The search is performed in the area displayed by your map, it is updated on each move

property **Shapes**:TECShapes

The group that contains the items, it's name is 'XAPI'

property **Search** : string

Research is done by doing a query on a server [XAPI](#)

You can find all the [tags OSM](#) using a syntax like 'node[key=value]'

```
// search bus stop
map.XapiLayer.Search := 'node[highway=bus_stop]';
```

For nodes you can use simplified syntax.

```
// search bus stop
map.XapiLayer.Search := 'highway=bus_stop';
```

For the key [amenity](#) you can simplify even more.

```
// search all restaurant, bar and café
map.XapiLayer.Search := 'restaurant|bar|cafe';
```


property **Visible** : boolean

show / hide layer

property **MaxItem** : integer

Maximum number of items

property **OnClick** : TOnShape

This event is raised when an item is clicked

property **OnChange** : TNotifyEvent

Raised after each request

```
map.XapiLayer.OnClick := doOnXapiClick;
map.XapiLayer.OnChange := doOnXapiChange;
...
procedure TForm1.doOnXapiChange(sender : TObject);
begin
  //
end;

// event click on xapi shape
procedure TForm1.doOnXapiClick(sender : TECshape);
begin
  //
end;
```

Use the [styles](#) to decorate your items

```
Selector := '#' + map.XapiLayer.Shapes.Name +
  '.marker.amenity';

map.styles.addRule(Selector + ':restaurant' +
  '{graphic:base64,iVBORw0KGgoAAAANSUhEUgAAABIAAAABAMAAAABI '
  +
  'sxEJAAAAGXRFWHRTb2Z0d2FyZQBBZG9iZSBJbWFnZVJlYWR5ccllPAAAABJQTFRFAAAA//AAAA '
  +
```

```

'AAAAAAAA/h6U3wAAAAV0Uk5TAAQgL++EJOXAAAAOE1EQVQY02MIFQ0MFWRUCXVigLBcQ0OgrND
+
)JYQIDOMg0NDYawBIFC2FgCSCxBerFMg0Es02AAP34wMx8/aIAAAAAASUVORK5CYII=;visible:t

map.styles.addRule(Selector+':bar' +
' {graphic:base64,iVBORw0KGgoAAAANSUheUgAAABoAAAAaCAYAAACpSkzOAAAZkleQVRix+2W
+
'FIAXFK2ES8HANVMikIGUOkICESUDCJCABCdtPSQgJC2UjeXlZk3402elZoRSIGgZgA3AqfS0tAYg
+
' iFmCxWaDLMaEAsqDQBNGsGlDeHyexnbJPAHwpKuIe9zSwDFzFPR6egM4P9HMgX3YQgDSr67gQrs
+
' z8ZqqqSwHrOkhsZqguAo0iyS5weL1kd5qZUc jPnfXV1HPIThmaY9vr4QH8KknbPXbfMgtiqvSMA+
+
)lrNy9/SK8g0PVamc2NlTK98F1MyKB+QkmGEAAAAASUVORK5CYII=;visible:true;}'

map.styles.addRule(Selector+':cafe' +
' {graphic:base64,iVBORw0KGgoAAAANSUheUgAAABoAAAAaCAYAAACpSkzOA'
+
'AAAuU1EQVRix2NgIACMjY0F0PgN+OTJBkCD9iMbCsT/cclTYkkCzGCofJfuB+D02eUosKQAZA jIM5
+
'JUBsgE2eEotQDIHyHXDJU2JRPyn8wQmgcfCfTPyelHghGw8/i4D4PBrbgVYWYcMGtLAoAakUmQ8V
+
' TOALyFXDGHZUtEqCXRbCy8DzNLAKlNqTMn0ATi0AVI5LYf1ol7/1I7Pk4a18q5Zv3BKsPKpUMCvQ
+
)RGS2+aW0SzGhYAB5lDXBZ7NtoAAAAASUVORK5CYII=;visible:true;}'

map.styles.addRule('#'+map.XapiLayer.Shapes.Name+

);marker.highway:bus_stop {color:blue;styleicon:Flat;visible:true;}'
map.styles.addRule('#'+map.XapiLayer.Shapes.Name+
'.marker {scale:1;}');
map.styles.addRule('#'+map.XapiLayer.Shapes.Name+
'.marker: hover {scale:1.5;}');

```

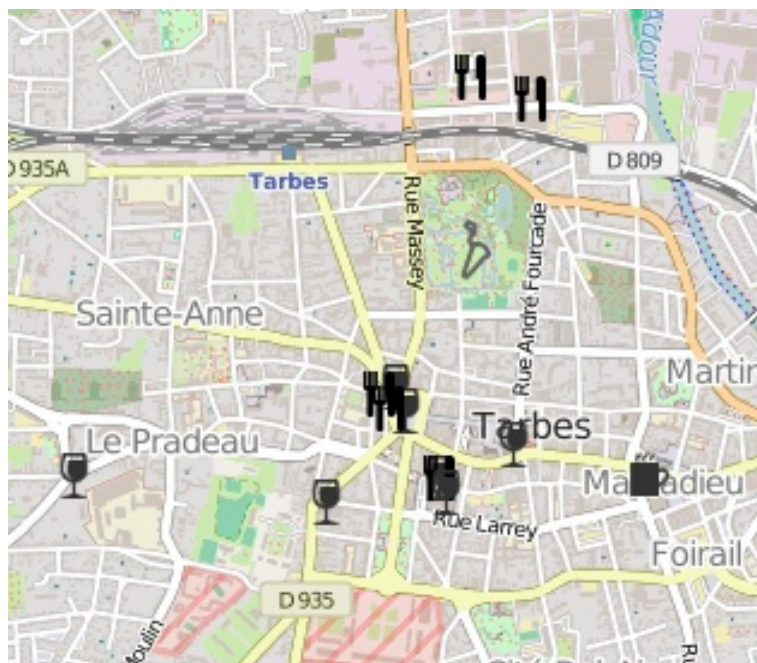


Fig. 137 'restaurant|bar|cafe'

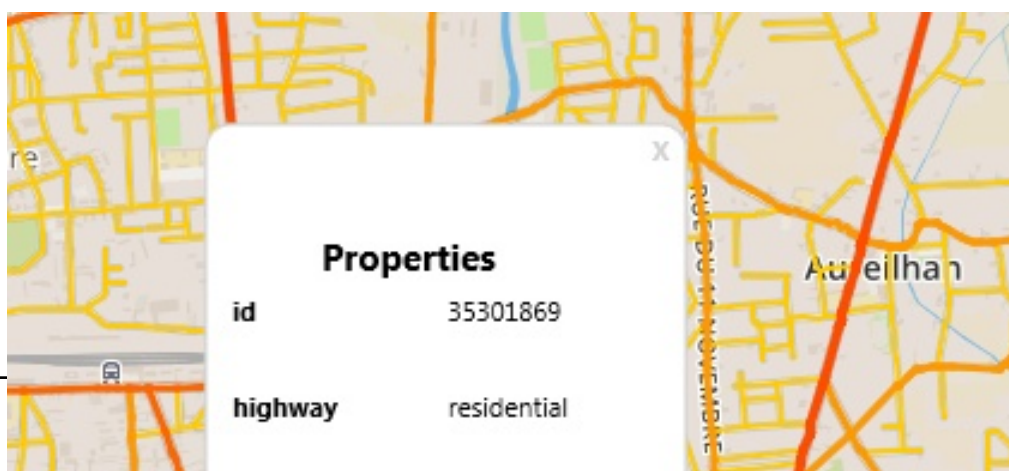


Fig. 138 'highway=bus_stop'

Way

With `way[key = value]` you can retrieve paths.

```
// find roads
map.XapiLayer.Search :=
;way[highway=motorway|primary|secondary|tertiary|residential]'
// styles routes
map.styles.addRule('#' + map.XapiLayer.Shapes.Name +
);line.highway:primary {zindex:9;weight:4;color:gradient(Red,Yellow,0.3);visi
map.styles.addRule('#' + map.XapiLayer.Shapes.Name +
);line.highway:secondary {zindex:8;weight:3;color:gradient(Red,Yellow,0.4);vi
```



You can also find the junctions between routes.

```
map.XapiLayer.Junction := true;
map.XapiLayer.Search :=
;way[highway=unclassified|road|motorway|trunk|primary|secondary|tertiary|resi
```

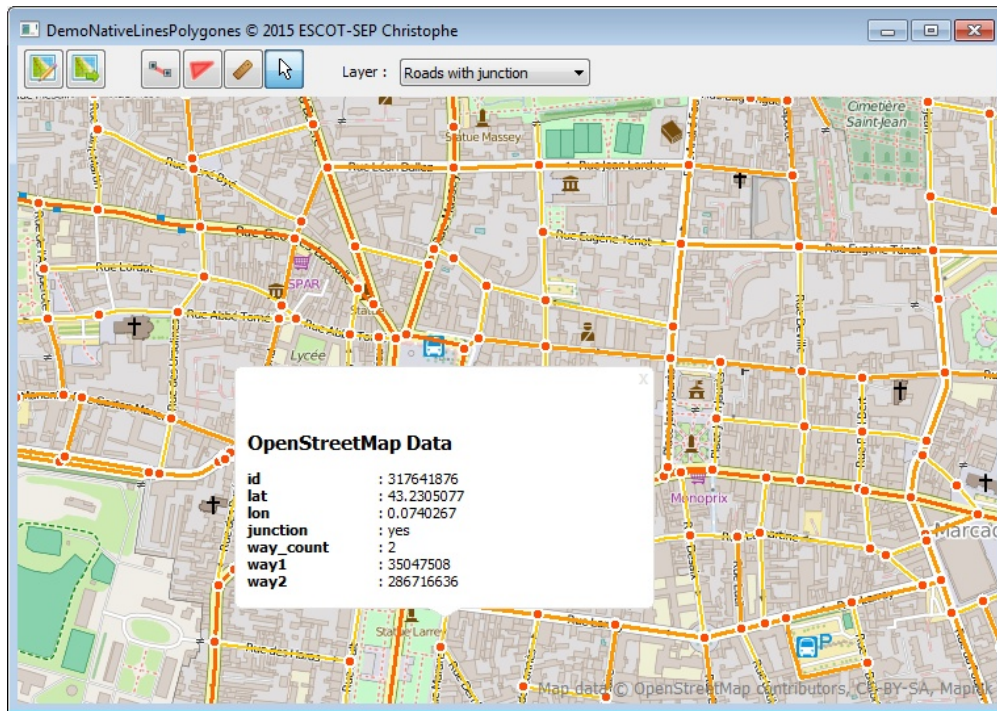


Fig. 140 Xapi Roads & Junction

Manual search

When layer is visible the search is synchronized with the visible area of your map.

To perform a search on any area you need to hide layer and use **Bound** to delimit the search box.

```
// sample, copie data in another group
map.XapiLayer.OnChange := XapiChange;
```

```

map.XapiLayer.visible := false;
map.XapiLayer.search   := 'highway=bus_stop';
    // ! call bound after set search !
map.XapiLayer.bound(43,0.7,44,0.8);
...
TForm.XapiChange(sender : TObject);
begin
    // here xapilayer contain openstreetmap data
    // copie in another group
    map.group[copy_xapi'].ToTxt
:= map.XapiLayer.shapes.ToTxt;
end;

```

TECNativeUTFLayer

This layer gives you support for the format [UTFGrid](#)

Example: Delphi translation of [Visible Map](#)

```

    // support UTFGrid Layer see http://www.mapbox.com/developers/utfgrid/

FUTFLayer := TECNativeUTFLayer.create(map,
'mapbox.geography',GetUTFTile);

FUTFLayer.OnMouseOver := doOnOverUTFData;
FUTFLayer.OnMouseOut  := doOnOutUTFData;

FUTFLayer.Shapes.InfoWindows.add(0,0,'');
FUTFLayer.Shapes.InfoWindows[0].zindex := 100;
FUTFLayer.Shapes.InfoWindows[0].color  := claBeige;
FUTFLayer.Shapes.InfoWindows[0].Width  := 110;
FUTFLayer.Shapes.InfoWindows[0].ContentCenter := true;

FUTFLayer.Visible := true;

...

    // connexion utf geography tiles

    //
http://a.tiles.mapbox.com/v3/mapbox.geography-class/4/7/7.grid.json for see d

procedure TForm1.GetUTFTile(var TileFilename:string;const

```



```

    x,y,z:integer);
begin
    TileFilename := format(
    ;http://%s.tiles.mapbox.com/v3/mapbox.geography-class/%d/%d/%d.grid.json'
        [Char(Ord('a') + random(3
    )),z,x,y]);
end;

// OnOver country

// FUTFLayer.Data['admin']      country name

// FUTFLayer.Data['flag_png']   country flag png base64 encoded

procedure TForm1.doOnOverUTFData(sender : TObject);
begin

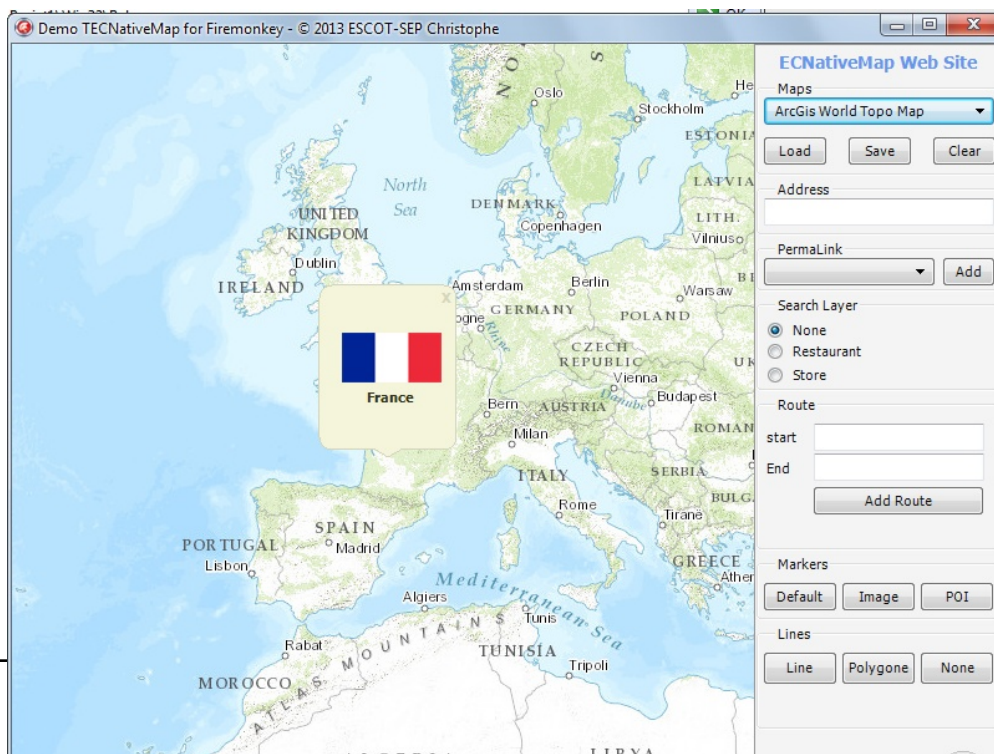
    FUTFLayer.Shapes.InfoWindows[0].Content :=
    '' +
    '<h3>'
    +FUTFLayer.data['admin']+ '</h3>';

    FUTFLayer.Shapes.InfoWindows[0
    ].SetPosition(map.MouseLatLng.Lat,map.MouseLatLng.Lng);
    FUTFLayer.Shapes.InfoWindows[0].Visible := true;

end;

procedure TForm1.doOnOutUTFData(sender : TObject);
begin
    FUTFLayer.Shapes.InfoWindows[0].Visible := false;
end;

```



TECNativeLabelLayer

This layer displays an infoWindow above the TECShape, It is updated and moved automatically depending on the Hint and the position of the TECShape

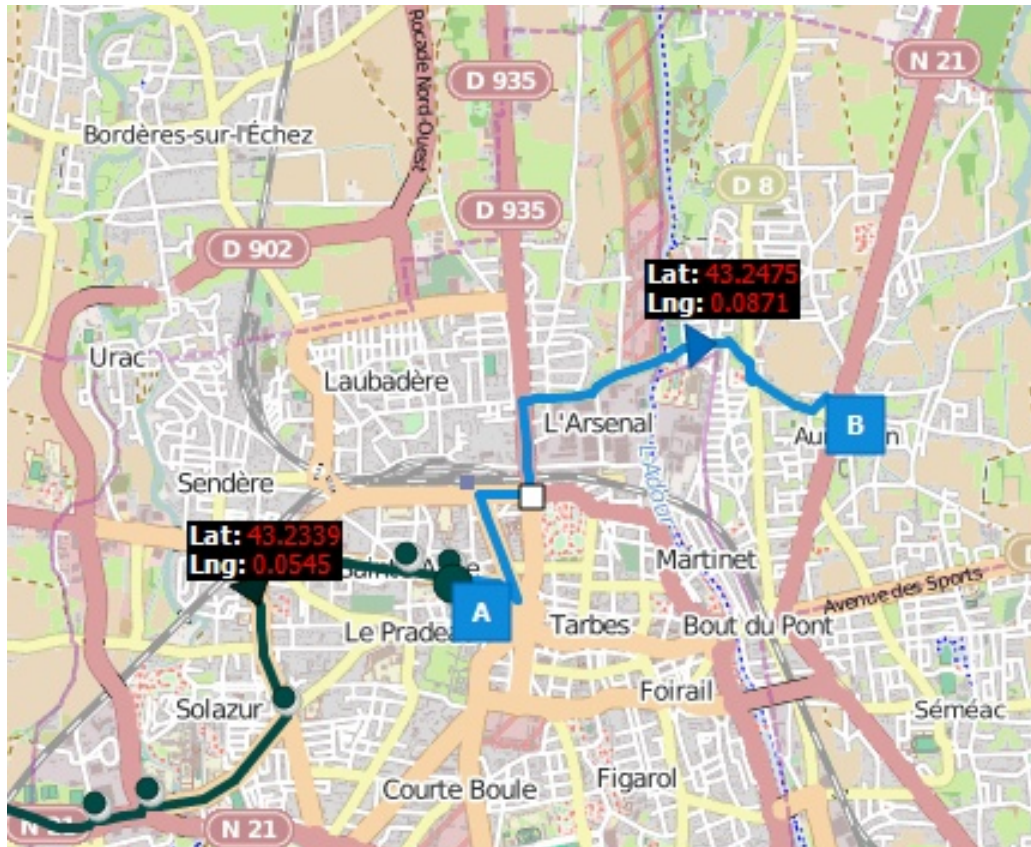


Fig. 142 Using labels to display positions

To use this layer you must embed the unit `uecNativeLabelLayer` or `FMX.uecNativeLabelLayer` Depending on whether you use the version `VCL` or `FireMonkey`

```
LLayer : TECNativeLabelLayer;
```

```

LLayer := TECNativeLabelLayer.Create(Map, 'my layer');

LLayer.visible := true;

// add shapes to layer
LLayer.add(map.Shapes.Markers[0]);
LLayer.add(map.Shapes.Pois[0]);

```

See sources of demos Firemonkey and DemoNativeRoute for an example of using

Mappilary

Mappilary is a service of the same type as Google Street Map, it will display photos at street level, the advantage is that you can enrich it by registering your routes.

TECNativeMap just use the information in public access, you must obtain a key to use the service of Mappilary.

To do this go to the www.mapillary.com website, log in, in the **Integrations** section add your applications, you will then get a **Client ID**, this is your key.

*To use mappilary you must incorporate the unit **uecMappilary** or **FMX.uecMappilary** depending on whether you use the **VCL** or **FireMonkey** version*

Use **TECMappilaryLayer** to embed a layer Mappilary.

```

FMappilaryLayer := TECMappilaryLayer.Create(map);
FMappilaryLayer.ClientId := 'HERE-YOUR-CLIENT-ID';

```



```
FMappillaryLayer.Visible := true;
```

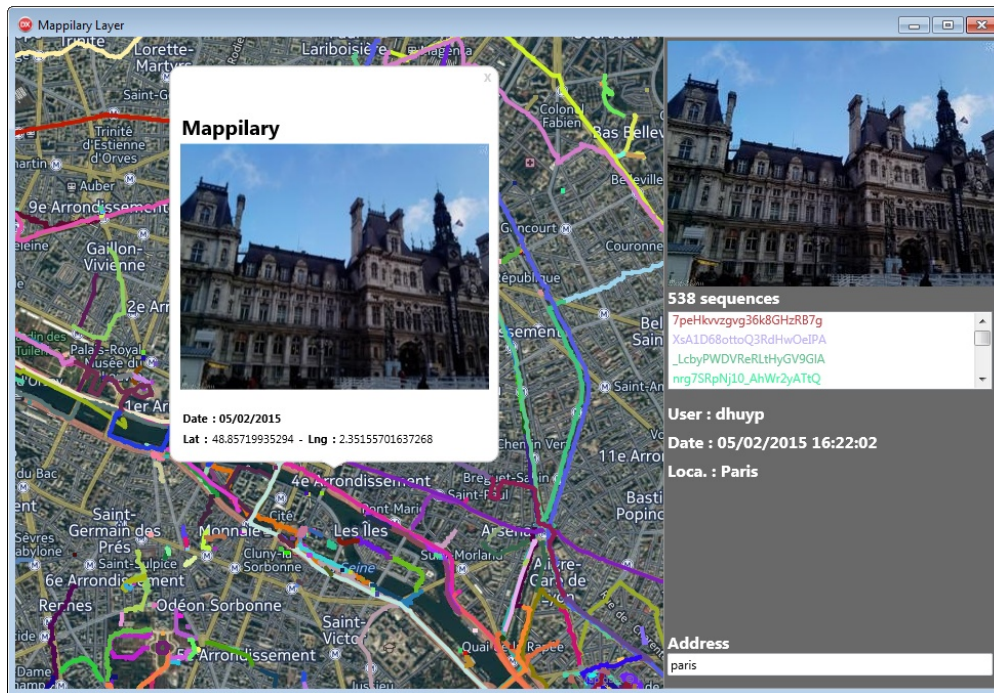


Fig. 143 Demo Mappillary

TECMappillaryLayer

procedure CancelSearchSequence;

Cancels a search for sequences.

procedure CancelSearchImageClose;

Cancels a search for images

procedure SearchImageClose(const Lat, Lng: double; Distance: integer);

Initiates a search of images at a maximum Distance, in metres, of the point Lat,Lng

You can connect to the **OnImages** event to be notified of a response.

The **Images** property contains the list of images.

function FindImageClose(const Lat, Lng: double; Distance: integer; var seq : TMapillarySequence; var PhotoIndex : integer) : integer;

Find the nearest photo amongst those displayed in the layer, the research is local.

Lat, Lng indicates the starting point of the research

Distance indicates the maximum distance in metres from the search point

Seq will contain the found sequence or nil

PhotoIndex will contain the index of the photo in seq or - 1

Returns the distance in metres between the point of research and the photo, 0 if no photo

function LoadImage(const Key: string): TBitmap; overload;

Gets a bitmap that contains the image we had the key.

Caution you must not release the resulting bitmap, to keep it you must make a copy !

function LoadImage(const Key: string; Bitmap:TBitmap):boolean;

Load a bitmap with the image .

Please note you are responsible for the creation and release of the bitmap !

```
var bmp:TBitmap;
...
bmp := TBitmap.create;
if LoadImage(key,bmp) then
begin
    // image ok
end;
...
bmp.free;
```

procedure **LoadImage**(const Key: string; OnBitmap: TOnGetMappylImage); overload;

Gets a bitmap containing the image associated with the key, **This function is not blocking**, **OnBitmap** est appelé lorsque l'image est chargée.

```
layer.onImages := doOnImage;
...
procedure
  TForm2.doOnImage(sender:TMapillaryImageList);
begin
  // you can test sender.More to see if there are
  other images waiting
end;
```

function **LoadImage**(img:TMapillaryImage):boolean; overload;

Gets the information of an image

```
// get data image
img := TMapillaryImage.create;
try
  // ! mandatory set key !
  img.Key := PhotoKey;

  if FECMapillaryLayer.LoadImage(img) then
    begin
      user.Text      := img.User;
      date.Text
:= DateTimeToStr(img.CapturedDateTime);
      location.Text := img.Location;
      // img.ca = angle of image
      // img.lat = latitude
      // img.lng = longitude
    end;

  finally
    img.Free;
  end;
```

function **UrlImage**(const key:string):string;

Gets the url of image associated with the key.

function **LocalPathImage**(const Key:string):string;

Gets the path to an image in the local cache

property **CanceledSearchSequence** : boolean ;

Lets you know if a search of sequences has been cancelled.

property **CanceledSearchImageClose** : boolean ;

Lets you know if a search has been cancelled.

property **ClientId**: string ;

Mandatory key to use the services of Mappilary

property **Connexion** : TMappilaryConnexion ;

Type of connection

mcOnlyMappilaryServer : only by internet

mcOnlyLocal : only in local

mcLocalOrServer : local and if need internet (default value)

property **LocalCache**: string ;

The local cache directory

property **Thumb**: TMappilaryThumbSize ;

Size of images (Thumb320, Thumb640, Thumb1024, Thumb2048)

by default **Thumb320**

property **Sequences**: TMappilarySequenceList ;

List of sequences found in the visible area of the map

The property is updated automatically to each movement of the map

property **Images** : TMapillaryImageList;

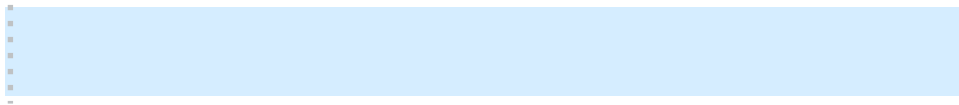
List of images obtained with a call to **SearchImageClose**

property **MaxDayInCache** : integer;

Lifetime of the data stored in the local cache, by default 30 days

property **OnClick**: TOnMAppillaryClick ;

Raised when click on an element mappillary



property **OnSequenceColor**:TOnMappilaySequenceColor;

To assign a color to a sequence.

By default a single color is assigned based on the key of the sequence, a sequence will have therefore always the same color.

```

Layer.OnSequenceColor := doSequenceColor;

procedure TForm2.doSequenceColor(Layer
: TECMappillaryLayer;

                                MappillarySequence: TMapillarySequence;
                                var SequenceColor:TColor);

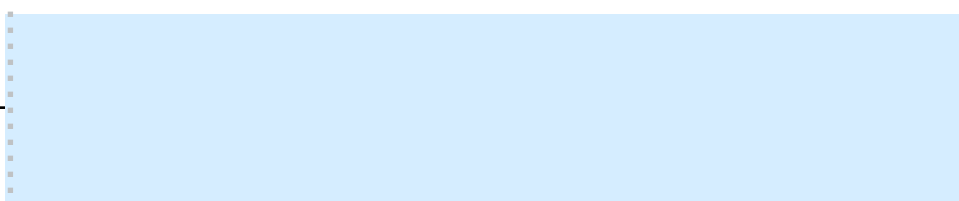
begin
    SequenceColor := your_color;
end;

```

property **OnSequences**: TOnMappilartySequences ;

Raised when the sequences are found.

The research of sequence is triggered when moving the map



```

procedure
  TForm2.doOnSequences(sender:TMapillarySequenceList);
var i,n:integer;
begin

  n := sender.Count;
  TotalSequences.Text := inttostr(n)+' sequences';

  sequences.Items.BeginUpdate;
  sequences.Items.Clear;

  for i := 0 to n-1 do
  begin
    sequences.Items.Add(sender.Sequences[i].Key)  ;
  end;

  sequences.Items.EndUpdate;

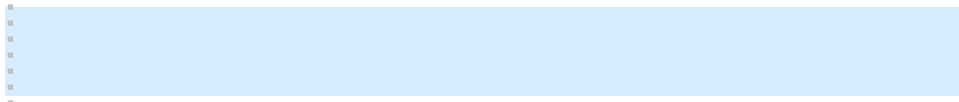
  // test sender.More to see if there are other
  sequences in the area

end;

```

property **OnImages** : TOnMapillaryImages;

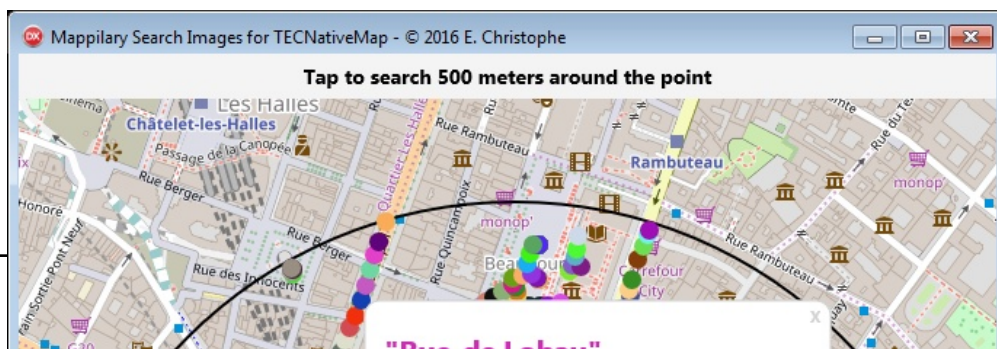
Raised when the images are received after a call to SearchImagesClose



TMapillaryAPI

Internally, TMapillaryLayer uses the TMapillaryAPI class, you can use it directly if you do not want to display layer.

The operation is the same, it is necessary to indicate the search area with the function box **Bound**(NELat, NELng, SWLat, SWLng: double) then call **SearchSequence** to start the search.



Heatmap

A heat map is used to represent the intensity of data for geographic points.

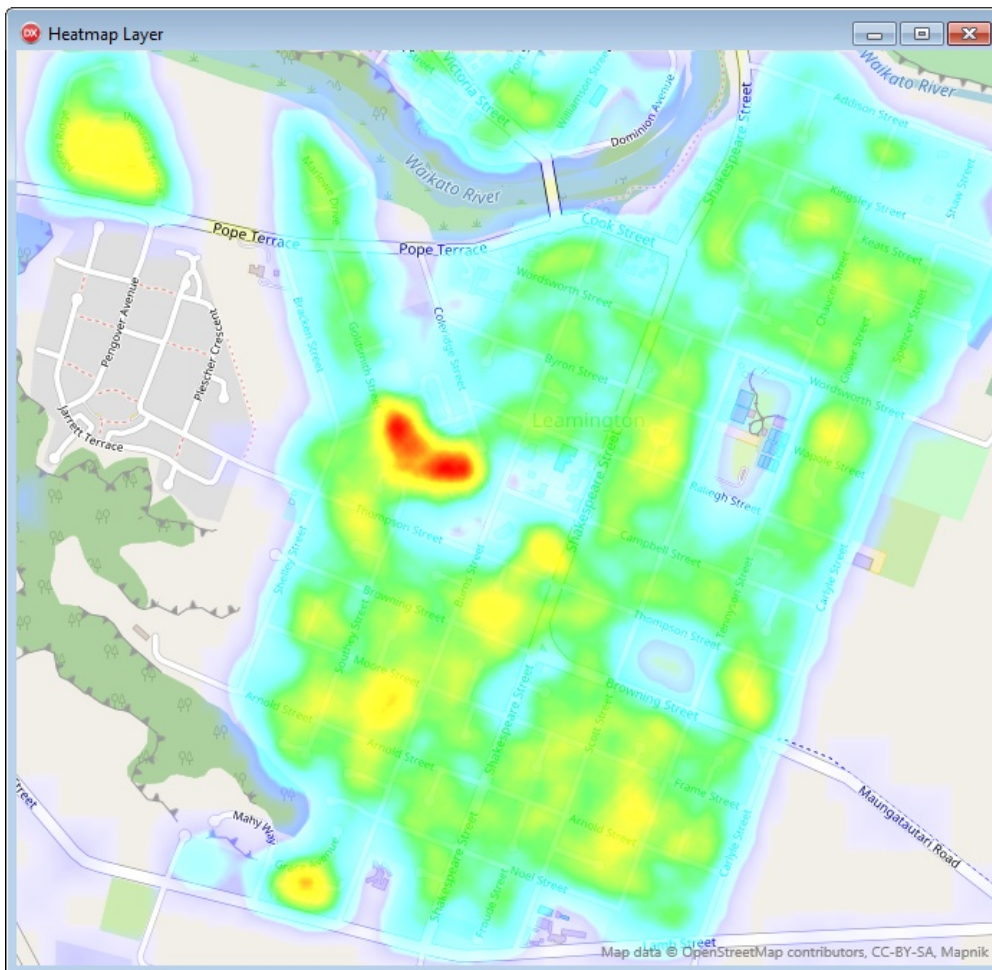


Fig. 145 Heatmap layer

TECHeatmapLayer is the class that allows to manage this type of layer.

constructor **Create(_FMap: TECNativeMap);**

```
// create heatmap layer
HeatmapLayer := TECHeatmapLayer.Create(map);
```


procedure Clear;

Erase all of the data

procedure Add(const Latitude, Longitude: double; const value: double = 1);

Add a geographical point and assign it a value, you can make several additions to the same point.

procedure Remove(const latitude, longitude: double; const Value: double);

Delete the Value element if it exists at these coordinates

procedure Update;

After adding the points, call Update to refresh the layer

property AutomaticUpdate: boolean

By default false, switch to true if you want the HeatMap to be automatically updated each time you change it.

property Palette: THeatPalette

Palette allows you to manage the colors that are used to create the gradient

```
HeatmapLayer.Palette.Clear;

// addColor(Red,Green,Blue,Value)
// Red,Green, and Blue byte 0..255
// value double 0..1

// this is the default palette, you can also use
// Palette.reset for recreate it

HeatmapLayer.Palette.AddColor(0,0,0,0);
// black for value=0
HeatmapLayer.Palette.AddColor(0,0,255,0.1);
// blue for value=0.1
```



```
HeatmapLayer.Palette.AddColor(0,255,255,0.25); // cyan
for value=0.25
HeatmapLayer.Palette.AddColor(0,255,0,0.5);
// green for value=0.5
HeatmapLayer.Palette.AddColor(255,255,0,0.75); // yellow
for value=0.75
HeatmapLayer.Palette.AddColor(255,0,0,1);
// red for value=1
```

property **Visible**: boolean ;

property **PointIsDisc** : boolean;

Determines if the point is displayed in the form of a disc, otherwise it's a square

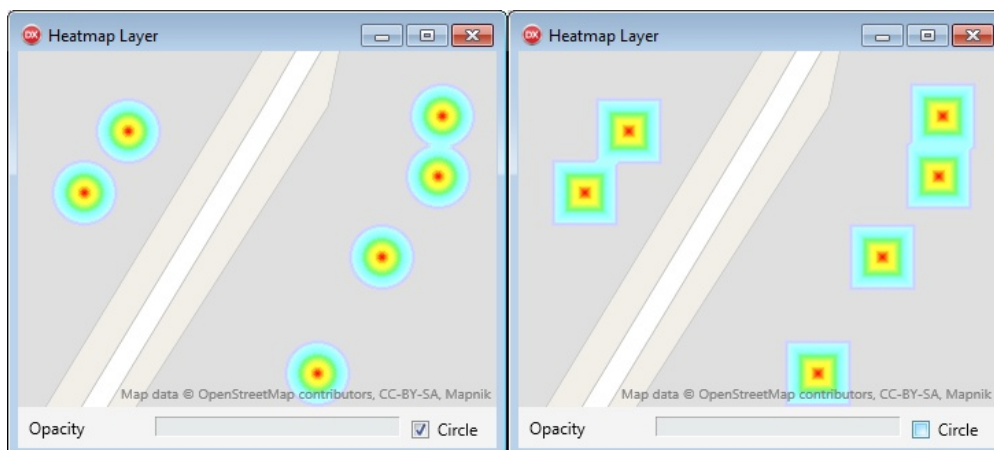


Fig. 146 Style of the point

Property **Radius**:integer;

The point size

property **Opacity** : byte;

Change the opacity of the layer (0 to 100)

property **MinZoom**: byte ;

Minimum zoom layer to display

property **MaxZoom**: byte ;

Maximum zoom layer to display

property **GroupZIndex**: integer ;

ZIndex of the group containing all of the heatmaps, is displayed in ascending order of the ZIndex

property **ZIndex**: integer ;

ZIndex of the layer compared to the other heatmap

property **OnUpdate**: TNotifyEvent;

Event fired when the heat map was generated

Weather Layer

By using the services of [OpenWeathermap.org](https://openweathermap.org) you will be able to see overlay layers owPrecipitation, owSnow, owClouds, owPressure, owTemp, owWind

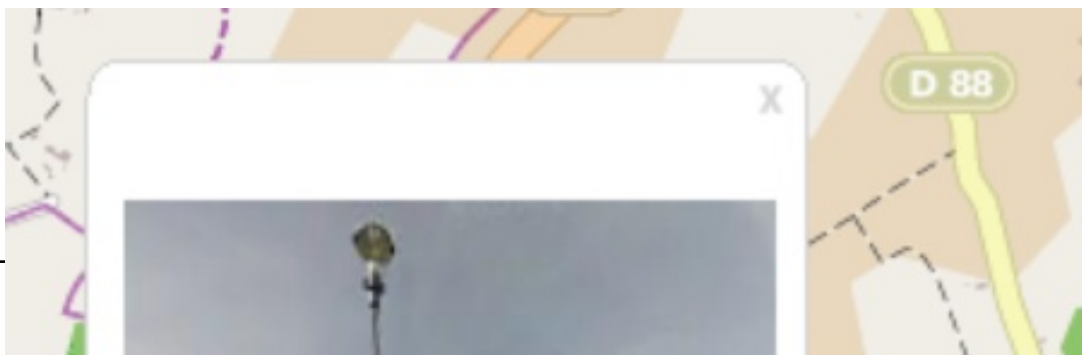
```
map.OpenWeatherTilesLayer.Key := 'your api key';  
    // see pressure  
map.OpenWeatherTilesLayer.Add(owPressure) ;  
    // see precipitation  
map.OpenWeatherTilesLayer.Add(owPrecipitation) ;  
    // remove pressure  
map.OpenWeatherTilesLayer.remove(owPressure) ;
```

ECNativeMap is available for **Android**, you can [download and install the demo apk](#).



Fig. 147 Demo ECNativeMap for Android

Long press on the map add a marker, its appearance is random, you can move it and click on it to obtain its address.





Zoom in and out



By rotation you can change the orientation of your map

Switch "**Find me**" to display your location



Allows you to go to a destination, or to create a route by filling the 2 fields

paris

Arrived

Cancel Navigate

Fig. 149 Goto Paris

Automatic monitoring will then be put in place, click the triangle to the deactivate or re-activate





Displays images panoramio, satellite and restaurants.

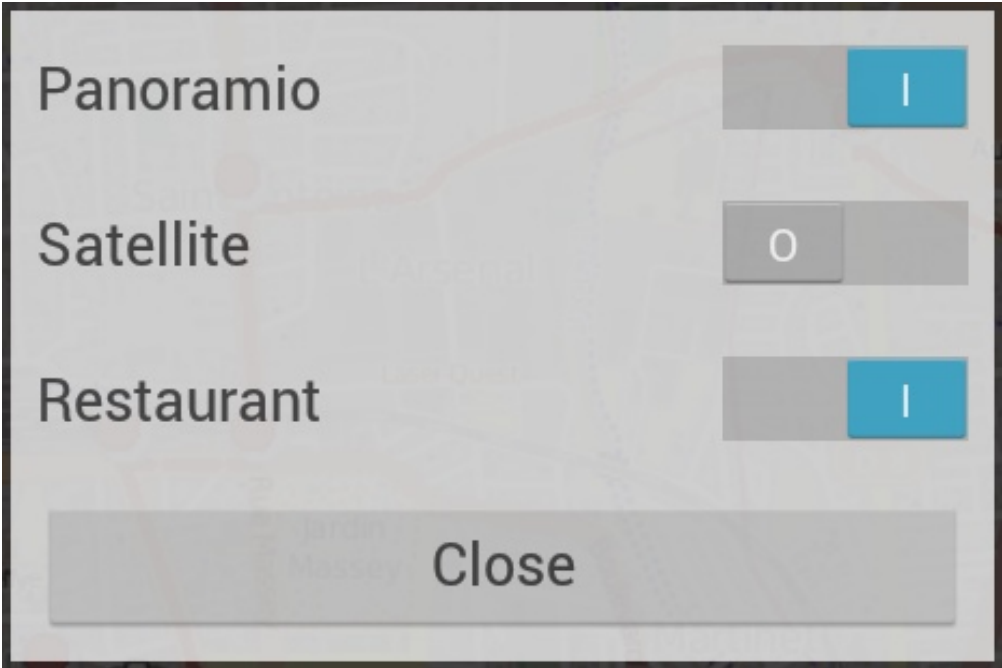
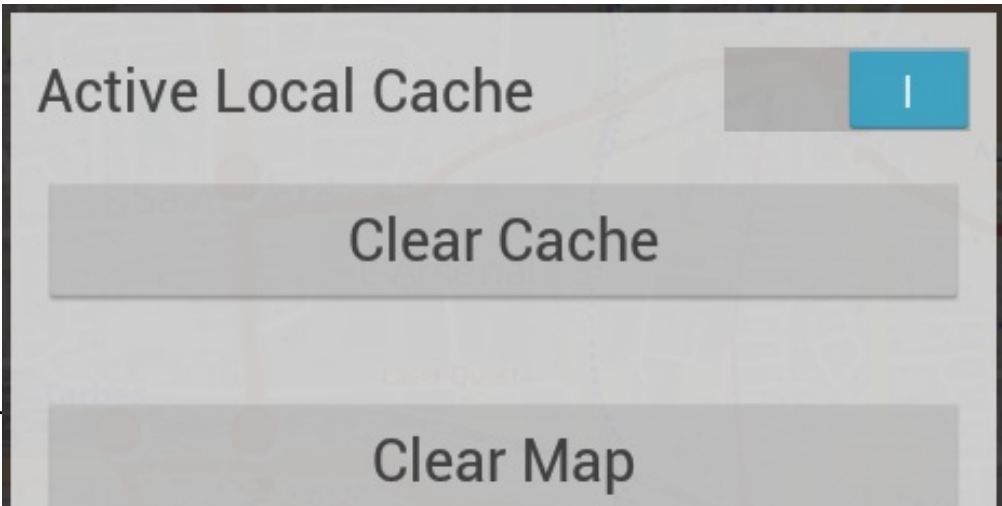


Fig. 151 Select Layers



To manage the cache, erase or save the map, manage the zoom effects , inertia and the [high-resolution](#) mode.



You can [download an apk for android](#) to a mini demo that displays [the OSM data](#) for a small area.



Fig. 153 Android Test OLT

TECNativeMap is also available for **iOS** !

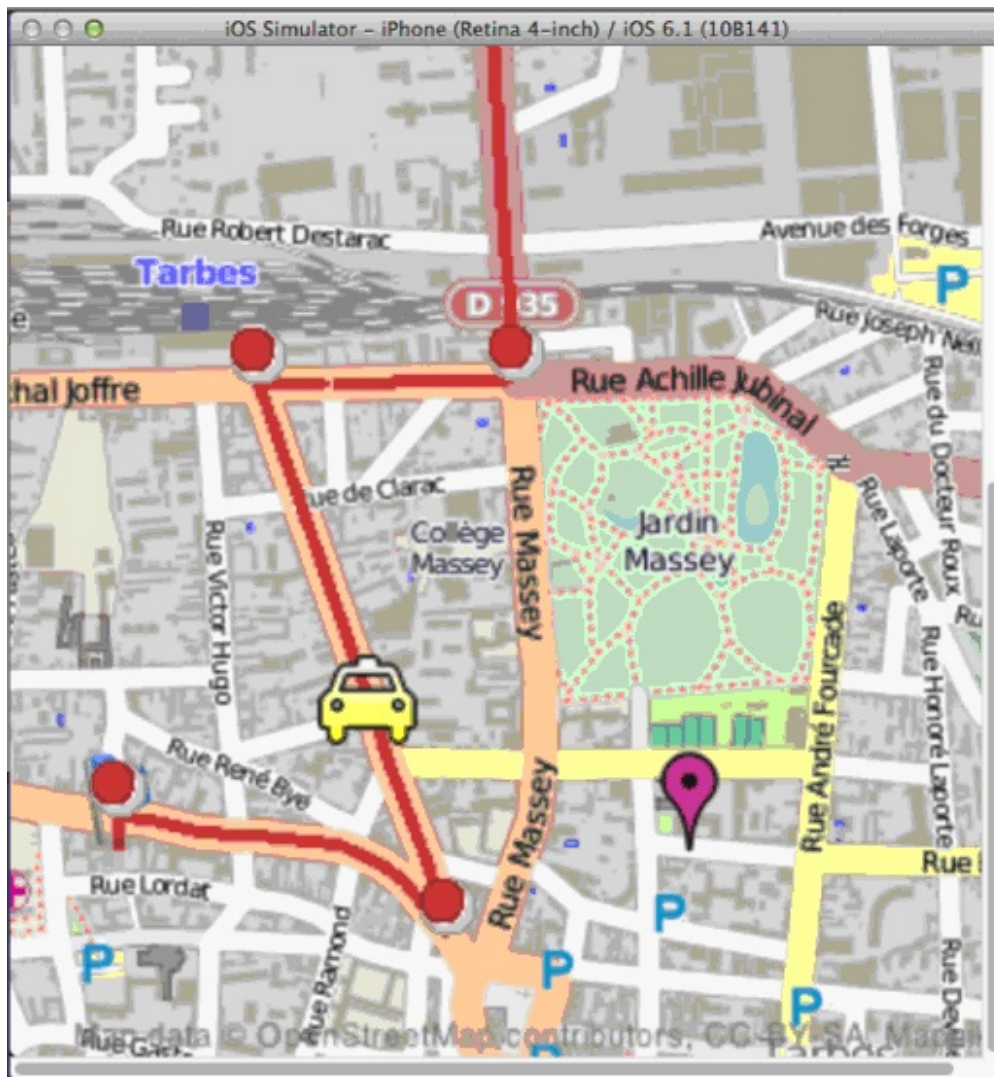


Fig. 154 ECNativeMap on iOS simulator

Other components available with TECMap

[TECCesiumMap](#)

Cesium is a Javascript library to manage globes 3D and 2D maps, information about the site www.cesiumjs.org[Lire](#)

[TECMapAdressEdit](#)

TECMapAdressEdit is a descendant of TEdit component that provides automatic support for the address entry

[Lire](#)

[TECStaticMap](#)

TEStaticMap is a descendant of Timage which will allow you to view a static map using the services of [Google Map](#), [Google StreetView](#) and [OpenMapQuest](#)

[Lire](#)

[TECVelib](#)

TECVelib is a class that gives you access to the api of [Self-service bicycles JCDecaux \(Vélib\)](#)

[Lire](#)

Cesium is a Javascript library to manage globes 3D and 2D maps, information about the site www.cesiumjs.org

TECCesiumMap is a component that allows you to use [Cesium](#) in your Delphi developments, it is at the moment available for the VCL (Windows)

Internet Explorer 11 is the minimum required to use Cesium, TECCesiumMap does not work on XP !



Fig. 155 TECCesiumMap

This component is integrated in full suite TECMap

Installation

Default TECCesiumMap uses cesiumjs.org servers, but for the production of your developments I strongly advise you to install on your own webserver cesium.

Download the latest archive of Cesium then unzip on your server.

TECesiumMap has the property **WebServer** just set with the root directory of your installation.

*Use **CreateRuntimeCesiumMap** to a dynamic creation*

```
// function CreateRuntimeCesiumMap(AOwner: TComponent=nil;AParent:TWinControl=nil;AWebServer:string='');
cesium := CreateRuntimeCesiumMap(self,panel1);
```

You must get a key to use the api of Cesium , connect you to the OnInitialize event to connect your Token.

```
map.OnInitialize := doInitialize;
...

procedure TForm.doInitialize(Sender: TObject);
begin
    map.IonAccessToken := 'your cesium ion token';
end;
```

TCesiumControl

The structure is the same that for `TECNativeMap`, elements work in the same way and you find the concepts of `groups`, `geolocation` and the `calculation of routes` are identical

```
function Add(const shape: TNativeShape; const Lat, Lng: double; const
GroupName: string = ""): TECCesiumShape;
```

```
function AddRoute(const routePath: TECroutepath; const GroupName:
string = ""): TECCesiumShapeLine;
```

```
function AddPOI(const Lat, Lng: double; const GroupName: string = "")
: TECCesiumShapePOI;
```

```
function AddPOIText(const Lat,Lng:Double;const Text:string="";const
GroupName:string=""): TECCesiumShapePOI;
```

```
function AddMarker(const Lat, Lng: double; const GroupName: string =
") : TECCesiumShapeMarker;
```

```
function AddLine(const Lat, Lng: double; const GroupName: string = "")
: TECCesiumShapeLine;
```

```
function AddPolygone(const Lat, Lng: double; const GroupName:
string = "") : TECCesiumShapePolygone;
```

```
function AddInfoWindow(const X,Y: double; const GroupName: string
= "") : TECCesiumShapeInfoWindow;
```

*Unlike `TECNativeMap` in `TECCesiumControl` you specify
the screen coordinates for positioning your
information window*

procedure **Remove**(const shape: TECCesiumShape);

procedure **Clear**

procedure **BeginUpdate**

procedure **EndUpdate**

procedure **SaveToFile**(const Filename: string);

procedure **LoadFromFile**(const Filename: string)

procedure **setCenter**(const Lat, Lng: double);

procedure **fitBounds**(const NELat, NELng, SWLat, SWLng: double);

procedure **fitBoundsRadius**(const dLat, dLng, dRadiusKm: double);

procedure **FlyTo**(const Lat, Lng, alt: double);

procedure **JumpTo**(const Lat, Lng, alt: double);

procedure **LookAt**(const Lat, Lng, alt: double);

procedure **GECamera**(const Lat, Lng, alt, Tilt, Heading, roll: double);

procedure **Javascript**(const js:string);

function **GetRoutePathByAdress**(const StartAdress, EndAdress:
string; const routeType: TMQRRouteTyp = rtFastest; const params: string
= "): TECroutepath;

function **GetRoutePathFrom**(const dLatLngs: array of double; const

routeType: TMQRouteType = rtFastest; const params: string =
"): TECroutepath;

function **GetASyncRoutePathByAdress**(const StartAdress,
EndAdress: string; const routeType: TMQRouteTyp = rtFastest; const
params: string = "): TECroutepath;

function **GetASyncRoutePathFrom**(const dLatLngs: array of double;
const routeType: TMQRouteType = rtFastest; const params: string =
"): TECroutepath;

property **Address**: string

property **MouseLatLng**: TLatLng

property **MouseAlt**: double

property **ScreenShot**: TBitmap

property **ClickX** : integer read FClickX;

X position (coordinate screen) click

property **ClickY** : integer read FClickY;

Position in Y (coordinate screen) click

property **MouseX** : integer read FMouseX;

X position (coordinate screen) of the mouse

property **MouseY** : integer read FMouseY;

Position in Y (coordinate screen) of the mouse

property **Shapes**: TECCesiumShapes read FShapes;

property **HintInfoWindow** : TECCesiumShapeInfoWindow
read getHintInfoWindow;

Provides access to the TECCesiumShapeInfoWindow that manages the ToolTips of items
useful to change the style

property **Group**[value: string]: TECCesiumShapes
read getShapesGroup;

property **Groups**:TECGroupShapesList read getShapesGroups;

property **toKml**: string read getToKml write setToKml;

property **toGpx**: string read getToGPX write setToGpx;

property **toTxt**: string read getToTxt write setToTxt;

property **toGeoJSON**: string read getToGeoJSON write setToGeoJSON;

property **Url**: string read getUrl write setUrl;

property **WebServer** : string read FWebServeur write setWebServeur;

[Your Cesium server URL](#)

property **BingKey**: string

property **Latitude**: double ;

property **Longitude**: double ;

property **Altitude**: double;

property **Zoom**: double;

property **Draggable**;

property **TileServer**: TTileServer

The following servers are available :

- tsOpenMapQuest
- tsOSM
- tsOpenCycleMap
- tsArcGisWorldTopoMap
- tsArcGisWorldStreetMap
- tsArcGisWorldImagery
- tsBingRoad
- tsBingAerial
- tsBingAerialLabels

It is advisable to fill the **BingKey** property with [your key Bing](#) to use Bing Maps tiles (tsBingRoad, tsBingAerial, tsBingAerialLabels)

```
map.BingKey      := YOUR_BING_KEY
map.TileServer   := tsBingRoad;
```

property **Terrain**: boolean;

Activates the relief of the terrain

property **SceneMode**: TCesiumMode;

Selects the type of display, by default 3D

- cm2D
- cmColombus
- cm3D

Événements

property **OnMapClick**: TOnMapLatLng

property **OnMapRightClick**: TOnMapLatLng

property **OnMapMove**: TOnMapLatLng

property **OnMapMouseMove**: TOnMapLatLng

property **OnMapMouseUp**: TOnMapLatLng

property **OnMapMouseDown**: TOnMapLatLng

property **OnShapeMove**: TOnShapeMove

property **OnShapeDrag**: TOnShapeMove

property **OnShapeDragEnd**: TNotifyEvent

property **OnShapeMouseOver**: TOnShapeMouseEvent

property **OnShapeMouseOut**: TOnShapeMouseEvent

property **OnShapeMouseDown**: TOnShapeMouseEvent

property **OnShapeMouseup**: TOnShapeMouseEvent

property **OnShapeClick**: TOnShapeMouseEvent

property **OnShapeRightClick**: TOnShapeMouseEvent

property **OnShapePathChange**: TNotifyEvent

property **OnCloseInfoWindow**: TOnCloseInfoWindow

property **OnRoutePath**: TOnRoutePath

property **OnLoad**: TOnLoadGroup

property **OnLoadShapes**: TOnLoadShapes

property **OnUrl**: TOnUrl

property **OnCZML** : TOnCZML

TECCesiumShapeMarker

```
i := cmap.Shapes.Markers.Add(lat,lng) ;  
  // hint accept html  
  cmap.Shapes.Markers[i].Hint := 'marker n° <b>' +  
    +inttostr(i)+'</b>' ;
```



```

uses uecCesiumShape;

var marker : TECCesiumShapeMarker;

    // use the shortcut to create the marker
    marker := cmap.AddMarker(Lat,lng);
    // you can find index with marker.IndexOf
    // cmap.shapes.markers[marker.IndexOf]

    // add a text
    marker.Description := 'Text';
    // adapt width
    marker.Width := 70;
    // change color
    marker.Color := clOlive;

```

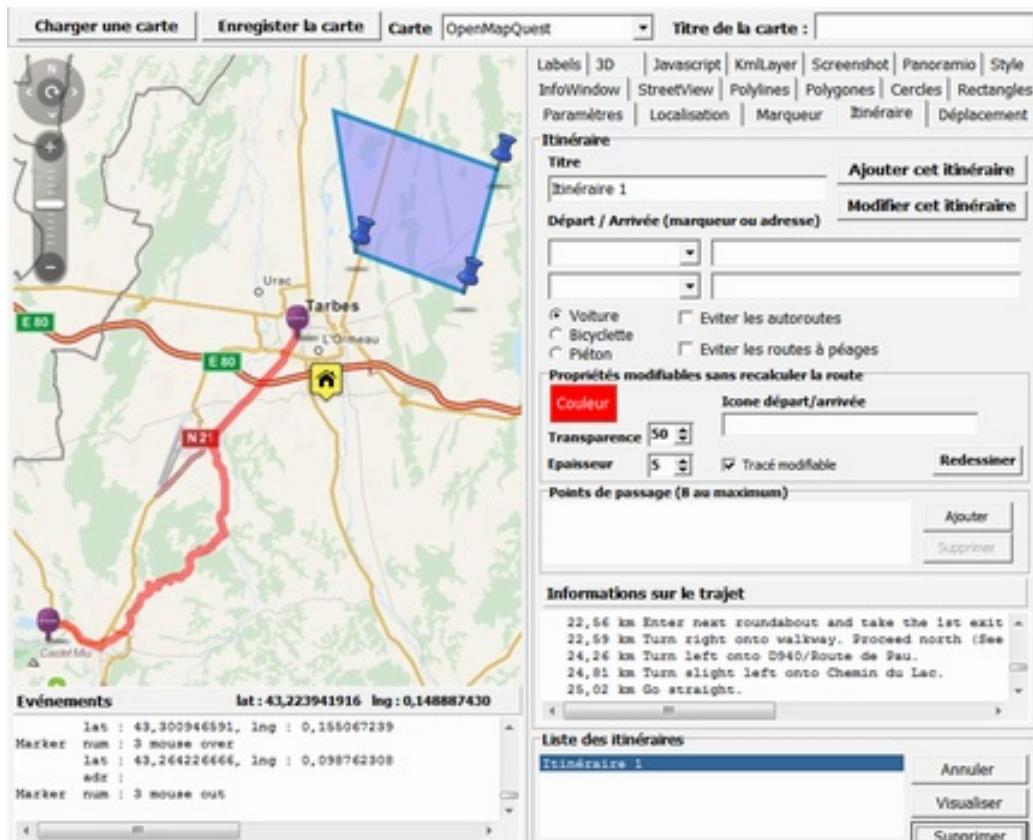


Fig. 157 Marker with Text

```

    // use the shortcut to create the marker
    marker := cmap.AddMarker(Lat,lng);

```

```
// use maki icon
marker.Description := 'maki:bicycle';
marker.Width := 64;
marker.Color := clBlue;
// set draggable
marker.Draggable := true;
```

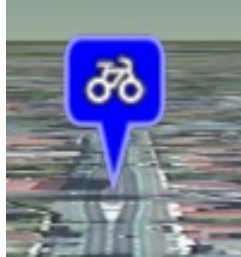


Fig. 158 Marker with Maki icon

```
// use the shortcut to create the marker
marker := cmap.AddMarker(Lat,lng);
// use png file
marker.Filename :=
'http://www.helpandweb.com/cake_32.png';;
```

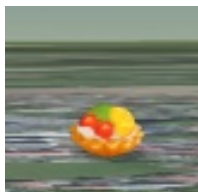


Fig. 159 Marker with Png

TECCesiumShapeLine

```
var line : TECCesiumShapeLine;

// create line and add first point Latitude, Longitude
line := cmap.AddLine(48.8594069779731,2.34283447265625) ;
// better for optimisation
line.BeginUpdate;
// change color
line.Color := clHotLight;
// change size
line.Weight := 10;
// add other points
line.Add(48.8638113489879,2.35519409179688);
line.Add(48.8608751447095,2.36532211303711);
line.Add(48.8543245298006,2.36360549926758);
// update
line.EndUpdate;
// zoom to line
line.fitBounds;
```



Fig. 160 TECCesiumShapeLine

TECCesiumShapePolygone

```

var polygone : TECCesiumShapePolygone;

    // create polygone and add first point Latitude, Longitude
polygone := cmap.AddPolygone(48.8594069779731,
2.34283447265625) ;
    // better for optimisation
polygone.BeginUpdate;
    // change color
polygone.fillColor := clYellow;
    // change opacity
polygone.fillOpacity := 30;
    // add other points
polygone.Add(48.8638113489879,2.35519409179688);
polygone.Add(48.8608751447095,2.36532211303711);
polygone.Add(48.8543245298006,2.36360549926758);
    // update
polygone.EndUpdate;
    // zoom to polygone
polygone.fitBounds;

```



Fig. 161 TECCesiumPolygone

TECCesiumShapePOI

Unlike TECNativeMap only poiEllipse and poiText types are supported at the moment

```
var Poi      : TECCesiumShapePOI;  
  
Poi := cmap.AddPOI(lat,lng) ;  
  
Poi.BorderSize := 2;  
Poi.Color      := clBlue;  
Poi.BorderColor := clWhite;  
// 16 pixels  
// for meters use Poi.POUnit := puMeter;  
Poi.Width := 16;
```



Fig. 162 TECCesiumShapePOI

```
var Poi : TECCesiumShapePOI;  
  
Poi := cmap.AddPOIText(lat,lng,'my label') ;  
  
// use description for change text  
//Poi.Description := 'my new label';  
  
// font  
Poi.CssFont := '14pt monospace';  
// color  
Poi.Color := clWhite;
```



Fig. 163 poiText

Material

You can use the material property for example to embed an image in a polygon.

```
Cesium.Shapes.polygons[index].Material :=  
  'https://cesiumjs.org/images/2015/02-02/cats.jpg';
```

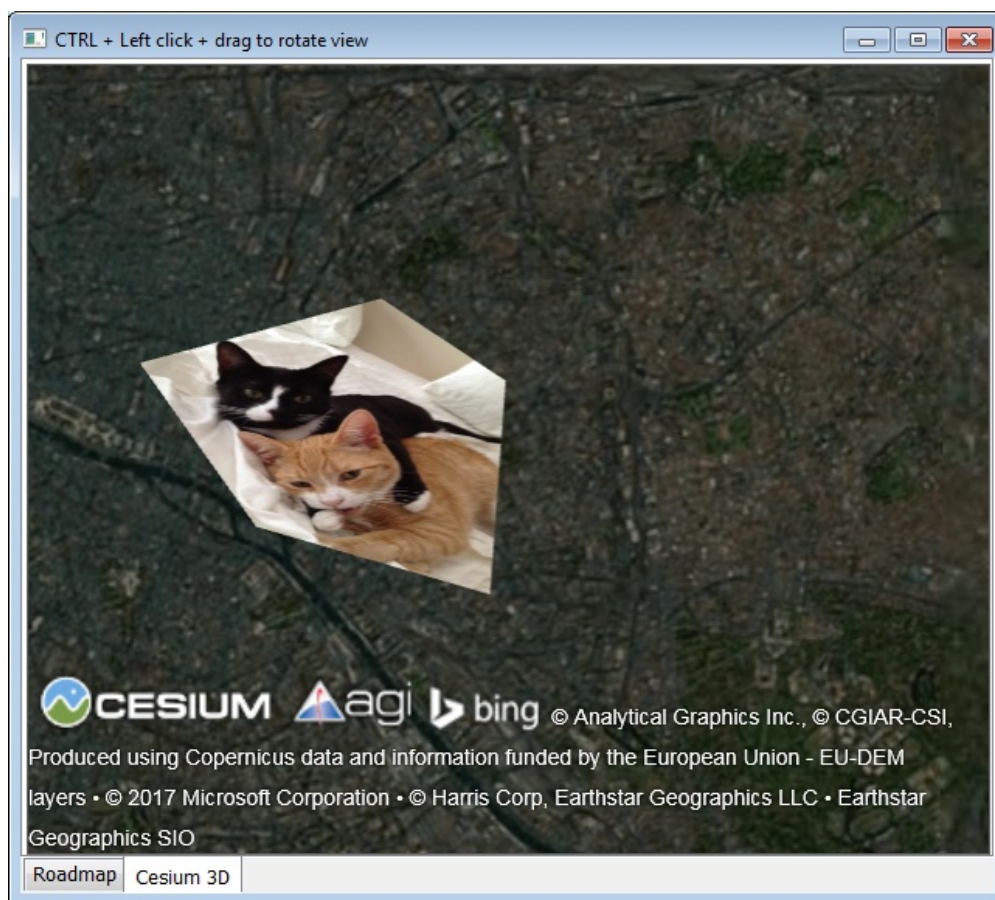


Fig. 164 material property

You can directly use javascript to define a more complex material.

```
Cesium.Shapes.polygons[index].Material :=  
  'new Cesium.ImageMaterialProperty({' +
```

```

        'image' : ''+UriImageToDataUri(
        'd:\alert-weather.png')+''+', '+
        'transparent:true }')';

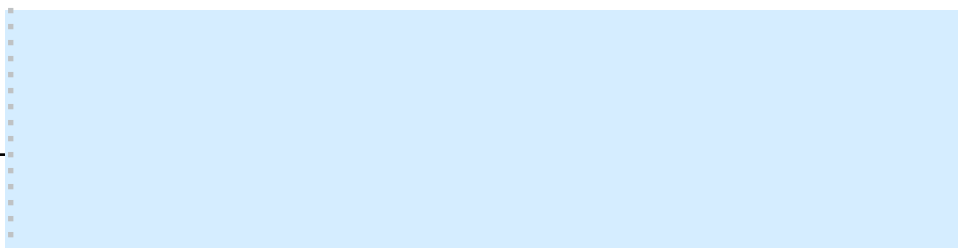
```



Fig. 165 material local image

TECCesiumShapeInfoWindow

Unlike TECNativeMap the position of the InfoWindows is shown in screen coordinates, you can use html for your content and use CSS to style your InfoWindows




```

var wn:TECCesiumShapeInfoWindow;

Wn := cmap.AddInfoWindow(10,10) ;
Wn.content := 'You can enter <b>html</b> content <br>text

and <i>image</i> '
// you can style with css
Wn.Style := 'here css';
// use color for color background
Wn.Color := clWhite;

Wn.CloseButton := true;

```

The **OnCloseInfoWindow** of the TECCesiumMap component event is raised when closing by the cross and the **InfoWindow** coming to be closed is recovered.

If your window contains a link, the click triggers the event

OnUrl

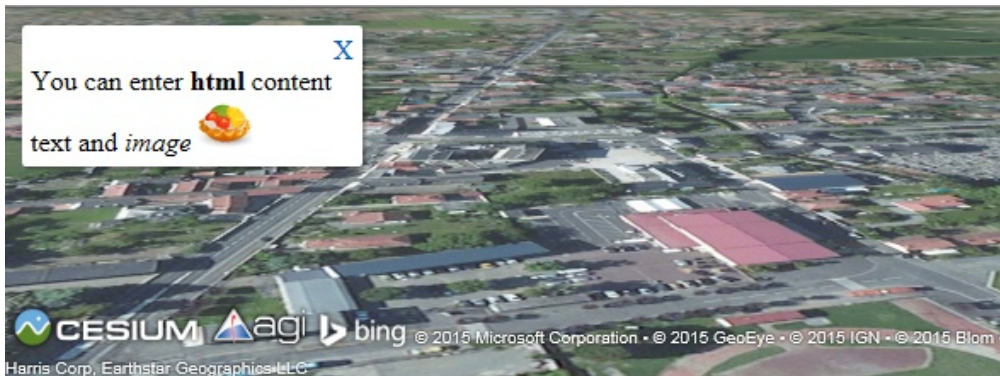


Fig. 166 InfoWindow

TECCesiumShapeModel

This class manages the 3D models, Cesium uses [the glTF format](#), a [converter](#)

COLLADA-> glTF is proposed

```
var model : TECCesiumShapeModel;

// you can use url or local filename for your .gltf
model := cmap.addModel(lat,lng,
)http://cesiumjs.org/Cesium/Apps/SampleData/models/CesiumGround/Cesium_Ground

// Values greater than 1.0 increase the size of the model
// while values less than 1.0 decrease it.
model.Scale := 2.0;

// specifying the approximate minimum pixel size of the
model regardless of zoom.
// This can be used to ensure that a model is visible
even when the viewer zooms out
model.MinimumPixelSize := 64;

// Orientation(heading,pitch,roll)
model.Orientation(90,model.Pitch,model.Roll);
```



[Download démos !](#)

CZML

CZML is the equivalent of KML but more powerful

procedure **LoadCZML**(const CZML:string;const idCZML:string="";const

```
Options:string="";const Promise:string="");
```

For options see

[//cesiumjs.org/Cesium/Build/Documentation/CzmlDataSource.html](http://cesiumjs.org/Cesium/Build/Documentation/CzmlDataSource.html) (Cesium.CzmlDataSource options))

Promise can execute javascript when the czml script is finished,
LoadAndZoomToCZML use.

```

procedure TCesiumEngine.LoadAndZoomToCZML(const CZML:
string; const idCZML: string = ''; const Options: string = '');
var promise: string;
begin

    promise := 'viewer.zoomTo(datasourceczml);'

    LoadCZML(CZML, idCZML, Options, promise);

end;

```

When the script is finished the event OnCZML of your component is triggered, it returns idCZML

```
procedure LoadAndZoomToCZML(const CZML:string;const
idCZML:string="";const Options:string="");
```

```
czml := [{ "id" : "document", '+
          "name" : "CZML Point", '+
          "version" : "1.0"'+
    }, { '+
      "id" : "toto", '+
      "name": "pointx", '+
      "position" : { "cartographicDegrees" : [-111.0, 40.0, 0]'+
        }, '+
      "point": { '+
        "color": { "rgba": [255, 255, 255, 255] }, '+
        "outlineColor": { "rgba": [255, 0, 0, 255]'+
        }, '+
        "outlineWidth" : 4, '+
        "pixelSize": 20'+
      }'+
    }
```

```
'}}';
```

```
Cesium.LoadAndZoomToCZML(czml, 'add_point');
```



Fig. 168 create point with czml

TECMapAdressEdit is a descendant of TEdit component that provides automatic support for the address entry

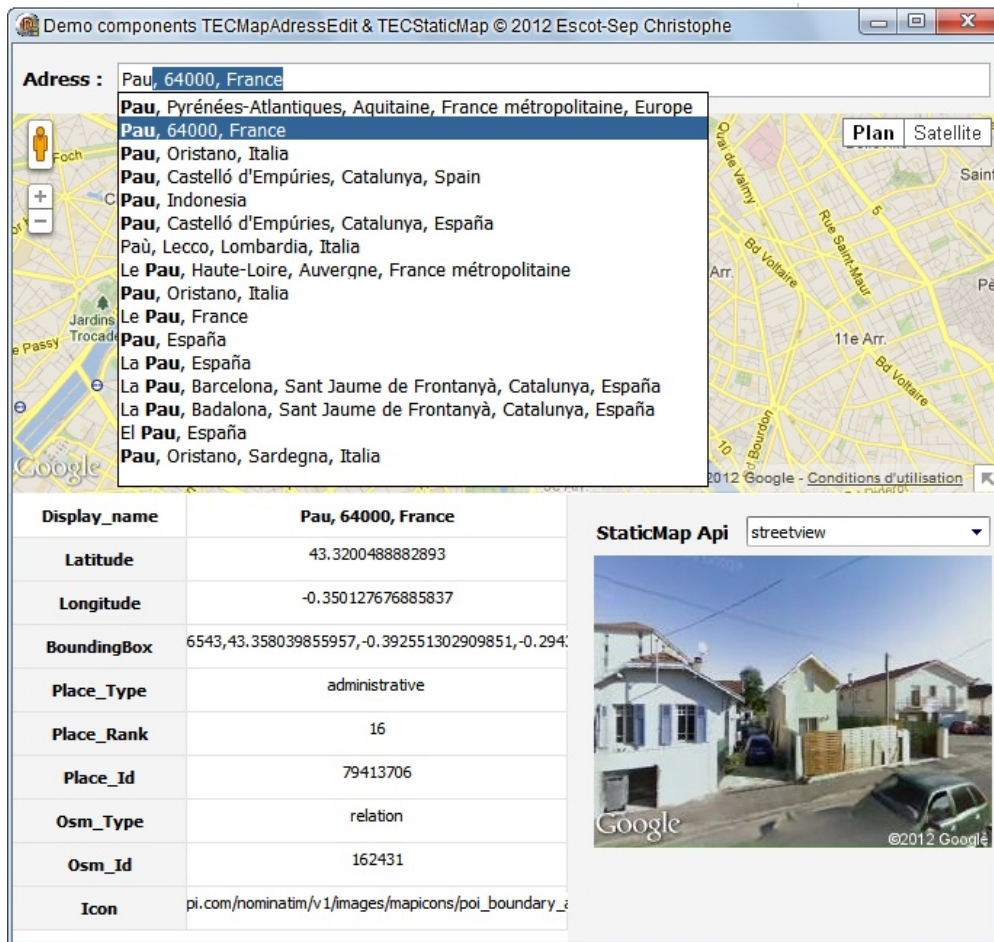


Fig. 169 TECMapAdressEdit in action

When you preselect in the list a proposal the **OnSelect** event is raised

When you valid an address, either by pressing the **return** key or clicking a proposal with the left button of the mouse, the **OnAddress** event is raised

You have access to the **Nominatim** property that contains the following information about the selected address

- TAdressEdit.nominatim.**Display_name**
- TAdressEdit.nominatim.**Latitude**
- TAdressEdit.nominatim.**Longitude**
- TAdressEdit.nominatim.**BoundingBox**
- TAdressEdit.nominatim.**Place_type**
- TAdressEdit.nominatim.**Place_rank**
- TAdressEdit.nominatim.**Place_id**
- TAdressEdit.nominatim.**Osm_type**
- TAdressEdit.nominatim.**Osm_id**
- TAdressEdit.nominatim.**Icon**

The addresses are obtained with the [MapQuest Nominatim](#) service

TEStaticMap is a descendant of Timage which will allow you to view a static map using the services of [Google Map](#), [Google StreetView](#) and [OpenMapQuest](#)

Api allows you to select the service to use (**saGoogle**, **saStreetView** ou **saOpenMapQuest**)

You choose the region to display by filling in either the property **address** or **Latitude** and **Longitude** properties

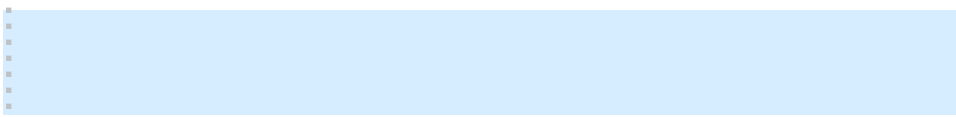
Adress *takes precedence over* **Latitude** *et* **Longitude**,
make sure **address** *is empty if you want to*
use coordinates.

MapType allows you to choose the type of cards (**stMap**, **stTerrain**, **stSatellite**, **stHybrid**) (no effect if you select StreetView)

Zoom gives you the choice of zoom

Params allows you to use additional parameters (ex **Params:=** **'key=API_console_key&scale=2'**) see the different documentations for more info

To display your image use the function **LoadMap** which returns **True** or **False** depending on success or failure



TECVelib is a class that gives you access to the api of [Self-service bicycles JCDecaux \(Vélib\)](#)

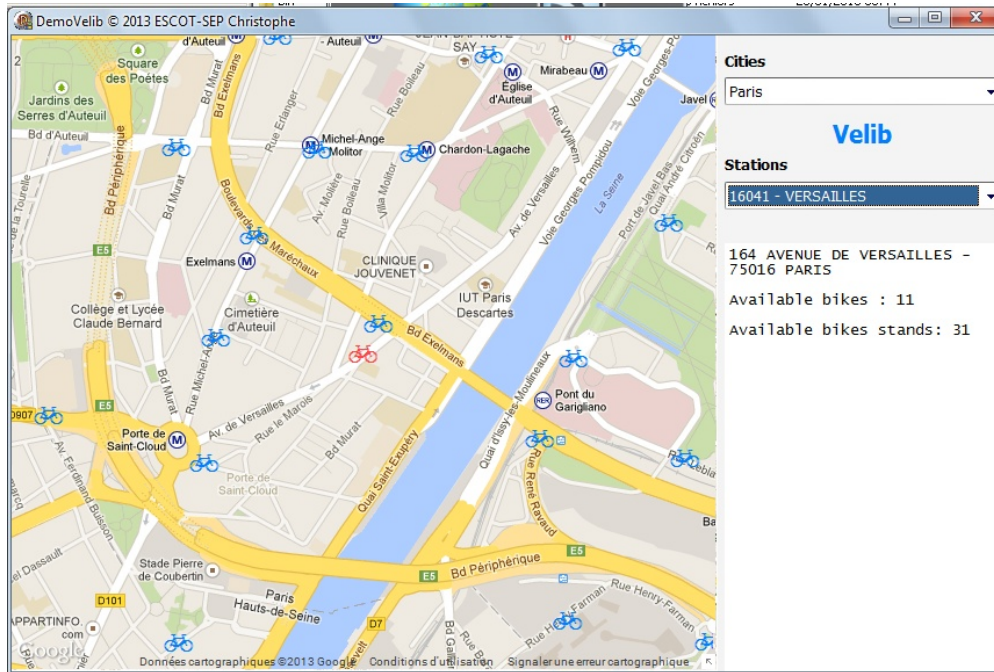


Fig. 170 DemoVelib

TECVelib

function **getContracts** : integer;

Retrieves all the contracts, they are stored in the array **Contracts**

Returns the number of contracts, accessible also through the property **ContractsCount**

function **getAllStations**:integer;

Retrieves all the stations of all contracts, accessible in the table **Stations**

Returns the number of stations

function **getContractStations**(const Contract:string):integer;

Retrieves all of the stations for a contract, available in the table **Stations**
Returns the number of stations

function **InfoStation**(const Contract: string; const Station: TECVelibStation): boolean;

Gets the station a contract-related information

property **ApiKey**: string;

An api key is required, it is free just [register on the site](#)
TECVelib has its own key but it is preferable that each uses its own.

property **Stations** : TECVelibStations;

List of stations (see **getAllStations** et **getContractStations**)

property **Contracts**[index:integer] : TECVelibContractInfo;

List of contracts (see **getContracts**)

property **ContractsCount**:integer;

Number of contracts

Example of use

```
// Delphi map component EMap
// use TECVelib

FECVelib := TECVelib.Create;

// get stations in Paris
FECVelib.getContractStation('Paris');
```

```
        // set stations on map
    ...
    var i      : integer;
        id     : integer;
        Station : TECVelibStation;
        Poi     : TECMapPoi;

    for i:=0 to FVelib.Stations.count-1 do
    begin
        Station := FECVelib.Stations[i];

        id := map.Pois.add(Station.latitude,Station.Longitude);

        Poi := map.Pois[id];

        Poi.caption := Station.Name;
    end;
```

In this chapter we will discuss specific cases and we will see how to use TECMap and its various components with Delphi

[Création interactive d'une route](#)

Let's make a class that will handle a component TECMap with the aim of creating a route by clicking on the map the point of departure and arrival, Once the road created we need to be able to change the mouse

[Read](#)

[Afficher StreetView dans une InfoWindow](#)

Let's see how to display [StreetView](#) in a [InfoWindow](#)

[Read](#)

[Afficher un Layer Panoramio dans toutes les cartes](#)

Google Maps already has a [layer Panoramio](#) , let's see how to wear this feature to all apis.

[Read](#)

[Afficher une MiniMap](#)

Let's see how to embed a mini synchronized map on the main map

[Read](#)

Let's make a class that will handle a component TECMap with the aim of creating a route by clicking on the map the point of departure and arrival, Once the road created we need to be able to change the mouse

An event will be raised whenever the road change and we will then consult the various information of it (distance, duration, crossing points)

```
FInfoRoute      := TInfoRoute.create;
  // event fired when new route or route change
FInfoRoute.OnRoute := OnInfoRoute;
  // route color
FInfoRoute.Color   := clRed;
  // Active InfoRoute
FInfoRoute.map     := map;
  // deactivate InfoRoute
FInfoRoute.map     := nil;
```

When connecting the TECMap component we safeguard the events on which we we connecting.

```
procedure TInfoRoute.setMap(const Map:TECMap);
begin

  Clear;

  RestoreEvents;

  FECMap := Map;

  AssignEvents;

end;

procedure TInfoRoute.AssignEvents;
begin

  if not assigned(FECMap) then exit;

  // save the events of Origins
  FoldMapClick      := FECMap.OnMapClick;
  FoldAfterReload   := FECMap.OnAfterReload ;
  FoldBeforeReload  := FECMap.OnBeforeReload ;
  FoldRouteChange   := FECMap.OnRouteChange;
```

```

    // connect to the events that interest us
    FECMap.OnMapClick      := doOnMapClick;
    FECMap.OnAfterReload   := doOnAfterReload;
    FECMap.OnBeforeReload := doOnBeforeReload;
    FECMap.OnRouteChange   := doOnRouteChange;

end;

// Reassign the events of origins
procedure TInfoRoute.RestoreEvents;
begin

    if not assigned(FECMap) then exit;

    FECMap.OnMapClick      := FOldMapClick;

    FECMap.OnAfterReload   := FOldAfterReload;
    FECMap.OnBeforeReload := FOldBeforeReload;

    FECMap.OnRouteChange   := FOldRouteChange;

end;

```

The main event is **OnMapClick** which allows us to respond when the user clicks on the map.

If the route is not created test if you need to create the marker beginning or end, for each one of them we adjust its coordinates with **AlignLatLngToRoute** to position themselves on the nearest road to the clicked point.

```

procedure TInfoRoute.doOnMapClick(sender: Tobject; const dLatitude,
dLongitude: double);
var id:integer;
    lat,lng : double;
begin
    if not assigned(FECMap) or
    assigned(FPolyline) or
    assigned(FRoute) then exit;

    lat := dLatitude;
    lng := dLongitude;

    // Start Marker

```

```

    if not assigned(FStartMarker) then
    begin
        FECMap.AlignLatLngToRoute(lat, lng);

        FStartLat := lat;
        FStartLng := lng;

        id := FECMap.AddMarker(lat, lng);

        FStartMarker := FECMap.Markers[id];

        FStartMarker.Draggable := true;

        FStartMarker.OnMarkerMove := doOnMoveMarker;

    end

    else    // End Marker

    if not assigned(FEndMarker) then
    begin
        FECMap.AlignLatLngToRoute(lat, lng);

        id := FECMap.AddMarker(lat, lng);

        FEndLat := lat;
        FEndLng := lng;

        FEndMarker := FECMap.Markers[id];

        FEndMarker.Draggable := true;

        FEndMarker.OnMarkerMove := doOnMoveMarker;

        SetRoute;
    end;

end;

```

It is connected to the event **OnMarkerMove** of markers to be able to manually edit the beginning and the end of our route in the case of the apis that do not allow to manage editable routes.

```

procedure TInfoRoute.doOnMoveMarker(sender: Tobject; const
    Index: integer;
                                var dLatitude,
    dLongitude: double);
begin

```

```

    FECMap.AlignLatLngToRoute(dLatitude, dLongitude);

    setRoute;

end;

```

During the creation of the arrival point is called **setRoute** to create our route.

Only the apis **Google Maps** and **OpenMapQuest** to change the road with the mouse, for others we will emulate this function using a **Polyline** and our start and end markers.

```

procedure TInfoRoute.setRoute;
var id : integer;

begin

    if assigned(FECMap) and
        assigned(FStartMarker) and
        assigned(FEndMarker) then

        begin

            if assigned(FPolyLine) then
            begin
                FECMap.Polylines.delete(FPolyLine.id);
                FRoutePath.free;
                FPolyline := nil;
            end;

            if assigned(FRoute) then
            begin
                FECMap.Routes.delete(FRoute.Id);
                FRoute := nil;
            end;

            FRoutePath := nil;

            // only Google and OpenMapQuest support dynamic route
            if (FECMap.MapAPI=apiGoogle) or
                (FECMap.MapAPI=apiOpenMapQuest) then
            begin

```



```

        id := FECMap.AddRoute( ' '
        ,FStartMarker.Latitude,FStartMarker.Longitude,FEndMarker.Latitude,FEndMarker.

        FRoute := FECMap.routes[id];

        // the dynamic routes have their own markers
        // so ours is deleted

        if assigned(FStartMarker) then
        begin
            FECMap.Markers.delete(FStartMarker.id);
            FStartMarker := nil;
        end;

        if assigned(FEndMarker) then
        begin
            FECMap.Markers.delete(FEndMarker.id);
            FEndMarker := nil;
        end;

    end

    else
    begin

        FRoutePath

:= FECMap.getRoutePathFrom([FStartMarker.Latitude,FStartMarker.Longitude,FEnd

        if FRoutePath<>nil then
        begin

            id := map.polylines.addFromRoutePath(FRoutePath);

            FPolyLine := FECMap.PolyLines[id];

            FPolyLine.Color      := FColor;
            FPolyLine.Opacity    := FOpacity;
            FPolyLine.Redraw;

            if assigned(OnRoute) then
                OnRoute(self);
            end;

        end;

    end;

end;

```

```
end;
```

With **Google Maps** or **OpenMapQuest** When the road is changed the event **OnRouteChange** is raised, if the road has just been created we make it editable and we adjust its color.

```
procedure TInfoRoute.doOnRouteChange(sender: TObject;
const idRoute: integer;const NewRoute: boolean);
begin
  if assigned(FRoute) and
    (FRoute.id=idRoute) and
    assigned(OnRoute) then
    begin
      if NewRoute then
        begin
          FRoute.Draggable := true;
          FRoute.Color      := FColor;
          FRoute.Opacity    := FOpacity;
          // updateOptions fire OnRouteChange
          FRoute.updateOptions;
        end
      else
        if assigned(OnRoute) then
          OnRoute(self) ;
        end
      else
        if Assigned(FOldRouteChange) then
          FOldRouteChange(sender,idRoute,NewRoute);
        end;
      end;
```

When the road is created/modified, the event **OnRoute** our class is raised, by you plugging you can retrieve this information through the property **RoutePath** type [TECMapRoutePath](#).

```
RoutePath.Distance; // distance in meters
RoutePath.Duration; // duration in seconds
```

Below the entirety of unit, You may notice events management **OnBeforeReload** and **OnAfterReload** to take account of any map reloading for example when changing API.

```

unit UInfoRoute;

interface
uses Windows, SysUtils, Classes, Graphics, EMaps;

type

TInfoRoute = class
private
    FECMap          : TECMap;
    FStartLat,FStartLng,
    FEndLat,FEndLng  : double;

    FStartMarker,
    FEndMarker      : TECMapMarker;
    FPolyLine       : TECMapPolyline;
    FRoutePath      : TECMapRoutePath;
    FRoute          : TECMapRoute;

    FOldMapClick    : TOnMapClick;
    FOldRouteChange : TOnRouteChange;

    FOldAfterReload,
    FOldBeforeReload,
    FOnRoute        : TNotifyEvent;

    FColor   : TColor;
    FOpacity : double;

    procedure setMap(const Map:TECMap);

    procedure AssignEvents;
    procedure RestoreEvents;

    procedure setRoute;
    function  getRoutePath:TECMapRoutePath;

    procedure doOnMapClick(sender: Tobject; const dLatitude,
dLongitude: double);

    procedure doOnMoveMarker(sender: Tobject; const Index
: integer;
                                var dLatitude, dLongitude:
double) ;

```

```

procedure doOnRouteChange(sender: Tobject; const idRoute:
integer;const NewRoute: boolean);

```

```

procedure doOnBeforeReload(sender: Tobject);

```

```

procedure doOnAfterReload(sender: Tobject);

```

```

public

```

```

constructor Create ;

```

```

destructor Destroy ; override;

```

```

procedure Clear;

```

```

property Map : TECMap read FECMap

```

```

write setMap;

```

```

property RoutePath : TECMapRoutePath read getRoutePath;

```

```

property Color : TColor read FColor write FColor;

```

```

property Opacity: double read FOpacity write FOpacity;

```

```

property OnRoute : TNotifyEvent read FOnRoute

```

```

write FOnRoute;

```

```

end;

```

```

implementation

```

```

constructor TInfoRoute.Create;

```

```

begin

```

```

    FColor := clBlue;

```

```

    FOpacity := 0.5;

```

```

    inherited;

```

```

end;

```

```

destructor TInfoRoute.Destroy ;

```

```

begin

```

```

    Map := nil;

```

```

    inherited;

```

```

end;

```

```

procedure TInfoRoute.AssignEvents;

```

```

begin

```

```

    if not assigned(FECMap) then exit;

```

```

    // save the events of Origins

```

```

FoldMapClick          := FECMap.OnMapClick;
FoldAfterReload       := FECMap.OnAfterReload ;
FoldBeforeReload      := FECMap.OnBeforeReload ;
FoldRouteChange       := FECMap.OnRouteChange;

// connect to the events that interest us
FECMap.OnMapClick      := doOnMapClick;
FECMap.OnAfterReload   := doOnAfterReload;
FECMap.OnBeforeReload  := doOnBeforeReload;
FECMap.OnRouteChange   := doOnRouteChange;

end;

// Reassign the events of origins
procedure TInfoRoute.RestoreEvents;
begin

    if not assigned(FECMap) then exit;

    FECMap.OnMapClick      := FoldMapClick;

    FECMap.OnAfterReload   := FoldAfterReload;
    FECMap.OnBeforeReload  := FoldBeforeReload;

    FECMap.OnRouteChange   := FoldRouteChange;

end;

procedure TInfoRoute.setMap(const Map:TECMap);
begin

    Clear;

    RestoreEvents;

    FECMap := Map;

    AssignEvents;

end;

procedure TInfoRoute.doOnBeforeReload(sender: Tobject);
begin

```

```

    Clear;

    if assigned(FoldBeforeReload) then
        FoldBeforeReload(FECMap);

    end;

    procedure TInfoRoute.doOnAfterReload(sender: TObject);
    var id:integer;
    begin

        if (FStartLat <> 0) and
            (FStartLng <> 0) and
            (FStartMarker = nil) then
            begin
                id := FECMap.AddMarker(FStartLat, FStartLng);

                FStartMarker := FECMap.Markers[id];

                FStartMarker.Draggable := true;

                FStartMarker.OnMarkerMove := doOnMoveMarker;

                if (FEndLat <> 0) and
                    (FEndLng <> 0) then
                    begin

                        id := FECMap.AddMarker(FEndLat, FEndLng);

                        FEndMarker := FECMap.Markers[id];

                        FEndMarker.Draggable := true;

                        FEndMarker.OnMarkerMove := doOnMoveMarker;

                        SetRoute;

                    end;

            end;

        end;

        if assigned(FoldAfterReload) then
            FoldAfterReload(FECMap);
    end;

    procedure TInfoRoute.doOnMapClick(sender: TObject; const dLatitude,
    dLongitude: double);
    var id:integer;
        lat,lng : double;
    begin

```

```

    if not assigned(FECMap) or
      assigned(FPolyline) or
      assigned(FRoute) then exit;

    lat := dLatitude;
    lng := dLongitude;

    if not assigned(FStartMarker) then
    begin
      FECMap.AlignLatLngToRoute(lat, lng);

      FStartLat := lat;
      FStartLng := lng;

      id := FECMap.AddMarker(lat, lng);

      FStartMarker := FECMap.Markers[id];

      FStartMarker.Draggable := true;

      FStartMarker.OnMarkerMove := doOnMoveMarker;

    end

    else

    if not assigned(FEndMarker) then
    begin
      FECMap.AlignLatLngToRoute(lat, lng);

      id := FECMap.AddMarker(lat, lng);

      FEndLat := lat;
      FEndLng := lng;

      FEndMarker := FECMap.Markers[id];

      FEndMarker.Draggable := true;

      FEndMarker.OnMarkerMove := doOnMoveMarker;

      SetRoute;
    end;

  end;

procedure TInfoRoute.doOnMoveMarker(sender: Tobject; const
  Index: integer;
                                var dLatitude,
  dLongitude: double);
begin

```

```

        FECMap.AlignLatLngToRoute(dLatitude, dLongitude);

        setRoute;

    end;

procedure TInfoRoute.setRoute;
var id    : integer;

begin

    if assigned(FECMap)      and
        assigned(FStartMarker) and
        assigned(FEndMarker)  then

        begin

            if assigned(FPolyLine) then
            begin
                FECMap.Polylines.delete(FPolyLine.id);
                FRoutePath.free;
                FPolyline := nil;
            end;

            if assigned(FRoute) then
            begin
                FECMap.Routes.delete(FRoute.Id);
                FRoute := nil;
            end;

            FRoutePath := nil;

            // only Google and OpenMapQuest support dynamic route
            if (FECMap.MapAPI=apiGoogle) or
                (FECMap.MapAPI=apiOpenMapQuest) then
            begin

                id := FECMap.AddRoute(' '
                ,FStartMarker.Latitude,FStartMarker.Longitude,FEndMarker.Latitude,FEndMarker.

                FRoute := FECMap.routes[id];

                // the dynamic routes have their own markers
                // so ours is deleted

```



```

        if assigned(FStartMarker) then
        begin
            FECMap.Markers.delete(FStartMarker.id);
            FStartMarker := nil;
        end;

        if assigned(FEndMarker) then
        begin
            FECMap.Markers.delete(FEndMarker.id);
            FEndMarker := nil;
        end;

    end

    else
    begin

        FRoutePath

:= FECMap.getRoutePathFrom([FStartMarker.Latitude,FStartMarker.Longitude,FEnd

        if FRoutePath<>nil then
        begin

            id := map.polylines.addFromRoutePath(FRoutePath);

            FPolyLine := FECMap.PolyLines[id];

            FPolyLine.Color      := FColor;
            FPolyLine.Opacity    := FOpacity;
            FPolyLine.ReDraw;

            if assigned(OnRoute) then
                OnRoute(self);
            end;

        end;

    end;

end;

procedure TInfoRoute.doOnRouteChange(sender: Tobject;
const idRoute: integer;const NewRoute: boolean);
begin
    if assigned(FRoute)    and
        (FRoute.id=idRoute) and
        assigned(OnRoute)  then
    begin

```

```

    if NewRoute then
    begin
        FRoute.Draggable := true;
        FRoute.Color      := FColor;
        FRoute.Opacity    := FOpacity;
        // updateOptions fire OnRouteChange
        FRoute.updateOptions;
    end

    else if assigned(OnRoute) then
        OnRoute(self) ;
    end

    else

    if Assigned(FoldRouteChange) then
        FoldRouteChange(sender,idRoute,NewRoute);

    end;

function TInfoRoute.getRoutePath:TECMapRoutePath;
begin
    if Assigned(FRoute) then
        result := FRoute.Path
    else
    if Assigned(FRoutePath) then
        result := FRoutePath
    else
        result := nil;
    end;

procedure TInfoRoute.Clear;
begin
    if not assigned(FECMap) then exit;

    if assigned(FStartMarker) then
    begin
        FECMap.Markers.delete(FStartMarker.id);
        FStartMarker := nil;
    end;

    if assigned(FEndMarker) then
    begin
        FECMap.Markers.delete(FEndMarker.id);
        FEndMarker := nil;
    end;

    if assigned(FPolyLine) then

```

```
begin
  FECMap.Polylines.delete(FPolyLine.id);
  FPolyLine := nil;
  FRoutePath.free;
  FRoutePath := nil;
end;

if assigned(FRoute) then
begin
  FECMap.Routes.delete(FRoute.Id);
  FRoute := nil;
end;

end;

end.
```

Let's see how to display **StreetView** in a **InfoWindow**

StreetView is available only with **Google Maps api !**

The first step is to create our **InfoWindow**, We store the index in `idStreetWindow`

Its contents will be a component **DIV** who must have a **unique name** here `"streetwindow"`

It must also define its size, 300 pixels in height and width.

```
idStreetWindow := map.InfoWindows.Add( '<div
id="streetwindow" style="width:300px;height:300px"></div>'
);
```

We connecting events **OnInfoWindowOpen** and **OnInfoWindowClose** to show/hide **StreetView** When **InfoWindow** will be open or closed.

```
map.OnInfoWindowOpen := mapInfoWindowOpen;
map.OnInfoWindowClose := mapInfoWindowClose;
```

In these events we use javascript to do the desired job.

```
procedure TForm.mapInfoWindowOpen(sender: TObject;
  const Index: Integer);
var js:string;
begin
if index=idStreetWindow then
begin
  js :=
    // get latitude and longitude of InfoWindow
    'var LatLng = new google.maps.LatLng('
+doubletostr(map.InfoWindows[Index].Latitude)+' , '
+doubletostr(map.InfoWindows[Index].Longitude)+' );'+

    'var pano    = new
```

```

google.maps.StreetViewPanorama(document.getElementById("streetwindow"), {
    'position:LatLng'+
    '});'+
    'pano.setVisible(true)';

    // run javascript
    map.Javascript(js);
end;

procedure TForm.mapInfoWindowClose(sender: TObject; const
Index: Integer);
begin

    if index=idStreetWindow then
    begin
        map.javascript('pano.setVisible(false);pano = null;');
    end;
end;

```

To open the InfoWindow you can either react directly when clicked on the map as you plugging into OnMapClick

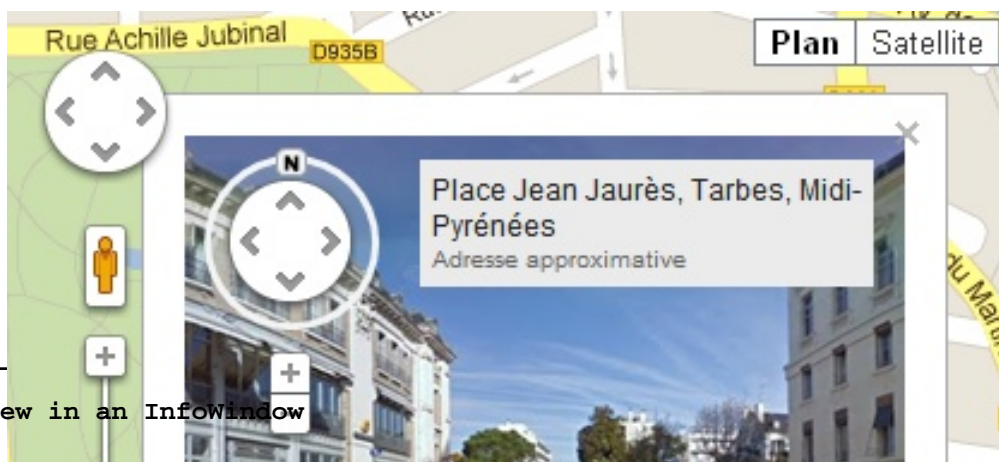
```

procedure TForm.mapMapClick(sender: TObject;const dLatitude,
dLongitude: Double);
begin

    map.InfoWindows[idStreetWindow].SetPosition(dlatitude,dlongitude);
    map.InfoWindows[idStreetWindow].Open := true;
end;

```

But Alternatively, you can associate it with a marker and then it will open when clicked on it



Let's create a class on the same model as that manages the [interactive road design](#) .

The principle is simple, every time that the view of the map changes we recover panoramio images available in the visible area, before display placed in a **Group** to be able to handle them more easily.

It uses [PanoramioView.Search](#) to search for images, It is launched in a separate Thread, we connect us on the **OnPanoramioSearch** event to retrieve our image list.

Using the class TPanoramioLayer

```
FPanoramioLayer := TPanoramioLayer.create;
...
// connect TECMap component for show Panoramio layer
FPanoramioLayer.Map := map;
// disconnect layer
FPanoramioLayer.Map := nil
```

Source of TPanoramioLayer

```
unit UPanoramioLayer;

interface
uses Windows, SysUtils, Classes, Graphics,
    ECMaps, UECMapUtil;

type

    TOnPanoramioLayerPhotoClick = procedure(sender: TObject;
const PhotoId : integer) of object;

    TPanoramioLayer = class
    private
        FECMap                : TECMap;

        FOldOnPanoramioSearch : TOnPanoramioSearch;
        FOldOnChangeMapBounds : TNotifyEvent;

        FOnPanoramio          : TNotifyEvent;
        FOnClick               : TOnPanoramioLayerPhotoClick;

        FChangedBounds,
        FInfowindow,
        FShow : boolean;

        FIdInfoWindow,
        FMaxPhoto : integer;

        procedure setMap(const ecMap:TECMap);

        procedure setShow(const value:boolean);

        procedure AssignEvents;
        procedure RestoreEvents;
```



```

    procedure doOnPanoramioSearch(sender: TObject; const First,Last:
Integer);
    procedure doOnChangeMapbounds(sender: TObject);

    procedure doOnMarkerClick(sender: TObject; const Index
: integer;
    const dLatitude, dLongitude: double);

public
    constructor Create ;
    destructor Destroy ; override;

    procedure Clear;

    property Map          : TECMap          read FECMap
write setMap;

    property MaxPhoto     : integer         read FMaxPhoto write
FMaxPhoto;

    property Show         : boolean read FShow      write
setShow;
    property Infowindow   : boolean read FInfowindow write
FInfowindow;

    property OnPanoramio : TNotifyEvent read FOnPanoramio
write FOnPanoramio;
    property OnClick      : TOnPanoramioLayerPhotoClick read
FOnClick write FOnClick;

end;

implementation

constructor TPanoramioLayer.Create;
begin

    FMaxPhoto     := 300;
    FInfowindow   := true;

    inherited;
end;

destructor TPanoramioLayer.Destroy ;
begin

    Clear;

    Map := nil;

    inherited;

```

```
end;
```

```
procedure TPanoramioLayer.AssignEvents;  
begin
```

```
    if not assigned(Map) then exit;
```

```
        // save the events of Origins  
FoldOnPanoramioSearch := Map.OnPanoramioSearch;  
FoldOnChangeMapBounds := Map.OnChangeMapBounds;
```

```
        // connect to the events that interest us  
Map.OnPanoramioSearch := doOnPanoramioSearch;  
Map.OnChangeMapBounds := doOnChangeMapbounds;
```

```
end;
```

```
        // Reassign the events of origins  
procedure TPanoramioLayer.RestoreEvents;  
begin
```

```
    if not assigned(Map) then exit;
```

```
    Map.InfoWindows.delete(FIdInfoWindow);
```

```
end;
```

```
procedure TPanoramioLayer.setMap(const ecMap:TECMap);  
begin
```

```
    Clear;
```

```
    RestoreEvents;
```

```
    FECMap := ecMap;
```

```
    FIdInfoWindow := -1;
```

```
    FChangedBounds := false;
```

```
    AssignEvents;
```

```

    if assigned(FECMap) then Show := true;

end;

procedure TPanoramioLayer.Clear;
begin

    if not assigned(Map) then exit;

    Map.Groups['panoramio'].Delete;

end;

procedure TPanoramioLayer.setShow (const value:boolean);
begin

    // delete old images
    Clear;

    if value then
    begin

        Map.PanoramioView.LatSW := Map.SouthWestLatitude;
        Map.PanoramioView.LatNE := Map.NorthEastLatitude;
        Map.PanoramioView.LngSW := Map.SouthWestLongitude;
        Map.PanoramioView.LngNE := Map.NorthEastLongitude;

        // search for images, called doOnPanoramioSearch when
        they are found
        Map.PanoramioView.Search;

    end;

    FShow := value;

end;

    // triggered by Show := true
procedure
    TPanoramioLayer.doOnPanoramioSearch(sender: TObject;
const First,Last: Integer);
var i,j:integer;
    marker : TECMapMarker;
    Lat,Lng : double;
    url      : string;
begin

```

```

    for i := first to Last - 1 do
    begin
        Lat := Map.PanoramioView.photo_lat[i];
        Lng := Map.PanoramioView.photo_lng[i];

        url
:= replace_str(Map.PanoramioView.photo_file_url[i],
'/medium/', '/mini_square/');

        j := Map.AddMarker(lat,lng);

        marker := Map.Markers[j];
        marker.Draggable := false;
        marker.Icon := url;
        marker.Group := Map.Groups['panoramio'];
        marker.Tag := i;
        marker.OnMarkerClick := doOnMarkerClick;
    end;

Map.Groups['panoramio'].show;

if assigned(FOnPanoramio) then
    FOnPanoramio(self);

if (Map.PanoramioView.HasMore) and (Map.Groups[
'panoramio'].count<MaxPhoto) then
begin
    Map.PanoramioView.NextSearch;
    exit;
end;

if assigned(FOldOnChangeMapBounds) and FChangedBounds
then
    FOldOnChangeMapBounds(self);

    FChangedBounds := false;

end;

procedure
TPanoramioLayer.doOnChangeMapbounds(sender: TObject);
begin
    if FChangedBounds then exit;

    FChangedBounds := true;

    Show := true;

```

```

end;

procedure
  TPanoramioLayer.doOnMarkerClick(sender: Tobject; const
Index: integer;
    const dLatitude, dLongitude: double);
var i:integer;
    s : string;
begin

    if not assigned(Map.Markers[Index]) then exit;

    if Infowindow then
begin

    i := Map.Markers[Index].tag;

    s := '<h3>'+Map.PanoramioView.photo_title[i]+'</h3>'+
        Map.PanoramioView.HtmlPhoto(i,pimMedium)+
        Map.PanoramioView.HtmlCopyright(i) ;

    if FidInfoWindow< 0 then
        FidInfoWindow := Map.InfoWindows.add(s)

    else

        Map.InfoWindows[FidInfoWindow].Content := s;

        // for Bing maps
        Map.InfoWindows[FidInfoWindow].SetOptions('width :550,
height :450,zIndex:500');

        Map.InfoWindows[FidInfoWindow].SetPosition(dLatitude, dLongitude);

        Map.InfoWindows[FidInfoWindow].Open := true;

    end;

    if assigned(FOnClick) then

        FOnClick(self,Map.Markers[Index].tag);

    end;

```

end.

Let's see how to embed a mini synchronized map on the main map

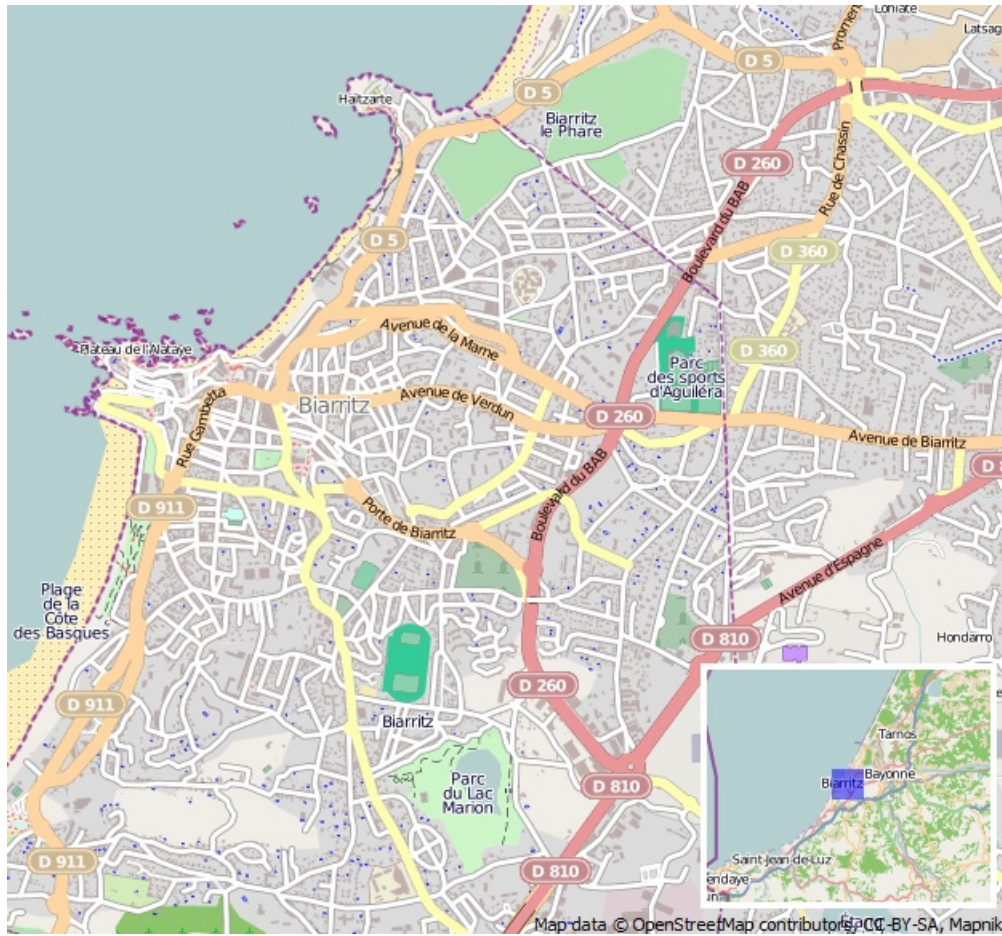


Fig. 173 A minimap on a component **TECNativeMap**

Design

Place a **TPanel** on your **TECNativeMap** component, attach the **color** to **clWhite**, **BevelInner** and **BevelOuter** properties to **false** and **BorderWidth** to **4**

Déposez sur le **TPanel** un composant **TECNativeMap** avec ses propriétés **Align** à **alClient**

Code

The mini map should be centered on the main Map with a zoom 5 notches below, and we will draw a rectangular box at the sight of the main card.

```

procedure TForm1.UpdateMiniMap;
begin

    // if the call comes from the direct change of the
    minimap, you must leave otherwise we go into an infinite loop
    if minimap.tag = 1 then exit;

    // indicates that map has initiated the movement
    map.tag := 1;

    if not assigned(FMiniPolygone) then
    begin
        minimap.ShowCopyrightTile := false;
        minimap.shapes.Polygons.add(0,0) ;
        FMinipolygone := minimap.shapes.Polygons[0];
        FMinipolygone.ShowText := false;
        FMinipolygone.fillColor := clBlue;
        FMinipolygone.hoverColor := clBlue;
        FMinipolygone.Opacity := 50;

        minimap.OnChangeMapZoom := minimapChangeMapZoom;
        minimap.OnMapMove := minimapMapMove;
    end;

    minimap.setCenter(Map.latitude,Map.longitude);
    minimap.zoom := Map.zoom - 5;

    FMinipolygone.SetPath([
        Map.SouthWestLatitude, Map.SouthWestLongitude,

        Map.SouthWestLatitude, Map.NorthEastLongitude,

        Map.NorthEastLatitude, Map.NorthEastLongitude,

        Map.NorthEastLatitude, Map.SouthWestLongitude,

        Map.SouthWestLatitude, Map.SouthWestLongitude
    ]);

```



```
map.tag := 0;
```

```
end;
```

You will need to call **UpdateMiniMap** in evenenemts **OnMapMove** and **OnChangeMapBounds** of the main card.

You should also respond to events **OnMapMove** and **OnChangeMapBounds** to adjust the view of the main map when it directly operates the mini map, without forgetting to adjust the position of the minimap in the event **OnResize**

```
unit UMainNativeMiniMap;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes,
  Graphics, Controls, Forms,
  Dialogs, uecNativeMapControl,
  ExtCtrls, StdCtrls, uecNativeShape;

type
  TForm1 = class(TForm)
    Panell: TPanel;
    Map: TECNativeMap;
    plus: TButton;
    Button1: TButton;
    pnMiniMap: TPanel;
    minimap: TECNativeMap;
    procedure FormCreate(Sender: TObject);

    procedure MapMapMove(sender: TObject; const Lat,
      Lng: Double);
    procedure MapResize(Sender: TObject);
    procedure MapChangeMapBounds(Sender: TObject);
    procedure minimapMapMove(sender: TObject; const Lat,
      Lng: Double);
    procedure minimapChangeMapZoom(Sender: TObject);
  private
    { Déclarations privées }
  end;
end;
```

```

    procedure UpdateMiniMap;
    var FMiniPolygone : TECShapePolygone;
    public
        { Déclarations publiques }
    end;

var
    Form1: TForm1;

implementation

    {$R *.dfm}

    procedure TForm1.FormCreate(Sender: TObject);
    begin
        Map.LocalCache := ExtractFilePath(application.exename) +
        'cache';
    end;

    procedure TForm1.MapChangeMapBounds(Sender: TObject);
    begin
        UpdateMiniMap;
    end;

    procedure TForm1.MapMapMove(sender: TObject; const Lat,
    Lng: Double);
    begin
        UpdateMiniMap;
    end;

    procedure TForm1.MapResize(Sender: TObject);
    begin
        pnMiniMap.Top := Map.Height - pnMiniMap.Height - 16;
        pnMiniMap.Left := Map.Width - pnMiniMap.Width - 8;
    end;

    procedure TForm1.minimapChangeMapZoom(Sender: TObject);
    begin
        // it deals only with direct access to the minimap
        if map.tag=1 then exit;

        // indicates that minimap has initiated the movement
        minimap.tag := 1;
        Map.zoom := minimap.zoom + 5;
        minimap.tag := 0;
    end;

    procedure TForm1.minimapMapMove(sender: TObject; const Lat,
    Lng: Double);
    begin
        // it deals only with direct access to the minimap

```

```

if map.tag=1 then exit;

    // indicates that minimap has initiated the movement
    minimap.tag := 1;
    Map.setCenter(minimap.latitude,minimap.longitude);
    minimap.tag := 0;
end;

procedure TForm1.UpdateMiniMap;
begin

    // if the call comes from the direct change of the
    minimap, you must leave otherwise we go into an infinite loop
    if minimap.tag = 1 then exit;

    // indicates that map has initiated the movement
    map.tag := 1;

    // if the first call, creation of the polygon indicates
    the main view
    if not assigned(FMiniPolygone) then
    begin
        minimap.ShowCopyrightTile := false;
        minimap.shapes.Polygones.add(0,0) ;
        FMinipolygone := minimap.shapes.Polygones[0];
        FMinipolygone.ShowText := false;
        FMinipolygone.fillColor := clBlue;
        FMinipolygone.hoverColor := clBlue;
        FMinipolygone.Opacity := 50;
    end;

    minimap.setCenter(map1.latitude,map1.longitude);
    minimap.zoom := map1.zoom - 5;

    // update polygon with boundingbox main map
    FMinipolygone.SetPath([
        map1.SouthWestLatitude, map1.SouthWestLongitude,

                                map1.SouthWestLatitude, map1.NorthEastLongitude,

                                map1.NorthEastLatitude, map1.NorthEastLongitude,

                                map1.NorthEastLatitude, map1.SouthWestLongitude,

                                map1.SouthWestLatitude, map1.SouthWestLongitude
        ]);

    map.tag := 0;

```

```
end;  
  
end.
```


INDEX

{

{{bbox}}	61
{{bbox}}	1
{{center}}	61
{{center}}	1

*

3D view	1
3d	383
45 °	1

A

Aborted	61
Abort	61
addBounds	1
AddByAdr	1
Add	1
AddInfoWindow	260
Add	122
AddLine	260
AddMarker	84
Add	84
AddMarker	260
add	1
Add	68
Add	53
AddPOI	260

AddPolygone	260
AddPolylineFromRoute	1
Addresses	1
Address	1
Address	143
AddRouteByAddress	1
Add	1
AddView	1
AdminLevelFromZoom	1
AdrControl	143
AdrCss	143
Adress	84
Adress	53
Adress	405
AdrPosition	143
AlertDistance	1
Altitude	1
Altitude	84
AltitudeMode	1
Altitude	1
Altitude	1
amAbsolute	1
amClampToGround	1
amClampToSea	1
Amenity	61
Amenity	1
amRelativeToGround	1
amRelativeToSea	1
Anchor	1
ancragePosition	174
android	377
android	174
angle	260
Animation	328
Animation	84
apiCloudMade	1
apiGoogle	1
apiLeaflet	1
apiOpenMapQuest	1
ApiVersion	1
ApiVersion	1
ArcGis	1
Archive	1

area	1
Area	68
Atmosphere	1
Author	164
AutomaticMapMovement	260
AutomaticUpdate	122
Available	143
AvoidHighWays	170
avoidHighways	1
AvoidTolls	170
avoidTolls	1

B

BeginUpdate	1
BeginUpdate	68
Bezier	303
bicyclinglayer	122
bicycling	122
Bing Maps	1
bing maps	1
Bing	1
BingKey	1
BingKey	174
bing	1
BorderColor	174
BorderColor	303
BorderSize	174
BorderSize	303
BOUNCE	84
Boundary	1
Bounds	61
bounds	1
BringToFront	260
ByLatLng	84
ByName	84

C

cache	1
-------------	---

cache	1
Camera	1
CanceledSearchImageClose	122
CanceledSearchSequence	122
CancelSearchImageClose	122
CancelSearchSequence	122
card type	1
cards	1
CEF1	9
CEF3	9
Cells	170
CenterOnMap	260
Center	68
cesium	383
chrome	1
chromium	1
chromium	9
circle	68
Circles	68
Clause	122
Clear	1
clear	1
ClearGuide	260
Clear	61
Clear	122
Clear	84
ClearMarker	260
clear	1
Clear	68
Clear	53
Clear	1
Clickable	84
Clickable	164
ClientId	122
CloseBtnVisible	143
CloseButton	323
CloudMAde and Google Maps	1
cloudmade	1
CloudMade	1
cloudmade	1
cloudmade	9
CloudMadeKey	1
CloudMade	1

CloudMade	174
Clusterable	260
ClusterManager	260
Cluster	84
Cluster	260
Color.	1
Color	1
color	1
Column	122
component	1
Connection	122
connection	1
Contains	1
ContainsLatLng	1
ContentCenter	323
Content	1
Content	105
CopyImage	84
Copyright	1
CountDetail	53
Count	170
CountKey	61
Count	122
Count	84
count	1
Count	68
Count	53
CountResult	53
Count	1
cpBottomCenter	1
cpBottomLeft	1
cpBottomRight	1
cpLeftBottom	1
cpLeftCenter	1
cpLeftTop	1
cpRightBottom	1
cpRightCenter	1
cpRightTop	1
cpTopCenter	1
cpTopLeft	1
cpTopRight	1
css	105
CurrentTimeTour	1

Cursor	84
Cursor	260
Cursors	84
cyclemap	122
czml	383

D

Delete	1
delete	1
Delete	122
Delete	84
delete	1
Delete	68
Delete	53
Delete	1
Delphi	1
Delphi	1
delphi	1
delphi	53
Delphi	1
Demo3D	9
DemoLayer	122
DemoLocalise	9
DEMOmobile	9
DemoOverlays	9
DemoRoute	9
Destination	170
Destinations	170
Direction	1
Display a Panoramio Layer in Bing maps	1
Distance	170
DistanceFrom	1
DistanceFrom	1
DistanceMatrix	170
Distance	1
Distance	68
DistanceRouteByAdress	1
DistanceRouteFrom	1
Distance	1
DistanceText	170

DistanceTo	260
Download	1
Draggable	105
Draggable	84
Draggable	1
Dragging	84
DragRect	174
Drag	260
DragZoom	174
Draw	260
DROP	84
drSelect	174
drZoom	174
Duration	170
Duration	1
Duration	1
DurationText	170

E

ecmap	1
EditMode	68
EditOnClick	1
EditOverlay	68
EditRoute	1
EnableLayer	1
EnableTouchRotation	174
EnableUIControl	1
Encoded	303
EncodePrecision	303
EndAdress	1
EndLatitude	1
endLongitude	1
EndMarkerFilename	1
EndUpdate	1
EndUpdate	68
EngineExcute	1
EngineExecute	1
EngineKey	1
EngineName	1
Engine	1

EngineUrl	1
EngineValidUrl	1
ErrorDistance	1
ExcludelfKeyExist	61
ExcludeKeyValue	61
ExecutionDistance	1
ExitTour	1

F

field of view	288
FileFormat	61
FilenameEndEditLine	303
FilenamePointEditLine	303
FilenameStartEditLine	303
fillColor	68
FillOpacity	68
FilterBounds	61
FilterNode	61
FilterPrimitive	61
FilterRelation	61
FilterWay	61
FindImageClose	122
Find	61
FindShapeByArea	260
FindShapeByFilter	260
FindShapeByKMDistance	260
FindToShapes	61
firemonkey	1
fitBounds	1
fitBounds	84
fitBounds	1
fitBounds	1
fitbounds	260
Flat	84
FlyToSpeed	1
FontSize	1
fovradius	288
fov	288
FreeHand	260
FromXYToLatLngAlt	1

FusionTablesLayer	122
fusiontables	122

G

geodesic	303
Geofence	1
geohash	1
GeoJSON	1
GeoLocationFromLatLng	1
geolocation	9
GeoLocation	1
GeoXml	122
GeoXml	68
GetAdressFromLatLng	1
getAdress	68
GetAltitudeAtLatLng	1
getAltitudes	68
GetAsyncRoutePathByAdress	1
GetASyncRoutePathFrom	1
getDetails	53
getGroundAltitude	1
getKey	61
getLatLngFromMeter	1
GetRoutePathByAdress	1
GetRoutePathFrom	1
Google maps	1
Google Maps	1
google maps	1
google maps	9
Google maps	1
Google	1
GoogleMapsKey	1
gpx	1
Grid	1
GroundOverlays	68
GroundOverlay	288
Group	1
GroupFilter	174

H

haversine	1
Heading	1
HeadingFrom	1
Heading	84
Heading	1
Heading	1
Heading	143
Heatmap	122
heatmap	122
Height	68
Here	174
hide	1
hint	260
HoverBorderColor	303
htmlInfoWindow	122
html	105

I

IconAnchor	84
Icon	84
IconOrigin	84
IconSize	84
Id	105
ID	122
Images	122
Import/Export	61
Index	1
Index	84
Index	1
IndexOf	1
IndexOfLatLng	1
IndexOf	84
IndexOf	1
IndexOf	53
infowindow	1
InfoWindowDescription	323
InfoWindow	1
InfoWindow	9

InfoWindow	84
InfoWindow	68
infowindow	260
InfoWindow	323
Insert	68
instructions	1
iOS	381
ios	174
Is	68
isTrackLineActive	260
ItemDetail	53
Item	170

J

junction	122
----------------	-----

K

Keys	61
kml	1
KML	122
KML	68
KMZ	122
KMZ	68

L

Label	105
Label	122
Labels	68
Latitude	1
Latitude	1
Latitude	84
Latitude	53
Latitude	1
Latitude	143
Latitude	405

layer	122
leaflet	1
Leaflet	1
leaflet	1
leaflet	1
leaflet	1
Leaflet	1
licence	1
line	68
link	323
LinkVisible	143
LoadAndZoomToCZML	383
loadczml	383
Loaded	1
LoadFromFile	1
LoadFromOSMStream	61
LoadFromOSMStream	260
LoadFromOSMString	61
LoadImage	122
LoadKml	1
localarchive	1
LocalCache	122
LocalCache	1
LocalCache	1
LocalCache	174
LocalPathImage	122
Location	1
LocationType	1
Longitude	1
Longitude	1
Longitude	84
Longitude	53
longitude	1
Longitude	143
Longitude	405
LookAt	1
ltBezier	303

M

mac osx	1
---------------	---

mac osx	174
magnet	260
map	1
Mapapa	1
MapAPI	1
maparchive	1
MapBox	1
MapBoxToken	174
map	1
mappilay	122
mapping	1
MapQuest	1
MapQuest	1
mapquest	1
MapQuest	1
MapQuest	174
maps	1
maps	9
maps	1
MapTypeControl	1
MapTypeControlPosition	1
MapTypeID	1
MapType	405
MapZen	1
marker	1
Marker	1
MarkerOptions	1
Markers	84
Markers	68
MaxDayInCache	122
maxdayincache	174
MaxWidth	1
MaxZoom	1
MaxZoom	1
mcLocalOrServer	122
mcOnlyLocal	122
mcOnlyMappilaryServer	122
Menu	1
MeterDistance	260
MinDistanceMeterForNotifyMove	260
MinZoom	1
MinZoom	1
Mobile	1

MobilesEnabled	1
MobilesTiming	1
Model	1
Models	1
MouseAltitude	1
MouseButton	260
MouseLatitude	1
MouseLatLng	174
MouseLongitude	1
MouseNavigationEnabled	1
MoveMarkerOnRouteToMeter	84
Move	1
mtHYBRID	1
mtROADMAP	1
mtSATELLITE	1
mtTERRAIN	1
MultiView	1

N

NameDetail	53
Name	84
NameResult	53
NavigationControl	1
Navigation	1
NavigationControl	143
NavigationControlVisibility	1
NavigationPosition	1
NavigationPosition	143
Navigation	1
NavigationStyle	143
ncvAuto	1
ncvHide	1
ncvShow	1
NextInstructionInKm	1
NextInstructionPosition	1
NextInstruction	1
NextManeuver	1
node	122
Nodes	61
Nominatim	403

NorthEastLatitude	1
NorthEastLatitude	1
NorthEastLongitude	1
NorthEastLongitude	1

O

Observer	260
Observer	174
offline mode	174
offline	1
OnAddRoute	1
OnAddShapeToView	1
OnAdress	403
OnAfterChangeMapApi	1
OnAfterChangeToTxt	1
OnAfterInstruction	1
OnAfterReload	1
OnAlert	1
OnAnimation	328
OnArrival	1
OnBeforeChangeMapApi	1
OnBeforeChangeToTxt	1
onbeforedraw	288
OnBeforeLoad	1
OnBeforeReload	1
OnBeforeUrl	174
OnBeforeUrl	323
OnChangeMapBounds	1
OnChangeMapBounds	1
OnChangeMapTypeld	1
OnChangeMapZoom	1
OnChangeRoute	1
OnClick	122
OnCloseInfoWindow	1
OnCloseInfoWindow	323
OnConnectRoute	1
OnCreateShapeLinePoint	303
OnData	61
OnData	1
OnDisconnectRoute	1

OnDrawRoute	1
OnEarthViewChange	1
OnEarthViewClickPlacemark	1
OnEarthViewFrameEnd	1
OnEarthViewInit	1
OnEarthViewShow	1
OnEarthViewUpdateModels	1
OnEarthViewUpdatePlacemarks	1
OnEditOverlayPointClick	68
OnEditOverlayPointDragEnd	68
OnEditOverlayPointRClick	68
OnEndAnimation	328
OnEndMobile	1
OnEnterGeofence	1
OnErrorRoute	1
OnError	1
OnImages	122
OnInstruction	1
OnJavascriptError	1
OnKmlLayerClick	122
OnKmlLayerClick	68
OnLeaveGeofence	1
ONLink	105
OnLoaded	61
onload	260
OnLoadShapes	61
OnLoad	1
OnlyIfKeyExist	61
OnlyKeyValue	61
onlylocal	174
OnMapDragEnd	1
OnMapDrag	1
OnMapDragStart	1
OnMapMouseMove	1
OnMapMove	84
OnMapMove	1
OnMapRightClick	1
OnMarkerAdress	84
OnMarkerClick	84
OnMarkerDbIClick	84
OnMarkerDragEnd	84
OnMarkerDrag	84
OnMarkerDragStart	84

OnMarkerMove	84
OnMarkerMove	1
OnMarkerRightClick	84
OnMouseOutCluster	260
OnMouseOverCluster	260
OnOverlayChange	68
OnOverlayClick	68
OnOverlayDbiClick	68
OnOverlayMouseDown	84
OnOverlayMouseDown	68
OnOverlayMouseOut	68
OnOverlayMouseOver	68
OnOverlayMouseUp	84
OnOverlayMouseUp	68
OnOverlayMove	105
OnOverlayMove	68
OnOverlayPathChange	68
OnOverlayRightClick	68
OnPanoramioLayerClick	164
OnPlacesDetail	53
OnPlacesSearch	53
OnRead	61
OnRouteChange	1
OnRouteError	1
OnRoutePath	1
OnSelectShape	174
OnSelect	403
OnSelect	303
OnSequenceColor	122
OnSequences	122
OnShapeClick	260
OnShapeDbiClick	260
OnShapeDragEnd	260
OnShapeDrag	260
OnShapeMouseDown	260
OnShapeMouseOut	260
OnShapeMouseOver	260
OnShapeMouseup	260
OnShapeMove	260
OnShapeRightClick	260
OnSnap	260
OnStreetViewAvailable	143
OnStreetViewPosition	143

OnStreetViewPOV	143
OnStreetViewVisible	143
Opacity	1
OpenClycleMap	174
Open	1
openlocationcode	1
openmapquest	1
openmapquest	1
OpenMapQuest	1
OpenStreetMap	53
OpenStreetMap	174
openweathermap	303

Ö

ÖPNV	174
------------	-----

O

OptimizeWaypoints	1
Origin	170
Origins	170
OSM XML	61
OSMFileToOLT	61
OSMViewer	1
Our	174
ovCircle	68
OVER_QUERY_LIMIT	1
overlay	68
Overlays	1
overlay	174
OverlayType	68
OverPassUrl	1
OverSizeForRotation	174
OverviewMap	1
ovGlobe	68
ovGroundOverlay	68
ovKml	68
ovLabel	68
ovLine	1

ovLine	68
ovMap	68
ovMarker	68
ovPolygone	1
ovPolygone	68
ovRectangle	68
ovRoute	1
ovRoute	68
OwnsGraphic	288

P

Panoramio	1
PanoramioLayer	122
Panoramio	122
Panoramio	1
PanoramioView	164
PanToBounds	1
PanToBounds	68
PanToBounds	1
PanTo	84
Panto	1
Params	405
Path	68
PauseTour	1
PhotoID	164
Pitch	143
place	1
PlacemarkCamera	1
PlacemarkCount	1
Placemark	1
PlacemarkLatLngAlt	1
PlacemarkLookAt	1
Place	53
PlayTour	1
PluginVersion	1
Point	68
POIUnit	298
PolygoneSelection	260
polygon	68
Polygons	68

Position	1
propertyValue	260
psUserStyle	303

Q

Query	61
Query	1

R

RADIUS	68
RADIUS	53
Range	1
RawDetail	53
RawResult	53
ReadKey	61
ReadValuesForKey	61
Ready	1
Ready	1
rectangle	68
Rectangles	68
ReDraw	143
Relationships	61
Relations	61
ReleaseAllView	1
ReleaseView	1
ReLoad	61
Reload	1
RemoveAllOverlayTiles	174
Remove	1
RemoveGuide	260
Remove	122
RemoveMarker	260
Request	1
ResetTour	1
Result	53
Results	53
Reverse	303
Road	1

roads	9
Roads	68
Roads	1
Roll	1
RouteDrawingTime	1
Route	1
RouteType	1
routeType	1
routing engine	1

S

SaveToFile	1
saveToFile	61
SaveToKMLFile	1
ScaleControl	1
scale	1
ScaleLegend	1
ScalePosition	1
Scale	1
ScreenShot	61
ScreenShot	1
ScreenShots	61
SearchImageClose	122
Search	61
Searching	53
Search	1
Search	53
Selected	174
SelectionFilter	260
SelectionFrom	303
Selection	260
SelectionTo	303
SendToBack	260
SensStartEnd	1
Sequences	122
Server	1
SetCenter	1
setCustomDash	303
SetHitBox	288
setImage	84

setImageShadow	84
setLatLngAlt	1
setPosition	1
setPosition	105
setPosition	84
SetPosition	143
setShape	84
Shadow	84
Shape	84
Shapes	260
show	1
ShowOnMap	84
Show	260
SnapDistance	260
snapdrag	260
SnapGuide	260
snap	260
SnapShape	260
SouthWestLatitude	1
SouthWestLatitude	1
SouthWestLongitude	1
SouthWestLongitude	1
Speed	1
SQL	122
SSL	1
StartAdress	1
StartLatitude	1
StartLongitude	1
StartMarkerFilename	1
StatusBar	1
Status	170
Status	53
Step	1
StopLoadOverlay	1
StreamPercent	1
StreetNames	61
StreetNames	1
Streets	61
Streets	1
StreetView	143
StreetView	1
styleicon	288
Style	122

style	260
styles	260
styles	1
style	1
style	1
Sun	1
SuppressInfoWindows	164
svg	288
symbol	84

T

Tag	84
Tag	164
TECAAnimationAutoHide	328
TECAAnimationFadePoi	328
TECAAnimationMarkerFilename	328
TECAAnimationMarkerZoomFilename	328
TECAAnimationShapeColor	328
TECCesiumMap	383
TECCesiumShapeInfoWindow	383
TECCesiumShapeLine	383
TECCesiumShapeMarker	383
TECCesiumShapeModel	383
TECCesiumShapePOI	383
TECCesiumShapePolygone	383
TECDistanceMatrixItem	170
TECDownLoadTiles	1
TECDownLoadTiles	174
TECesiumMap	383
TECMapAdressEdit	403
TECMapEarthModel	1
TECMapEarthModels	1
TECMap	1
TECMapFusionTablesLayer	122
TECMapInfoWindow	1
TECMapInfoWindows	1
TECMapMarker	84
TECMapMarkers	84
TECMapOverlay	68
TECMapPanoramioLayer	164

TECMappilaryLayer	122
TECMapPlaces	53
TECMapRoutePathPoint	1
TECMapRoute	1
TECMapRoutes	1
TECMapStreetView	143
TECNativeMap	1
TECPlaceResults	53
TECShapeInfoWindow	323
TECShapeLine	303
TECShapePoi	298
TECShapePolygone	320
TECShape	260
TECStaticMap	405
TerrainExaggeration	1
TextSearch	53
TGeoCoderReponses	1
The BorderSize property	174
Thumb1024	122
Thumb2048	122
Thumb320	122
Thumb640	122
Thumb	122
tickets	53
TileServer	174
tiles	1
Tilt	1
tilt	1
Timestamp	61
Title	84
TLatLngList	68
TListGeoCoderReponses	1
TMappilaryAPI	122
tmBicycling	1
tmDriving	170
tmDriving	1
tmWalking	170
tmWalking	1
TNativeMapServer	1
ToKml	1
toKml	1
ToKml	1
ToKml	122

ToKml	84
ToKml	68
ToKml	1
TomTom	174
TopGroupZIndex	260
TopZIndex	260
ToShapes	61
TOSMFile	61
ToTxt	1
toTxt	1
ToTxt	1
ToTxt	105
ToTxt	122
ToTxt	84
ToTxt	1
ToTxt	68
ToTxt	164
ToTxt	1
ToTxt	143
ToTxtType	1
tour	1
TOverlayType	1
TPopupMenu	1
Tracking	1
TrackLine	260
TrafficLayer	122
traffic	122
TravelMode	170
TravelMode	1
TRoutingEngine	1
tsMapBoxSatellite	174
tsMapBoxStreetsBasic	174
tsMapBoxStreetsSatellite	174
tsMapBoxStreets	174
turn by turn	1
TurnByTurn	1
type of card	1

U

UEC_OSM_STYLESHEET	1
--------------------------	---

Undo	1
UnitSystem	170
UnSelectedAll	174
Update	170
updateOptions	1
Update	1
UrlImage	122
URL	174
UseInfoWindowDescription	323
useOSM	53
UserId	164
usMetric	170
UTFGrid	122

V

Values	61
vector icon	84
vector tiles	1
vector	260
vector	1
ViewCount	1
view	1
Views	1
Visible	1
Visible	105
Visible	84
Visible	164
Visible	143

W

way	122
WayPoints	1
Ways	61
WeatherLayer	122
weather	303
WebServer	383
WebSocket	1
Weight	1

Weight	1
Width	68

X

XAnchor	288
xapi	122
XAPI	53
XMargin	174

Y

YAnchor	288
Yandex	174
YMargin	174

Z

ZIndex	84
ZIndex	1
zip	1
ZoomAround	174
ZoomControl	1
zoom	1
ZoomPosition	1
Zoom	1
ZoomScaleFactorAround	174
zoomscalefactor	174
Zoom	143
Zoom	405