

Table des matières

ECMap	1
Achat - Contact	4
Licence d'utilisation	7
Installation	10
TECMap	21
Positionnement	27
Type de carte	34
Contrôles	47
Localisation	50
Places	58
Import/Export	66
Overlays	72
Markers	87
Mobiles	97
InfoWindows	103
Labels	108
Routes	111
Layers	125
Pois	136
Groupes	142
StreetView	147
EarthView	153
Panoramio	168
DistanceMatrix	174
TECNativeMap	178
Mode hors-ligne	201
Tuiles vectorielles	206
Multi vue	217
Serveur de tuiles distant	219
Localisation	227

Import/Export	241
Haute résolution	258
Shapes	262
TECShapeMarker	285
TECShapePoi	294
TECShapeLine	299
TECShapePolygone	315
TECShapeInfoWindow	318
Animation	323
Routes	333
Layers	345
Android	370
iOS	375
Autres composants	376
TECCesiumMap	377
TECMapAdressEdit	397
TECStaticMap	399
TECVelib	401
Programmation	404
Création interactive d'une route	405
Afficher StreetView dans une InfoWindow	420
Afficher un Layer Panoramio dans toutes les cartes	423
Afficher une MiniMap	431
Table des illustrations	437

ECMap est une suite de composants pour Delphi, depuis la version 7, qui prend en charge les apis [Google Maps 3](#) , [Google Earth](#), [Bing Maps](#), [CloudMade](#) , [OpenMapQuest](#) et [Leaflet](#)



Fig. 1 Deux composants, 5 Apis, 3 Moteurs d'affichage !

TECMap IE & Chromium

Avec [TECMap](#) vous pouvez utiliser 2 moteurs d'affichage, soit Internet Explorer soit [Google Chromium](#) pour ce dernier il vous faudra d'abord installer le composant [Delphi Chromium Embedded](#)

Vous avez alors accès aux web apis Google maps et Earth, Bing Maps, CloudMade, OpenMapQuest et Leaflet.

Avec [TECCesiumMap](#) intégrez le [globe 3D Cesium](#) dans vos applications !



TECNativeMap 100% Delphi VCL et Firemonkey

TECNativeMap utilise un rendu **100% delphi, 0% WebBrowser & Javascript** donc plus rapide et plus léger en mémoire

Profitez entre autre du [mode hors-ligne](#), de [mappilary](#), des [vues multiples](#), de [captures bitmap](#) des zones même hors écran (maximum 8000x8000), [d'animations](#), d'[imports exports KML](#) etc...

Disponible en version VCL (D7 à 10.2 Tokyo) et Firemonkey (Xe3 à 10.2 Tokyo) (**Windows , Mac OS X , iOS et Android**)

[*Testez une démonstration pour Android*](#)


Vous allez pouvoir gérer de manière optimale vos cartes et vos données géographique en utilisant le langage que vous connaissez le mieux Delphi !

Les **sources des composants et des démonstrations sont livrées** avec une aide au format chm et PDF, vous pouvez directement consulter le manuel sur le site et vous faire une idée des capacités du composant en téléchargeant les [10 démonstrations](#).


Gratuit [*avec votre licence complète TECMap*](#)
[*les composants TECesiumMap, TECMapAdressEdit ,*](#)
[*TECStaticMap et TECVelib*](#)

Les [versions d'essai des composants TECMap et TECNativeMap](#) sont







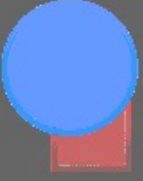
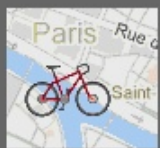






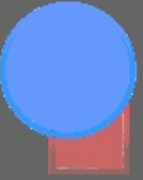

disponible sur [simple demande](#)




TECMap
IE & Chromium Browser
Multi Web Api



TECNativeMap
100% delphi, fast & light
0% WebBrowser %0 Javascript

 Earth3D	 Localise	 Routes	 Localise
 Routes	 Mobiles	 Pois	 Velib
 Overlays	 Matrix	 Layers	 MiniMap
 Layers	 AdressEdit		
 Native Pois	 Velib		



Double click on a point to delete, double click on a line for

GRATUIT - Téléchargez les démos !

Identification

ECMap et ce site sont édités par **ESCOT-SEP Christophe**, auto-entrepreneur immatriculé à Saint-Lô sous le numéro **509 737 532** dont le siège social est **231 rue des fauvettes, 50000 SAINT-LÔ**

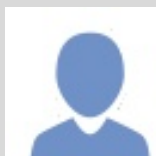
Documentation

Vous pouvez télécharger le contenu de ce site au format [PDF](#) et [CHM](#)

Achat

Vous pouvez payer directement en ligne depuis le site en cliquant sur le bouton **Acheter**, par chèque à mon nom ou par virement (faire la demande par email pour avoir mon RIB)

TECMap suite complète

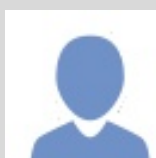


Licence **1 développeur**

500 euros (tva 0%)

La licence comprend les mises à jour gratuites, l'intégralité du code source des composants (TECMap, [TECNativeMap](#), [TECCesiumMap](#), TECMapAdressEdit, TECStaticMap, TECVelib) et des démonstrations, une assistance gratuite par mail pour l'utilisation du composant

TECNativeMap



1 développeur

220 euros (tva 0%)

Le tarif d'une licence
1 développeur
220 euros (tva 0%)

Le tarif d'une licence
multiple développeurs
700 euros (tva 0%)

La licence comprend les mises à jour gratuites, l'intégralité du code source des composants en version WebGL / Firemonkey (Windows, Android, iOS, etc.)

Prestations

Je peux prendre en charge l'intégration du composant dans votre projet, faire des modifications spécifiques.

• **250 euros** en télétravail

•
400 euros) par journée en déplacement (minimum 2 jours)

Envoyez-moi votre cahier des charges pour que je vous fasse un devis.

Nous contacter

infos@helbardweb.com NativeMap.


La licence de TECMap vous permet d'intégrer celui-ci dans vos applications et de les distribuer librement sans royalties, vous pouvez modifier les sources pour l'adapter à vos besoins.


Il vous est interdit de redistribuer le composant, source ou binaire, de réaliser un composant à partir de TECMap et de le distribuer, y compris gratuitement.


La licence d'utilisation du composant TECMap n'est pas une licence d'utilisation des apis et données cartographiques employées, celles-ci sont respectivement la propriété de Google, CloudMade, MapQuest, Microsoft et OpenMapStreet.


Avant toute utilisation vous devez **lire et accepter** leurs conditions d'utilisations

- [Conditions d'utilisation des apis Google Maps/Google Earth](#)
- [Conditions d'utilisation de l'api CloudMade](#)
- [Conditions d'utilisation de MapQuest Nominatim](#)
- [Condition d'utilisation de MapQuest Elevation](#)
- [Conditions d'utilisation de la carte OpenStreet MapQuest](#)
- [Conditions d'utilisation d'OpenMapStreet](#)
- [Conditions d'utilisation de Bing Maps](#)

MapQuest Open Elevation  est employé pour la recherche d'altitudes avec l'api [CloudMade](#) , [OpenMapQuest](#) et [Leaflet](#)

[MapQuest Nominatim](#)  est utilisé pour retrouver une adresse à partir de la latitude et longitude avec CloudMade , OpenMapQuest et Leaflet.

[MapQuest Xapi API Service](#)  est utilisé sous CloudMade , OpenMapQuest et Leaflet en équivalence de [Places](#) de Google.

La carte [MapQuest](#)  , les cartes [Mapnik](#), [Osmarender](#) et [CycleMap](#), peuvent-être utilisées avec les apis Google , CloudMade et Leaflet

Enregistrement

TECMap utilise une clef par défaut pour Bing, OpenMapquest, CloudMade et Leaflet, vous **devez obtenir** gratuitement la votre en vous enregistrant sur leur site.

[Obtenir une clef pour CloudMade / Leaflet](#)

[Obtenir une clef pour OpenMapQuest](#)

Entrez vos clefs dans les propriétés **GoogleMapsKey**, **MapQuestKey**, **CloudMadeKey** et **BingKey** de TECMap.

La clef pour Google Maps n'est pas obligatoire, l'api peut fonctionner sans.

Google Maps API Premier

Si vous avez souscrit à Google Maps API Premier vous avez obtenu un "*Client ID*" du style **gme-xxx**, vous devez renseigner la propriété **ClientID** de TECMap pour pouvoir l'utiliser, placez votre clef dans **GoogleMapsKey**

.

[contactez google pour savoir comment souscrire à ce service](#)

Installer la suite ECTMap

- Ouvrez simplement le fichier **ECTMap.dpk** présent dans le répertoire Source
- Faites un clic droit sur **ECTMap.bpl** dans le gestionnaire de projet et cliquez sur **Installer**
- Ajoutez le répertoire Source dans la **liste des bibliothèques - Win32** (Menu Outils - Options).

Vous pouvez utiliser les composants qui se retrouvent sur l'onglet **ECTMaps** de la palette des composants

Installer TECNativeMap

Répétez la procédure ci-dessus mais avec les fichiers **ECNativeMAP.dpk** (Vcl) et **FMX.ECNativeMAP.dpk** (Firemonkey)

Vous n'avez pas besoin d'installer ECNativeMAP.dpk avec la suite ECTMap car il est compris dedans, par contre vous devez installer FMX.ECNativeMAP.dpk

Activer Chromium

Pour pouvoir utiliser le navigateur Google Chromium à la place d'Internet Explorer vous devez tout d'abord installer le composant [Delphi](#)

Chromium Embedded CEF1 développé par [Henri Gourvest](#).

ECMap supporte aussi [Delphi Chromium Embedded CEF3](#)

Que ce soit la version CEF1 ou CEF3 prenez les dernières versions
présentent sur SVN

Puis modifiez le fichier **Delphi_Versions.inc** livré avec ECMap
et remplacez **{\$DEFINE CHROMIUM}** par **{\$DEFINE CHROMIUM}**

Si vous utilisez une version de Chromium CEF1 <= 275 vous
devez remplacer **{\$DEFINE CEF275_UP}** par **{\$DEFINE CEF275_UP}**

Pour utiliser CEF3 remplacez **{\$DEFINE CEF3}** par **{\$DEFINE CEF3}**

Vous pouvez maintenant installer le composant ECMap, pour basculer
le moteur d'affichage il faut modifier la propriété
TECMap.DisplayBrowser en lui attribuant les valeurs **dbIE** ou
dbChromium

Vous devrez distribuer les binaires de chromium avec votre exécutable,
ils sont dans le répertoire bin\win32 des sources de chromium.

Dans le .dpr de votre projet vous pouvez indiquer le chemin vers
ces binaires

```
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.

Ici vous aurez le sous-répertoire bin\win32 au même niveau que votre executable

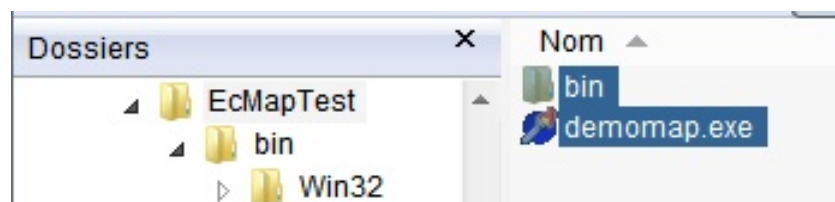


Fig. 4 Livraison d'un projet utilisant chromium

CEF3 permet d'utiliser plusieurs processus, par défaut ce n'est pas le cas, le mode processus unique n'est recommandé que pour le débogage pas pour la livraison.

Pour activer le mode multi-processus vous devez rajouter ces 2 lignes après `CefLibrary := ...` dans votre `.dpr`

```
CefSingleProcess := False;  
if not CefLoadLibDefault then Exit;
```

Démonstrations

Le composant est livré avec 8 programmes de démonstrations qui vous montreront comment l'exploiter.

DemoLocalise vous apprend à vous servir des fonctions de [geolocalisation](#), se rendre à un point à partir de son adresse, retrouver une adresse à partir de sa position géographique.

Vous pourrez rechercher un type de lieu avec l'utilisation de [Places](#)

Vous verrez aussi l'utilisation d'une [InfoWindow](#), la sélection de l'[api employée](#) (Google ou CloudMade) et du [type de carte](#) affichée.

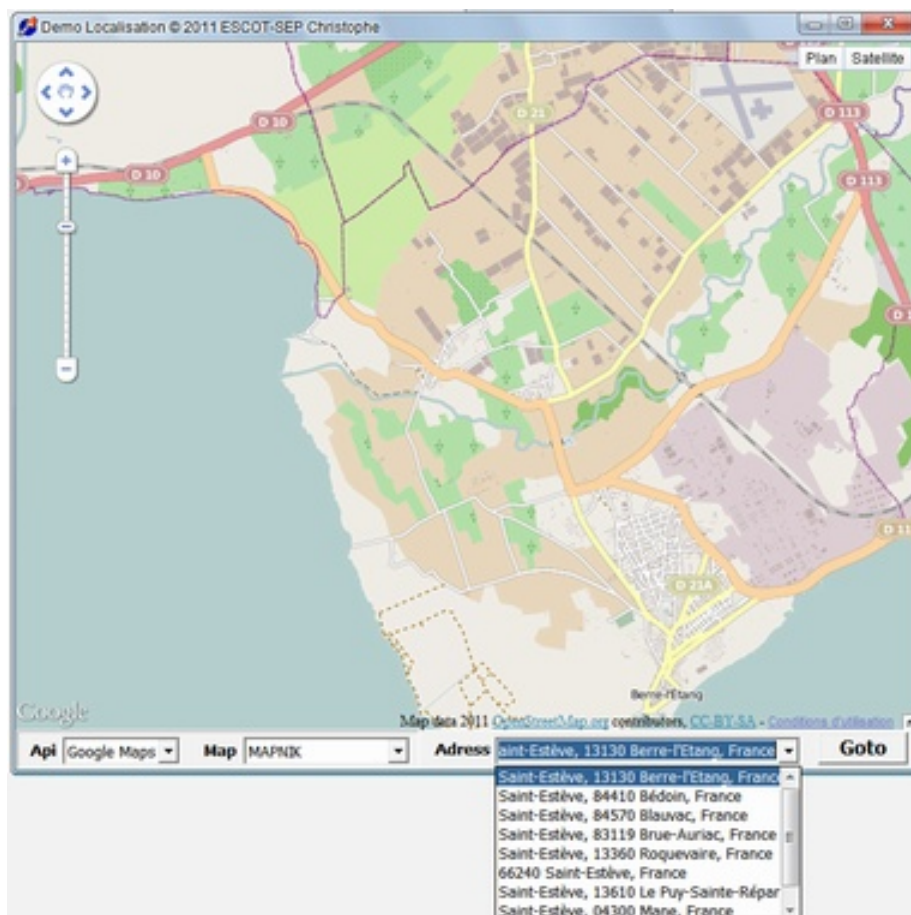


Fig. 5 DemoLocalise

DemoRoute est centré sur la gestion des routes, vous y verrez aussi comment obtenir l'altitude d'un ensemble de points

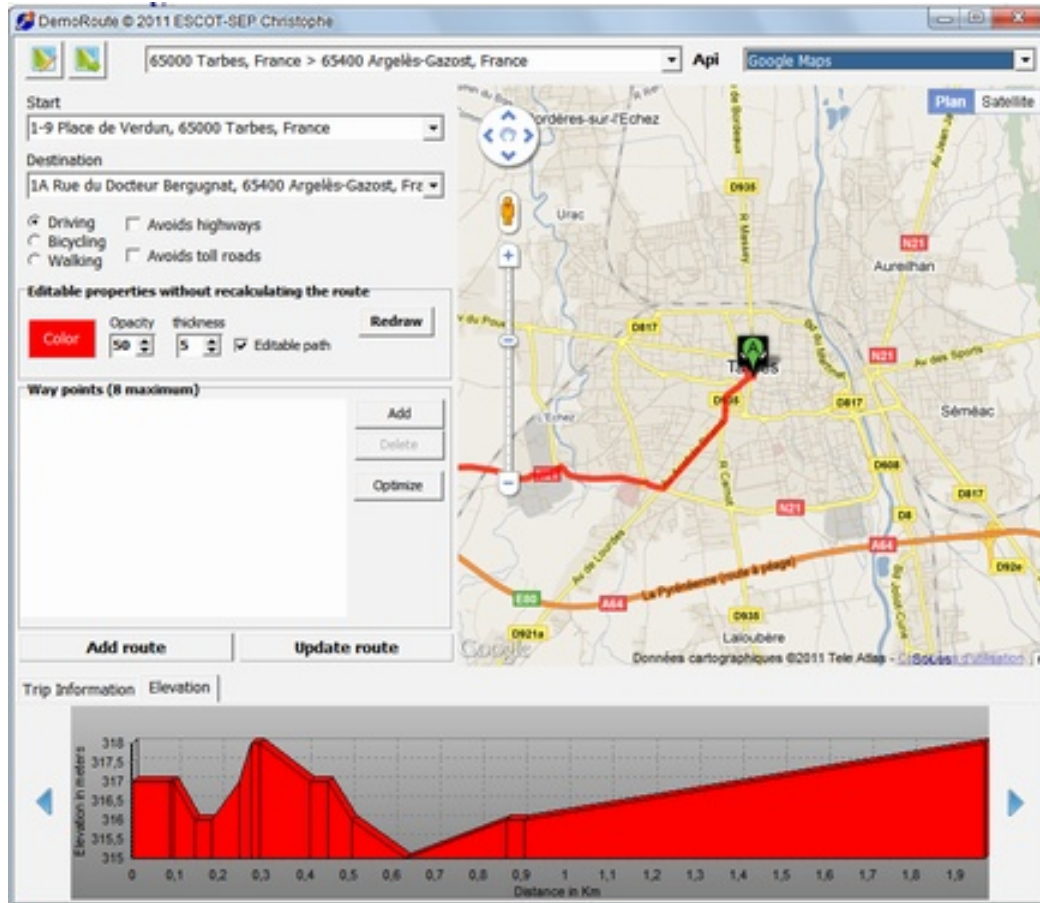


Fig. 6 DemoRoute

DemoMobile concerne la gestion automatique des mobiles sur un tracé couplé à un suivi **StreetView**

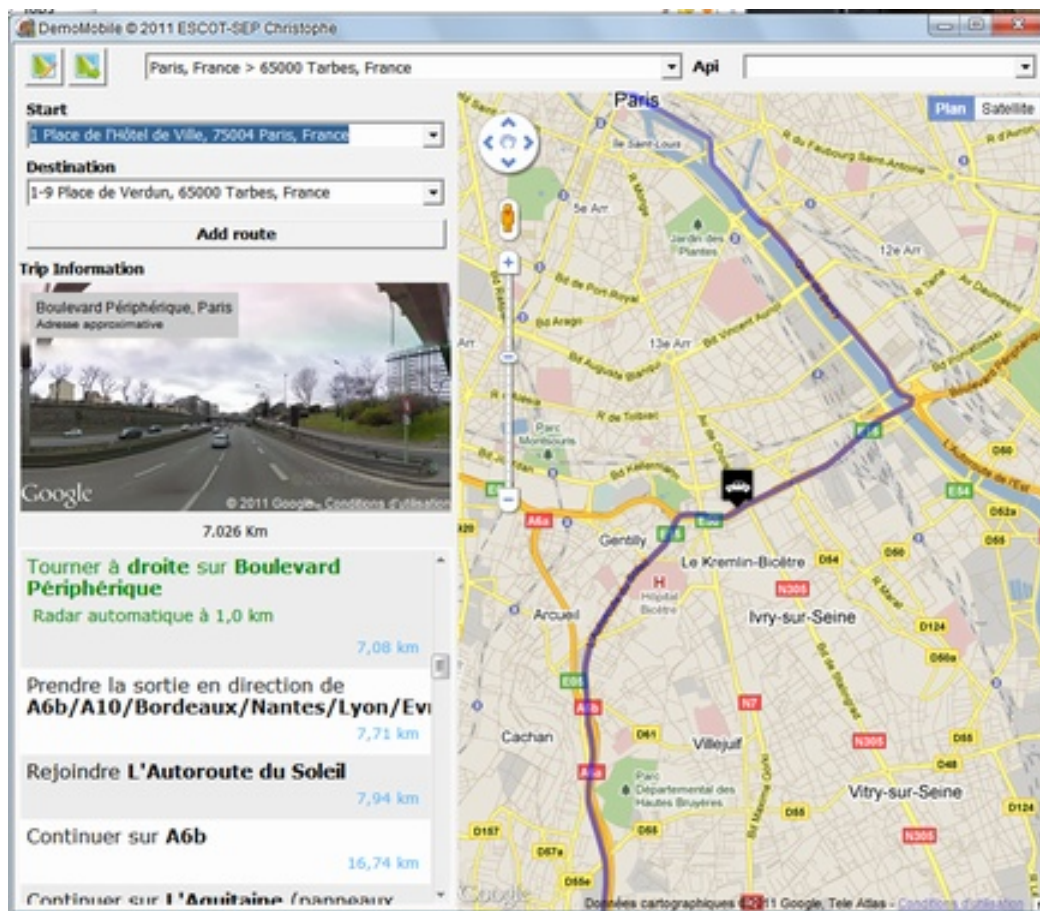


Fig. 7 DemoMobile

Demo3D utilise **EarthView** et vous montre comment afficher un **modèle 3D** et lui faire suivre une route



DemoOverlays vous fait découvrir la gestion dynamique des [overlays](#)

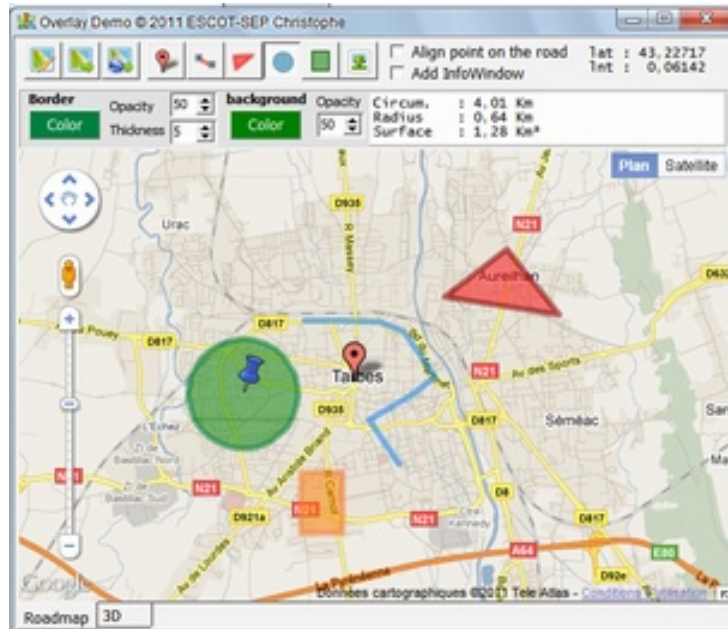
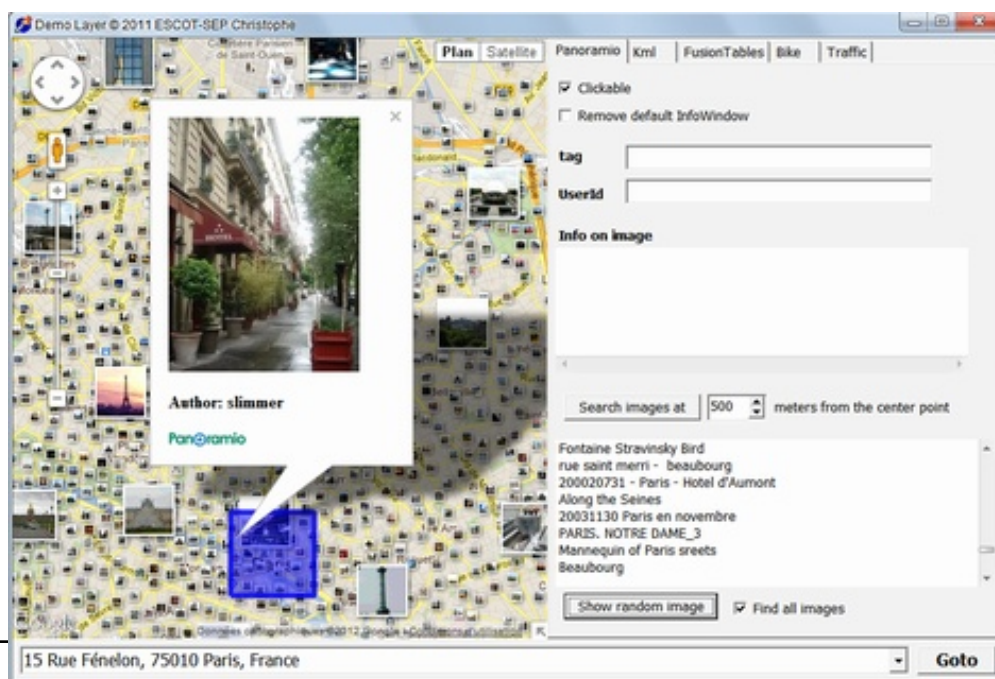


Fig. 9 DemoOverlays

DemoLayer vous permet de voir comment utiliser [Panoramio](#) ainsi que les autres types de [layers](#)



DemoMatrix est une adaptation en Delphi de l'[exemple javascript de google](#) et vous montre comment utiliser le [service DistanceMatrix](#)

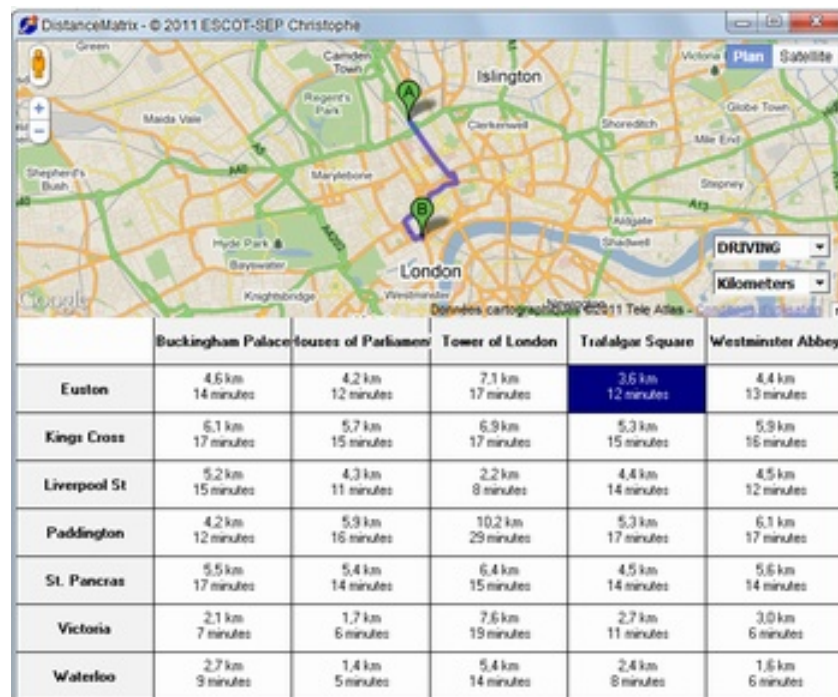
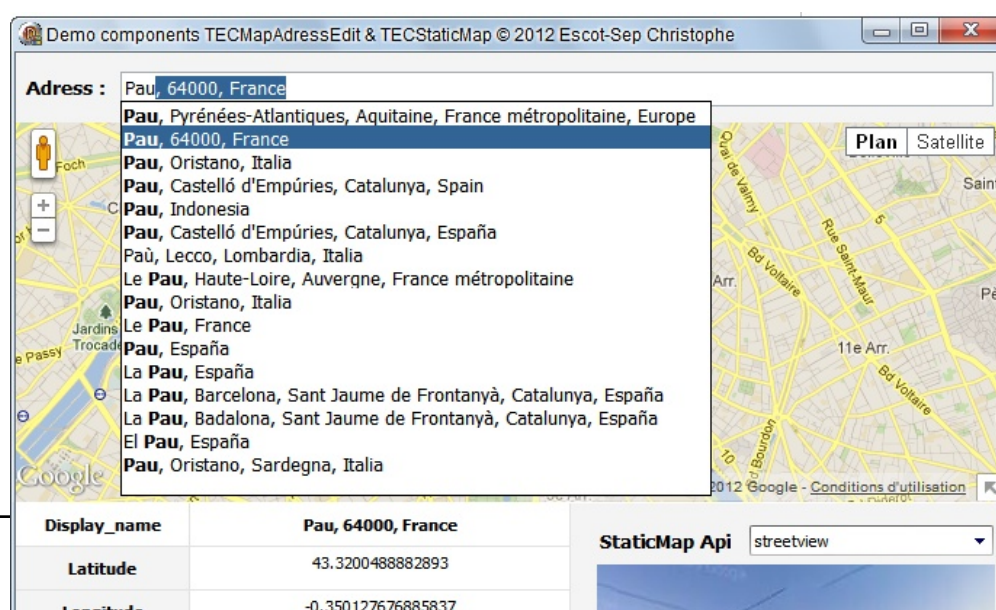


Fig. 11 Démonstration de l'utilisation de DistanceMatrix

DemoAdressEdit vous permet de tester les composants [TECMapAdressEdit](#) et [TECStaticMap](#)



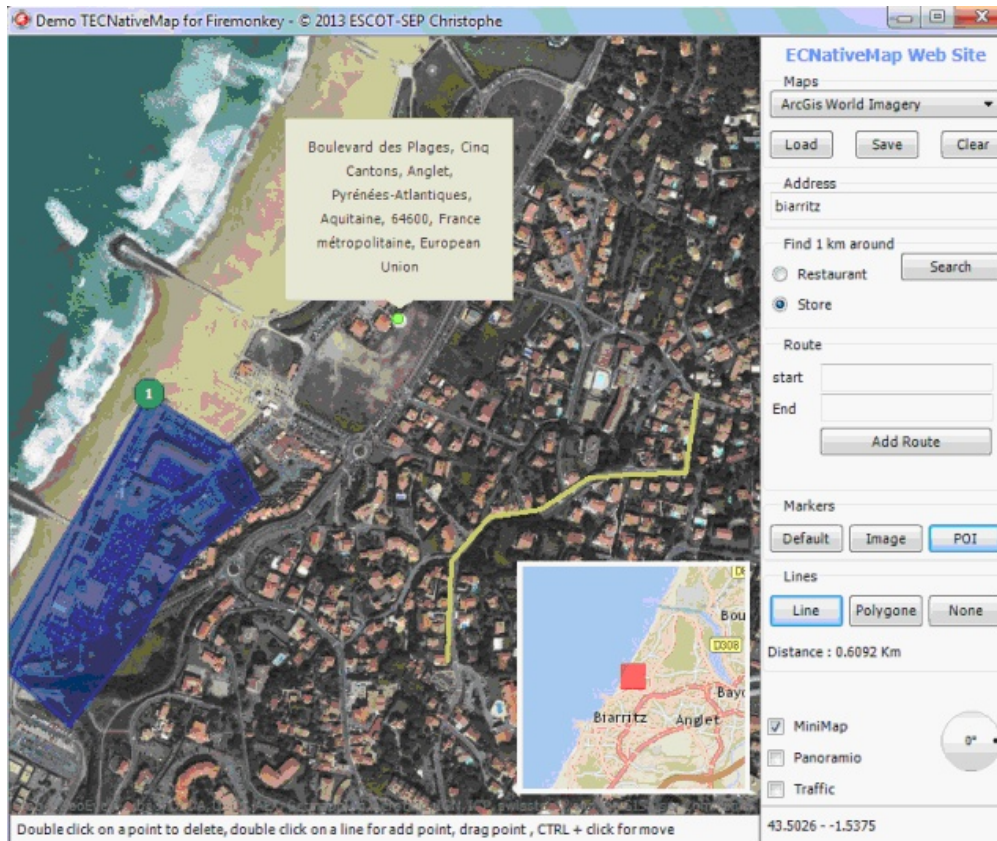


Fig. 13 Firemonkey Demo

Sources

L'intégralité des sources est livrée à l'achat de la licence, mais une version de test est disponible gratuitement et sans condition sur [simple demande](#).

La version d'essai de TECMAP est limitée sur les points suivants

- Seul Google maps est disponible
- C'est une ancienne version non corrigée !

- Pas de support de Chromium
- Google Earth n'est pas disponible en totalité, pas de KML
- Le composant ne doit servir qu'à tester le produit aucune distribution n'est autorisée
- Une boîte d'information est ouverte au lancement

Version d'essai TECNativeMap

**Uniquement les bpl et dcu pour Delphi Windows
32bits VCL/FireMonkey**

Nous allons découvrir les méthodes, propriétés et événements liés à tout ce qui attrait à la position d'un point géographique.

Positionnement sur la carte

Vous pouvez déterminer les coordonnées du centre de la carte au travers des propriétés **Latitude** et **Longitude**, elles sont de type **double** et accessible en **lecture/écriture**

Vous pouvez directement modifier les coordonnées au travers de la procédure **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;
```

La procédure **PanTo(const dLatitude,dLongitude:double)** vous offre aussi la possibilité de changer le positionnement.

Position de la souris

Les latitude et longitude du point situé sous le curseur de la souris sont renvoyés par les propriétés **MouseLatitude** et **MouseLongitude**

Vous pouvez vous brancher sur l'évènement **OnMapMouseMove** pour connaitre en temps réel votre position

Réagir au changement de position

Lorsque le centre de la carte est déplacé, que cela soit par code ou directement à la souris, l'évènement **OnMapMove(sender: TObject;const dLatitude,dLongitude:double)** est déclenché

Sender représente le composant ECMap qui a été modifié, dLatitude et dLongitude les nouvelles coordonnées, qui sont aussi accessible au travers de **Latitude** et **Longitude**

Lorsque la carte commence à bouger l'évènement **OnMapDragStart** est déclenché, puis **OnMapDrag** pendant le déplacement et **OnMapDragEnd** en fin de déplacement.

Vous pouvez aussi réagir lors du déplacement de la souris en vous branchant sur **OnMapMouseMove**

Altitude

Vous obtenez l'altitude du centre de la carte au travers de la propriété **Altitude**, elle est de type **double** et accessible *uniquement en lecture*.

La fonction

GetAltitudeAtLatLng(const dLatitude,dLongitude:double):double vous donne l'altitude d'un point quelconque.

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

Les polygones disposent d'une méthode spéciale permettant le [calcul de l'altitude de l'ensemble de leurs points](#).

Zone affichée

Vous pouvez déterminer les coordonnées du point Nord Est (en haut à droite) et du point Sud Ouest (en bas à gauche) de la zone affichée par le composant ECMap au travers des propriétés **NorthEastLatitude**, **NorthEastLongitude**, **SouthWestLatitude** et **SouthWestLongitude**, elles sont de type double et accessibles *uniquement en lecture*.

La procédure

PanToBounds(const dLatlo,dLnglo,dLathi,dLnghi:double) vous permet de changer la vue en passant les nouvelles coordonnées des point bas (Sud Ouest) et haut (Nord Est).

La procédure **fitBounds(const dLatlo,dLnglo,dLathi,dLnghi:double)** vous permet de d'ajuster la vue aux coordonnées passées.

fitBounds fonctionne aussi avec Google Earth

La fonction

ContainsLatLng(var dLatitude,dLongitude:double):boolean vous indique si le point en dLatitude,dLongitude est dans la portion visible de la carte.

Dès que la vue change l'évènement

OnChangeMapBounds(sender: TObject) est déclenché.

La propriété **ScreenShot** retourne un TBitmap contenant l'image de

la carte

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

Zoom

La propriété **Zoom** vous permet de contrôler la définition de votre carte, elle est de type integer et est accessible en **lecture/écriture**

*En utilisant une carte de type Google vous avez accès aussi au propriété **MaxZoom** et **MinZoom** qui vous permettent de déterminer les limites du zoom*

Le changement de zoom déclenche l'évènement **OnChangeMapZoom(sender: TObject)**

Distance

La fonction **DistanceFrom**(const dLatitudeStart,dLongitudeStart,dLatitudeEnd,dLongitudeEnd:double):double permet de calculer la distance entre 2 points, le résultat est en **mètres**.

Angle par rapport au Nord

La fonction **HeadingFrom**(const dLatitudeStart,dLongitudeStart,dLatitudeEnd,dLongitudeEnd:double):integer

vous donne l'angle, de 0° à 360°, pour la direction allant du point
dLatitudeStart,dLongitudeStart au point
dLatitudeEnd,dLongitudeEnd

Nous allons découvrir les méthodes, propriétés et événements liés à tout ce qui attrait à la position d'un point géographique.

Positionnement sur la carte

Vous pouvez déterminer les coordonnées du centre de la carte au travers des propriétés **Latitude** et **Longitude**, elles sont de type **double** et accessible en **lecture/écriture**

Vous pouvez directement modifier les coordonnées au travers de la procédure **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;
```

La procédure **PanTo(const dLatitude,dLongitude:double)** vous offre aussi la possibilité de changer le positionnement.

Position de la souris

Les latitude et longitude du point situé sous le curseur de la souris sont renvoyés par les propriétés **MouseLatitude** et **MouseLongitude**

Vous pouvez vous brancher sur l'évènement **OnMapMouseMove** pour connaitre en temps réel votre position

Réagir au changement de position

Lorsque le centre de la carte est déplacé, que cela soit par code ou directement à la souris, l'évènement **OnMapMove(sender: TObject;const dLatitude,dLongitude:double)** est déclenché

Sender représente le composant EMap qui a été modifié, dLatitude et dLongitude les nouvelles coordonnées, qui sont aussi accessible au travers de **Latitude** et **Longitude**

Lorsque la carte commence à bouger l'évènement **OnMapDragStart** est déclenché, puis **OnMapDrag** pendant le déplacement et **OnMapDragEnd** en fin de déplacement.

Vous pouvez aussi réagir lors du déplacement de la souris en vous branchant sur **OnMapMouseMove**

Altitude

Vous obtenez l'altitude du centre de la carte au travers de la propriété **Altitude**, elle est de type **double** et accessible *uniquement en lecture*.

La fonction

GetAltitudeAtLatLng(const dLatitude,dLongitude:double):double vous donne l'altitude d'un point quelconque.

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

Les polygones disposent d'une méthode spéciale permettant le [calcul de l'altitude de l'ensemble de leurs points](#).

Zone affichée

Vous pouvez déterminer les coordonnées du point Nord Est (en haut à droite) et du point Sud Ouest (en bas à gauche) de la zone affichée par le composant EMap au travers des propriétés **NorthEastLatitude**, **NorthEastLongitude**, **SouthWestLatitude** et **SouthWestLongitude**, elles sont de type double et accessibles *uniquement en lecture*.

La procédure

PanToBounds(const dLatlo,dLnglo,dLathi,dLnghi:double) vous permet de changer la vue en passant les nouvelles coordonnées des point bas (Sud Ouest) et haut (Nord Est).

La procédure **fitBounds(const dLatlo,dLnglo,dLathi,dLnghi:double)** vous permet de d'ajuster la vue aux coordonnées passées.

fitBounds fonctionne aussi avec Google Earth

La fonction

ContainsLatLng(var dLatitude,dLongitude:double):boolean vous indique si le point en dLatitude,dLongitude est dans la portion visible de la carte.

Dès que la vue change l'évènement

OnChangeMapBounds(sender: TObject) est déclenché.

La propriété **ScreenShot** retourne un TBitmap contenant l'image de

la carte

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

Zoom

La propriété **Zoom** vous permet de contrôler la définition de votre carte, elle est de type integer et est accessible en **lecture/écriture**

*En utilisant une carte de type Google vous avez accès aussi au propriété **MaxZoom** et **MinZoom** qui vous permettent de déterminer les limites du zoom*

Le changement de zoom déclenche l'évènement **OnChangeMapZoom(sender: TObject)**

Url

La propriété **Url** retourne/accepte une chaîne au format '**#zoom/Latitude/Longitude**'

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

L'assignation d'une valeur à cette propriété déclenche l'évènement **OnBeforeUrl**(sender : TObject; var Url:string) qui peut vous permettre de modifier l'url, et même d'annuler le changement en retournant une

chaîne vide.

Vous pouvez passer une tel lien dans une [InfoWindow](#)

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

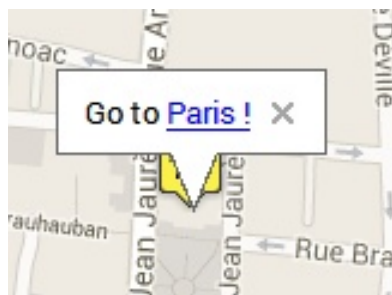


Fig. 14 Url Link

Si vous passez une url "classique" elle s'ouvrira dans votre navigateur par défaut.

Distance

La fonction **DistanceFrom**(const dLatitudeStart,dLongitudeStart,dLatitudeEnd,dLongitudeEnd:double):double permet de calculer la distance entre 2 points, le résultat est en **mètres**.

Angle par rapport au Nord

La fonction **HeadingFrom**(const

dLatitudeStart,dLongitudeStart,dLatitudeEnd,dLongitudeEnd:double):integer

vous donne l'angle, de 0° à 360°, pour la direction allant du point

dLatitudeStart,dLongitudeStart au point

dLatitudeEnd,dLongitudeEnd

API

TECMap vous permet d'utiliser soit l'[API Google Maps](#) soit l'[API CloudMade](#) soit l'[API OpenMapQuest](#) soit l'[API Leaflet](#) pour afficher vos cartes, vous effectuer le basculement en paramétrant la propriété **MapAPI** à **apiGoogle** , **apiCloudMade** , **apiOpenMapQuest** ou **apiLeaflet**

La propriété **SSL** vous permet d'utiliser une connexion sécurisé, pour l'instant seule l'api Google offre cette fonctionnalité.

La propriété **ApiVersion** de type string vous permet d'indiquer la version de l'api que vous souhaitez employer, cela ne fonctionne que pour Google, si vous n'en indiquez pas c'est la dernière version en date qui sera utilisée.

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

Cela peut-être utile en cas de bug de certaines versions.

Le changement d'API provoque les évènements

OnBeforeChangeMapApi et **OnAfterChangeMapApi**, le premier à lieu juste avant le changement effectif, le second juste après.

Lorsque la carte est prête l'évènement **OnLoad** est déclenché, vous pouvez aussi vous brancher sur **OnBeforeLoad** si vous souhaitez modifier l'url de connexion à l'api et rajouter du code pour connecter des bibliothèques javascript


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

Vous pouvez recharger une carte et son contenu en appelant la fonction **Reload**, utile pour forcer une libération de mémoire, les événements **OnBeforeReload** et **OnAfterReload** sont disponibles.

Vous n'avez pas à vous préoccuper de sauvegarder le contenu de la carte, il sera intégralement rechargé à l'exception du résultat d'une [recherche de lieux](#).

*Le changement des propriétés **MapApi**, **ApiVersion**, **SSL** déclenche un rechargement.*

[StreetView](#) ayant des fuites de mémoire, en attendant que google les corrige, un appel à `reLoad` est déclenché automatiquement toutes les 1000 vues.

TECMap se cale au plus près des apis, n'hésitez pas à lire leur documentation respective.

- [Documentation Google Maps](#)
- [Documentation CloudMade](#)
- [Documentation OpenMapQuest](#)
- [Documentation Leaflet](#)

En plus de la [licence d'utilisation que je vous invite à lire](#), les principales différences sont l'absence de SSL, de vue satellite, de [vue 3D](#), de [Panoramio](#), de [DistanceMatrix](#) et de [StreetView](#) dans l'api CloudMade et OpenMapQuest.

Les différences ne déclenchent pas d'erreur dans le composant TECMap, l'utilisation d'une propriété non supportée n'a simplement pas d'effet, il est possible que suivant l'évolution future des apis les manques soient corrigés.

[Vous pouvez changer d'api à tout moment sans perte de données, à l'exception des résultats d'une recherche de lieux.](#)

Type de carte

OpenMapQuest supporte la carte classique, une vue satellite et une vue hybride.



Avec CloudMade et Leaflet vous avez la carte routière classique.



Fig. 16 Carte CloudMade


Vous avez aussi les cartes OpenMapStreet, Mapnik, Osmarender, CycleMap et [MapQuest](#) (standard, satellite et hybride) 



Fig. 17 Carte OpenMapStreet Mapnik





Fig. 19 OpenMapStreet CycleMap



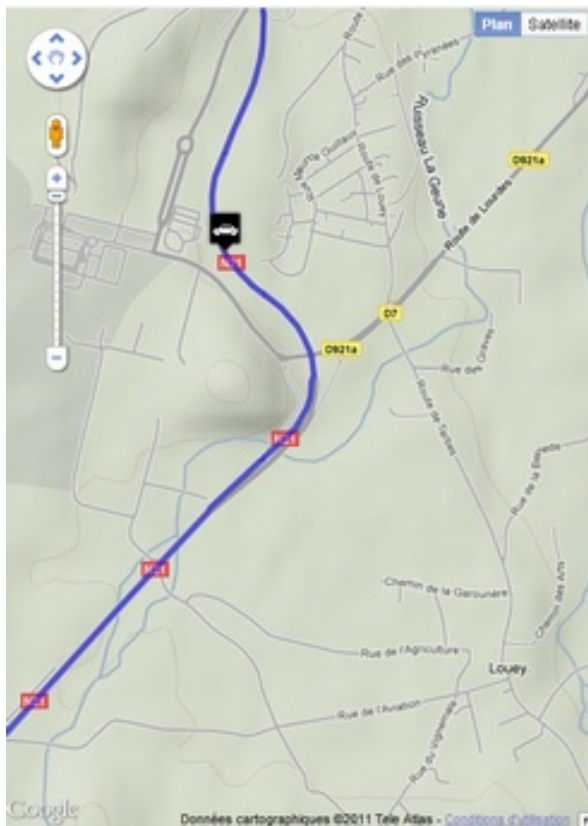
Avec l'api Google vous avez la carte classique, la vue satellite avec ou sans les légendes, une vue relief, OpenMapStreet Mapnik, OpenMapStreet A Render, CycleMap et MapQuest

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




g. 21 Google : Type Roadmap





g. 23 Google : Type TERRAIN

Vous changez de vue au travers de la propriété

MapTypeId de valeur possible

mtROADMAP

mtSATELLITE

mtHYBRID,

mtTERRAIN, mtMAPNIK, mtOSMARENDER, mtMapQuest et mtCYCLEMAP

Pour les images satellites (mtSATELLITE, mtHYBRID) Google Map commence à mettre en place

Tilt

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

```
mapApiVersion = 3.7 for google api
```

Cela n'est pas mis en place partout,

[voir la documentation de google pour plus d'infos](#)

Lors du changement de type de carte que cela soit par code ou directement sur la carte par le menu **Change Map Type**(Sender: TObject)

Et sur [Google Maps](#) vous disposez aussi des vues

Styles

La propriété

Styles permet de redéfinir l'affichage des cartes standards.

// Delphi map component EMap

MapAPIVersion: 3 for google api

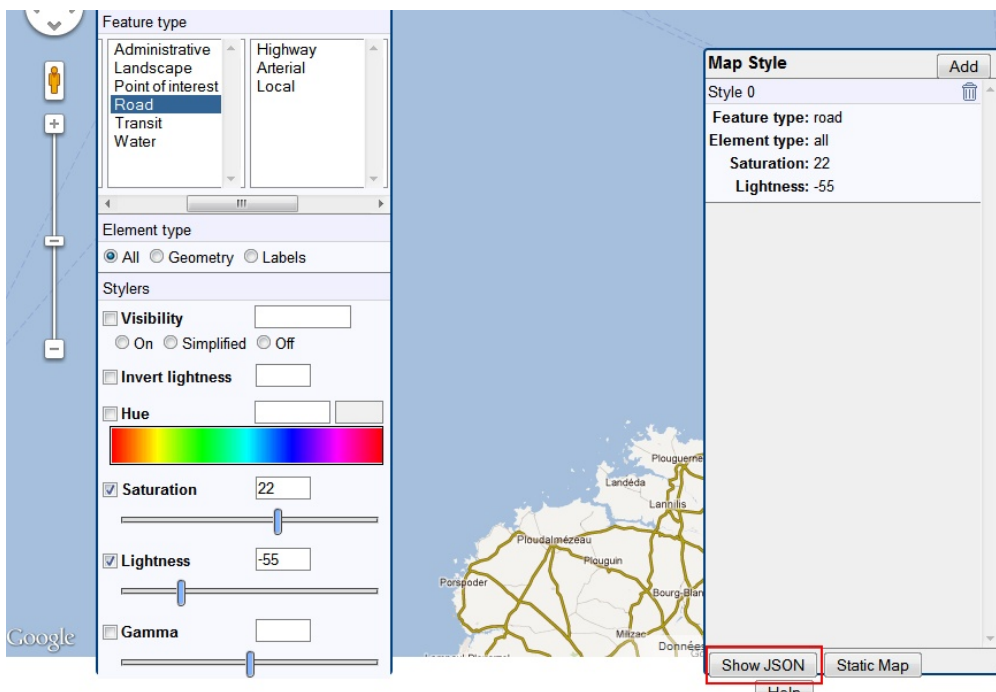
Pour plus d'informations sur la syntaxe des styles.

Google met aussi à votre disposition

Il vous suffit simplement d'assigner à

mapStyles au format

JSON par cet assistant.



g. 24 Obtenir les styles au format JSON

Google Maps API v3 Styled Maps JSON

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

g. 25 Styles au format JSON

Supprimer les Points d'intérêts Google Maps

Vous pouvez utiliser les styles pour supprimer tous ou certains POIs de votre carte



g. 26 POI

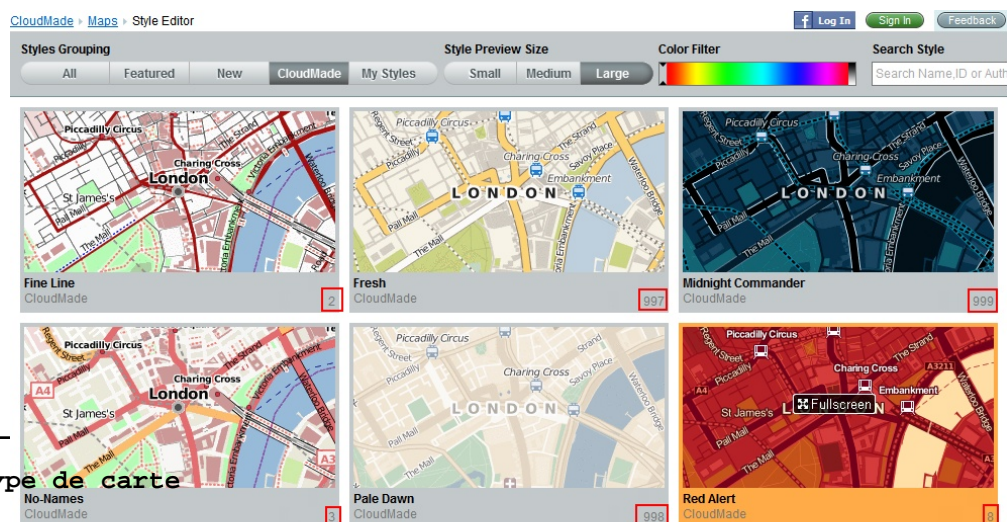
Cet exemple vous montre comment effacer tous les POIs sauf les poi.park

```
map.setStyle({featureType: "poi",elementType: "all", stylers: [{visibility: "off" } ]})
+,{featureType: "poi.park",elementType: "all",stylers:[{visibility: "on" }, ] } ]'
```



g. 27 Only poi-park

Voir [la documentation de Google](#) pour toutes les valeurs featureType
Sous CloudMade et Leaflet la propriété
[StyleName](#) qui définit, vous pouvez aller consulter



2
3
8
997
998
999

permettent d'idenfier les styles, exemple d'utilisation

map.Styles:=
'999'

OpenMapQuest ne supporte pas les styles



Fig. 29 Contrôles des cartes Google

- 1 Barre de navigation
- 2 Type de cartes
- 3 Zoom
- 4 Echelle

Avec les cartes CloudMade et OpenMapQuest seul le contrôle du zoom et de l'échelle sont disponibles

Vous pouvez supprimer l'affichage de tous les contrôles en basculant

la propriété **EnableUIControl** sur **false**

```
// Delphi map component EMap  
// hide all controls  
map.EnableUIControl := false;
```

NavigationControl gère l'affichage de la barre de navigation, **MapTypeControl** le type de carte, **ZoomControl** le zoom et **ScaleControl** l'échelle,

NavigationPosition, **MapTypeControlPosition**, **ZoomPosition** et **ScalePosition** gèrent la position du contrôle sur la carte, ces propriétés peuvent prendre les valeurs suivante :

- cpTopLeft
- cpTopCenter
- cpTopRight,
- cpRightTop
- cpRightCenter
- cpRightBottom
- cpBottomRight
- cpBottomCenter
- cpBottomLeft

- cpLeftBottom
- cpLeftCenter
- cpLeftTop

Sous CloudMade seule les positions cpTopLeft, cpTopRight, cpBottomLeft et cpBottomRight sont fonctionnelles

Menu

TECMap est muni d'une propriété **Menu** qui vous permet de lui associer un **TPopupMenu** qui s'ouvrira lors d'un clic droit sur la carte.

L'évènement **OnMapRightClick** est quand même déclenché si un menu est associé, il survient juste après l'ouverture du TPopupMenu

Clefs pour les API

Par défaut TECNativeMap utilise les services d'OpenStreetMap.

Si vous souhaitez utiliser MapQuest vous devez [obtenir une clef auprès de MapQuest](#)

Faites de même pour utiliser les services de [MapZen](#) et [MapBox](#).

La propriété **Address** vous donne l'adresse du centre de la carte, elle est de type **string** et est accessible en **lecture/écriture**

Format de l'adresse

Pour rechercher une adresse vous pouvez utiliser un format libre du genre "adresse,ville, code postal".

L'API CloudMade et OpenMapQuest utilisent [Nominatim Search Service](#)

Limitation des geolocalisations

La conversion coordonnées/adress ouvre une connexion vers le serveur du fournisseur de votre carte, Google met en place un système de quota pour éviter trop de demande en un minimum de temps.

Si vous dépassez cette limite l'évènement **OnJavaScriptError(sender: TObject;const sError:string)** est déclenché avec sError = 'OVER_QUERY_LIMIT'

ECMap met en place un système de cache qui fait que tant que les

coordonnées ne changent pas la connexion sur le serveur n'a lieu que pour la première consultation, vous pourrez donc effectuer plusieurs lecture sans vous soucier du quota.

Ce système de cache est valable pour tous les objets permettant leur geolocalisation, par exemple les Markers

Vous pouvez pour vos recherches

Geolocalisation

Pour obtenir l'adresse d'un point précis vous avez la fonction **GetAddressFromLatitude,dLongitude:double):string;**

Vous pouvez obtenir les coordonnées d'une adresse avec la fonction **GetLatLngFromAddress:string;var dLatitude,dLongitude:double):boolean;**

En attribuant une valeur à la propriété **Address** vous allez changer la position du centre de votre carte pour la faire correspondre à l'adresse indiquée

```
// Delphi map component EMap  
// move center of map  
map.Address := 'Tarbes';
```

Cela va déclencher l'évènement **OnAddress(sender: TObject;const sAddresses:string;var Index:integer)**

GetLatLngFromAdress *déclenche aussi* OnAdress

sAdresses contiendra toutes les adresses, séparé par #13#10 pouvant correspondre avec votre requête.

Index contiendra l'index de l'adresse retenue, par défaut 0, vous pourrez le modifier dans cet évènement.

En sortie de cet évènement le centre de la carte correspond avec l'adresse retenue

La propriété **Adresses** de type **TGeoCoderReponses** contient l'ensemble des réponses retournées par le serveur.

Exemple d'utilisation

```
// Delphi map component EMap
// move center of map with response 2
if map.Addresses.count>1 then
  map.setCenter(map.Addresses[1].Latitude,map.Addresses[1].Longitude);
```

Geolocalisation asynchrone

GetAdressFromLatLng, GetLatLngFromAdress et Adress := 'paris' attendent le résultat de la requête, vous avez à votre disposition deux procédure asynchrones (non bloquantes) équivalentes.

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

Sous CloudMade cette procedure est aussi bloquante

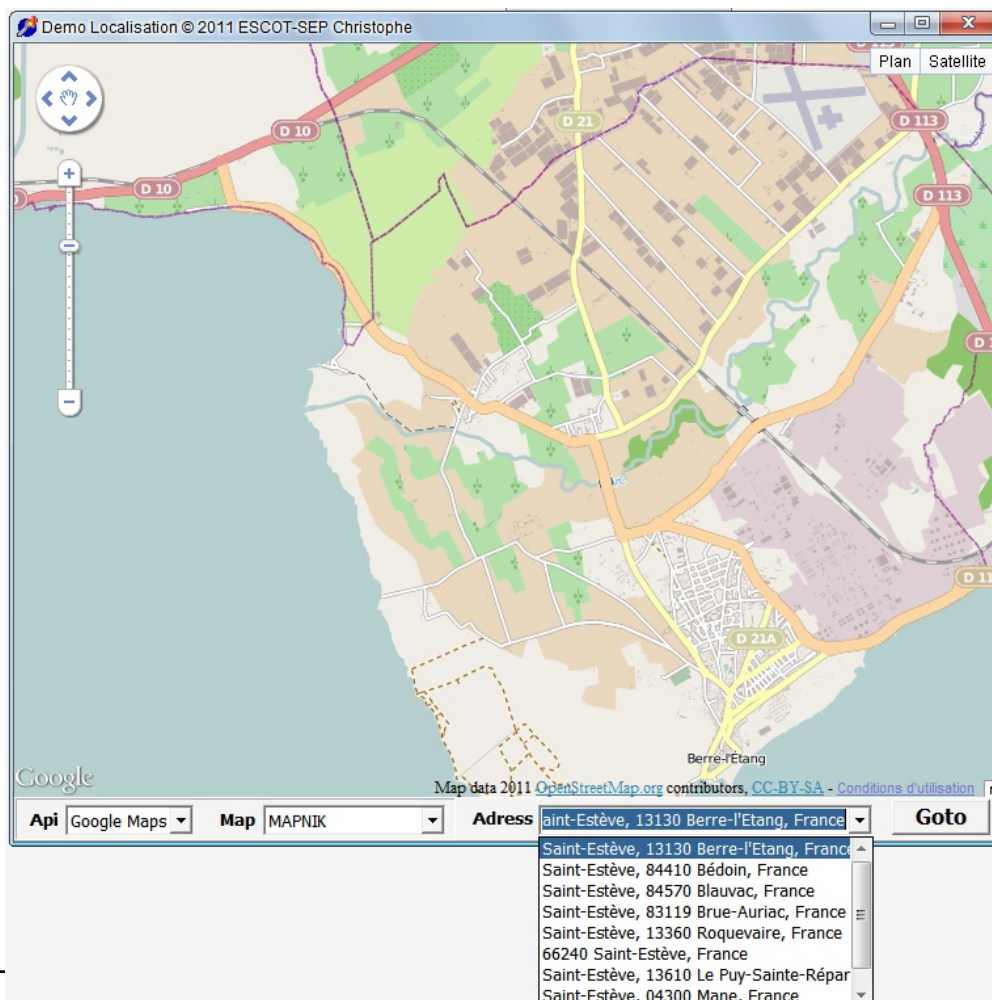
mais pas GeoLocation

procedure **GeoLocation**(const sAdresse:string);

Ces deux procédures déclenchent l'évènement **OnGeoLocation** (sender:TObject;co
dLatitude,dLongitude:double;const sAdresse:string);

dLatitude,dLongitude et **sAdresse** correspondent à la première
réponse disponible, comme pour les fonctions synchrones la propriété
Adresses contient l'ensemble des résultats.

*Vous pouvez utiliser le composant TComboBox pour à la
fois contenir la requête et le résultat de celle-ci, la démo
DemoLocalise vous montre comment faire*



Propriétés de TGeoCoderReponses

property **Addresses:string**

La liste de toute les adresses séparées par #13#10

property **Adresse[index:integer] : string**

L'adresse numéro Index

property **Error [index:integer] : string**

L'eventuelle erreur

property **Latitude[index:integer] : double**

La latitude correspondante à l'adresse Index

property **Longitude[index:integer] : double**

La longitude correspondante à l'adresse Index

property **LocationType[index:integer]: string**

La précision de la réponse (uniquement pour google)

Historique des réponses

Chaque geolocalisation va changer le contenu de la propriété

Addresses, vous pouvez la stocker dans une liste spécialisée,

accessible depuis la propriété **ListAdresses de type**

TListGeoCoderReponses, en vue d'une utilisation ultérieure, le bon moment est dans l'évènement **OnAdress**

Propriétés de TListGeoCoderReponses

function Add(const geoCode:TGeoCoderReponses):integer;

Ajoute une valeur TGeoCoderReponses dans la liste

```
function count:integer;
```

Nombre de TGeoCoderReponses stockées

```
procedure clear;
```

Efface toutes les valeurs

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

Supprime la valeur d'indice index

```
property GeoCoderReponses[index:integer]:TGeoCoderReponses;
```

Accès en lecture/écriture aux valeurs par leur indice

Utilisation de ListAddresses

Le cas typique d'utilisation de cette liste est celui où vous devez gérer une adresse de départ et une d'arrivée, les demos DemoMobiles et DemoRoutes vous montre une façon de faire.

Une ComboBox est utilisée pour le départ et une pour l'arrivée, lors de la validation d'une adresse par **ENTREE** une demande de géolocalisation est faite.

Dans l'évènement **OnAddress** on stocke dans la ComboBox les adresses en clairs, on place dans **ListAddresses** les données intégrales retournées par la requête, et on stocke l'indice dans le tag de du Combobox.

On pourra alors obtenir les coordonnées de n'importe quelle adresse

placée dans la ComboBox.

```
// Delphi map component EMap

// sample for DemoMobile

{
    press RETURN key on combobox start and destination for
    Geolocalize adress

    fired event OnAdress
}
procedure TFDemoMobile.cbStartKeyPress(Sender: TObject;
var Key: Char);
var lat,lng : double;
begin
    if Key=#13 then
    begin
        FComboQueryAddress := Sender As TComboBox;
        if assigned(FComboQueryAddress) then
            map.GetLatLngFromAdress(FComboQueryAddress.text,lat,lng)
        end;
    end;

{
    event OnAdress

    fired by map.GetLatLngFromAdress(...)

    @param sender instance of component TECMap

    @param sAdresses list of possible addresses (adr0#13#10adr1#13#10...)
    @param index selected index adress
}
procedure TFDemoMobile.mapAdress(sender: TObject; const
    sAdresses: String;
    var Index: Integer);
begin
    if assigned(FComboQueryAddress) then
    begin
        FComboQueryAddress.Items.text := sAdresses;
        FComboQueryAddress.ItemIndex := index;

        // save actuel map.Adresses in map.ListAdresses
```

```

        // Add value if necessary
        if FComboQueryAdress.Tag > -1 then
            map.ListAdresses[FComboQueryAdress.Tag]
:= map.Adresses
        else
            FComboQueryAdress.Tag
:= map.ListAdresses.Add(map.Adresses);

        end;
    end;

    // Find Latitude, Longitude of ComboBox Itemindex
    function TFDemoMobile.SelectLatLng(const cb : TComboBox;
var Lat,Lng:double):boolean;
begin

    result := false;

    if not assigned(cb) then exit;

    if cb.ItemIndex > -1 then
    begin
        if (cb.Tag > -1) and (cb.Tag < map.ListAdresses.count) then
        begin
            Lat
:= map.ListAdresses[cb.Tag].latitude[cb.ItemIndex];
            Lng
:= map.ListAdresses[cb.Tag].longitude[cb.ItemIndex];

            result := true;
        end;
    end;

end;
end;

```

Utiliser les services d'OpenMapQuest avec Google Maps

En basculant la propriété **UseOpenMapQuestServices** à true vous utilisez OpenMapQuest pour tous ce qui touche à la geolocalisation, l'altitude et la [recherche de lieux](#) (Places)

La propriété **Places** de type **TECPlaces** vous permet d'effectuer des recherches sur des lieux spécifiques dans une zone donnée, par exemple trouver des restaurants dans un rayon de 500 mètres.

Avec l'api Google vous allez utiliser le [service Places](#), avec les autres vous employez [MapQuest Xapi API Service](#) qui utilise les données d'[OpenStreetMap](#), cela entraîne une petite différence dans la syntaxe de la recherche que nous détaillerons par la suite.

*Vous pouvez forcer l'utilisation de Xapi sous Google en fixant à true la propriété **UseOpenMapQuestServices***

La propriété **XapiServer** vous permet d'utiliser un autre serveur Xapi que celui de MapQuest

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

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

Le service Places de google est limité à 20 résultats par requête

TECPlaces

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

Lancement d'une recherche, tags contient la requête, la syntaxe varie en fonction de l'api Google ou CloudMade

Liste des tags disponibles sous Google

[Documentation Xapi](#)

```
// 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;

      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);
```

Lorsque la recherche est terminée, l'évènement **OnPlacesSearch** est déclenché

Chaque lancement de Search efface les résultats d'une précédente recherche

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

Lancement d'une recherche en langage naturel, uniquement disponible avec l'api Google

Lorsque la recherche est terminée, l'évènement **OnPlacesSearch** est déclenché

property **AutoComplete** : boolean

Affiche ou non un champ de recherche auto complété, uniquement disponible avec l'api Google

Lorsque une sélection est terminée, l'évènement **OnPlaceAutoComplete** est déclenché

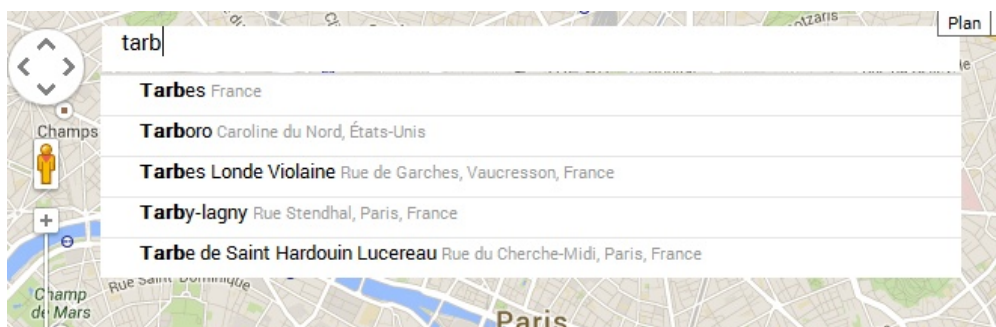


Fig. 31 Place AutoComplete

Si vous souhaitez adapter la taille du champ de recherche à la largeur de la carte, branchez vous sur l'évènement **OnResize** et ajoutez cette ligne

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

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

Propriété en **lecture/écriture** qui indique le point central de la zone de recherche, elle a la priorité sur les propriétés **Latitude** et **Longitude**

property **Status** : string;

Propriété en **lecture seule** qui retourne une string indiquant le status de la recherche, 'OK' si tout c'est bien passé

Le status est consultable dans l'évènement **OnPlacesSearch**

property ItemDetail : integer;

Propriété en **lecture seule** qui contient l'index du résultat dont on cherche les détails supplémentaires

property **Latitude** : double;

Propriété en **lecture/écriture** qui indique la latitude du point central de la zone de recherche, cela invalide le contenu de la propriété **Adress**

property **Longitude**: double;

Propriété en **lecture/écriture** qui indique la longitude du point central de la zone de recherche, cela invalide le contenu de la propriété **Adress**

property Radius : integer;

Propriété en **lecture/écriture** qui indique le rayon de recherche **en mètres**

property **useOSM** : boolean;

Permet d'utiliser les données d'OpenStreetMap avec l'api Google

property **maxResult** : integer;

limite le nombre de résultat pour une recherche dans les données d'OpenStreetMap, ignoré avec **Google Places**

property **Searching**: boolean;

Propriété en **lecture seule** qui indique si une recherche est en cours

property **Results**:TECPlaceResults;

Liste des resultats, l'événement **OnPlacesSearch** est déclenché lorsque les résultats sont disponibles

TECPlacesResults

Cette classe gère la liste des resultats retournés par **Search**

procedure **Clear**;

Efface l'ensemble des résultats

function **Count**:integer;

Retourne le nombre de résultat correspondant à la requête

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

Efface le résultat dont on passe l'index

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

Tableau permettant l'accès aux résultats, c'est la propriété par défaut donc vous pouvez y accéder directement par **map.Places.Results[index]** au lieu de **map.Places.Results.Result[index]**

TECPlaceResult

Classe gérant un résultat correspondant à une recherche

procedure **getDetails**;

Effectue une requête supplémentaire sur le résultat pour obtenir des détails

 *Non disponible sous CloudMade*

Lorsque les détails sont disponibles l'évènement **OnPlacesDetail** est déclenché.

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

Retourne la valeur de la clef que l'on passe en paramètre

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

Retourne la valeur de la clef que l'on passe en paramètre pour les détails

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

Retourne la clef d'un résultat en fonction de son index

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

Retourne la clef d'un détail en fonction de son index

property **CountResult**:integer;

Retourne le nombre d'élément (clef=valeur) d'un résultat

property **CountDetail**:integer read getCountDetail;

Retourne le nombre d'élément (clef=valeur) d'un détail

property **Latitude** : double;

Latitude du résultat

property **Longitude**: double;

Longitude du résultat

property **RawResult** : string;

Retourne l'ensemble des éléments du resultat sous la forme d'une chaîne constituée de lignes
clef=valeur

property **RawDetail** : string;

Retourne l'ensemble des éléments du détail sous la forme d'une chaîne constituée de lignes
clef=valeur

```
// 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 résultat
        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;

end;

```

Les résultats ne sont pas conservés lors d'un rechargement de la carte (reLoad, changement d'api ou autre), ils ne sont pas non plus enregistrés lors de la sauvegarde

Démonstration

le programme **DemoLocalise** vous montre comment gérer **Places**



TECMap permet d'enregistrer et de recharger la totalité de ses données dans un simple fichier texte, cela comprend non seulement les paramètres du composant et des vues mais aussi les données cartographiques des [overlays](#)

Vous pouvez aussi importer/exporter vos données aux formats [GPX](#), [KML](#) et [GeoJSON](#) mais dans ces formats seules les données géographiques sont exploitées.

Sauvegarde/Restauration

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

Enregistre la carte dans un fichier texte.

Utilisez les extensions .gpx , .kml et .json pour spécifier un format, autrement c'est le format interne d'ECMap qui sera employé

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

Charge la carte avec un fichier texte

Utilisez les extensions .gpx , .kml et .json pour spécifier un format, autrement c'est le format interne d'ECMap qui sera employé

LoadFromFile peut télécharger les fichiers sur internet

```
property toGPX : string;
```


Propriété en **lecture/écriture** qui donne accès aux données de la carte dans le format GPX.

Cette propriété est utilisée par **SaveToFile**, **LoadFromFile**.

property **toTxt** : string;

Propriété en **lecture/écriture** qui donne accès aux données de la carte dans un format texte.

Cette propriété est utilisée par **SaveToFile**, **LoadFromFile**.

property **ToKml** : string;

Propriété en **lecture / écriture** qui retourne les **overlays** (tous sauf les **Labels**) au **format Kml**

Cette propriété est utilisée par **SaveToKmlFile**

EarthView permet l'import/export KML dans un format plus complet

property **ToGeoJSON** : string;

Propriété en **lecture / écriture** qui retourne/importe les **overlays** (juste les markers, les polylines, les polygones, les cercles et les rectangles) au **format GeoJSON**

L'affectation d'une valeur à **toTxt**, **ToKml**, **ToGpx**, **ToGeoJSON** ou le chargement d'un fichier par **LoadFromFile** provoque les événements **OnBeforeChangeToTxt** et **OnAfterChangeToTxt**, le premier à lieu juste avant le changement de valeur et le second juste après.

OnLoadOverlay est aussi déclenché après l'ajout de chaque overlay (marker, polyline, polygone...) cela peut vous permettre d'ajuster les propriétés pendant le chargement des données.

Vous pouvez stopper le chargement en basculant **StopLoadOverlay** à 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;

```

Paramétrage import/export

Lorsque vous importez/exportez au format texte, par défaut l'ensemble des données le sont, que cela soit les paramètres de la carte (type d'api, position etc) ou les overlays ([markers](#), circles, [routes](#) etc).

De même que lors d'une importation les anciens overlays sont remplacés par les nouveaux.

La propriété **ToTxtType** vous offre un certain contrôle.

```

// 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];
```

format texte

chaque partie est découpée en section, les données sont nommées et n'ont pas besoin d'être dans un ordre particulier.

Pour les overlays chaque ligne correspond aux données d'un élément (un marker, un cercle etc)

Exemple de carte

```
[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;s
```

```

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

content=<h1>Nuage</h1><br>Image locale;anchor=2;maxwidth=0;zindex=0;lat=43.23
[routes]
startlat=43.23443;startlng=0.0737;endlat=43.24342;endlng=0.09581;wplat0=43.24

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

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

/ Paris;marker.icon=http://maps.google.com/mapfiles/ms/micons/purple-pushpin.
startlat=43.00821;startlng=-0.09972;endlat=42.85121;endlng=-0.13994;travelmod
d'espagne;marker.icon=;marker.visible=true;marker.clickable=true;marker.flat=
[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;direction
distance=21139;marker=3;route=3;speed=30;segment=329;mobile=true;direction=st
[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=43.2287477339158,0.073096832238
[polygones]
color=16744448;opacity=80;weight=5;zindex=1;latlng=43.2392845896161,0.0741697
[circles]
color=16384;opacity=80;weight=3;zindex=1;latlng=43.2348454,0.0756332;visible=
[rectangles]
[labels]

```

Import/Export des overlays

Les overlays disposent eux aussi de propriété **toTxt** permettant

l'importation et l'exportation, que cela soit au niveau de leur liste ([Markers](#), [Routes](#) etc) que de l'élément lui même.

Au niveau des listes l'importation est obligatoirement un ajout, il n'est pas tenu compte de la propriété `toTxtType` qui n'est prise en charge qu'au niveau global.

```
// Delphi map component EMap
// add circle by txt
map.Circles.toTxt :=
;color=16384;opacity=80;weight=3;zindex=1;latlng=43.2348454,0.0756332;visible
```

Lorsque vous utilisez `toTxt` au niveau des overlays vous ne devez pas indiquer la section, toutes les propriétés ne sont pas nécessaire, les absentes auront des valeurs par défaut.

Les overlays sont tous les éléments basés sur des points géographiques que vous pouvez incruster sur vos cartes, ils descendent de la classe `TECMapItem`.

TECMapItem

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

Modifie la position géographique de l'élément

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

Ajoute l'élément dans un [groupe](#)

property **Id**: integer;

Indice de l'élément dans sa liste

property **ItemType** : TOverlayType ;

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

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

Indique le groupe auquel appartient l'élément, positionnez à nil pour sortir l'élément du groupe

Liste d'overlays

Chaque type d'overlay est géré dans une liste séparée, vous pourrez ajouter, supprimer, importer/exporter vos éléments

Les listes sont accessibles au travers des propriétés:

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

elles se manipulent au travers de méthodes et propriétés similaires, seule le paramétrage de la fonction d'ajout diffère.

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;

Pour les **Routes**

*Add retourne dans tous les cas le numéro d'index de
l'élément ajouté*

```
// 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;

supprime l'ensemble des éléments

function Count:integer;

retourne le nombre d'élément

procedure Delete(index:integer);

supprimer l'élément numéro **index**

property **ToTxt** : string

propriété en lecture/écriture permettant d'exporter/importer au format texte l'ensemble des éléments

*L'importation fonctionne comme un ajout, les éléments préexistants sont conservés, si vous souhaitez remplacer ceux-ci il faut avant l'importation faire appel à **Clear***

property **ToKml** : string

Exportation au [format Kml](#) de l'ensemble des éléments

Les labels ne sont pas exportables au format Kml

TECMapOverlay

Les cercles, rectangles, polylignes, polygones et groundOverlays descendent du type **TECMapOverlay**, **TECMap** dispose d'une propriété de ce type nommée **EditOverlay**.

En lui assignant un overlay compatible (cercle, rectangle, polyline, polygone ou GroundOverlay), vous pouvez modifier à la souris sa position et sa taille directement depuis votre carte.

Pour cela vous devez basculer à True la propriété **EditMode** de **TECMap**

EditOverlay dispose de la propriété **OverlayType** de type **TOverlayType** qui vous permet de connaître le type d'overlay

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

EditOverlay a aussi une propriété **Id** de type integer qui vous donne le numéro d'index de votre overlay dans sa liste.

```
// 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;
// edit overlay with mouse
map.EditOverlay := map.Circles[id];
// map.EditOverlay.OverlayType = ovCircle
// map.EditOverlay.Id = id
```

Méthodes & Propriétés communes

Ce type d'overlay partage les propriétés

Color : TColor

Couleur du trait extérieur

Clickable : boolean

Permettre ou non à l'élément de réagir au clic de la souris, à **False** l'overlay ne pourra pas basculer en mode édition même si **EditMode** est à **True**.

Geodesic : boolean

Rendre chaque bordure comme une géodésique (un segment d'un "grand cercle"). Une géodésique est le plus court chemin entre deux points le long de la surface de la Terre.

Id : integer

Numéro d'index de l'overlay dans sa liste

InfoWindow : integer

Numero d'index de l'[InfoWindow](#) associée, -1 si aucune

Lors d'un clic sur l'overlay si une infowindow est définie elle sera ouverte

 *Non disponible sous CloudMade*

Opacity : double

Pourcentage d'opacité de la couleur

Visible : boolean

Visible ou pas

Weight : integer

Épaisseur du trait

ZIndex : integer

Indice de profondeur par rapport aux autres overlays, permet de déterminer si un overlay est "au-dessus" d'un autre

ToTxt : string

Propriété en lecture/écriture permettant d'exporter/importer au format texte l'ensemble des propriétés de l'overlay

Tag : integer

Vous pouvez utiliser cette propriété comme bon semble

Lorsque vous modifiez les propriétés Color, Opacity, Weight ou ZIndex vous devez explicitement faire un appel à la procédure reDraw pour que cela soit pris en compte

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

Les cercles, rectangles et les polygones disposent en plus des propriétés

Area : double

Surface en m²

Distance : double

Distance du pourtour en m

FillColor : TColor

Couleur intérieur

FillOpacity : double

Pourcentage d'opacité de la couleur d'intérieur

Les polylines disposent aussi de la propriété Distance

Les cercles possèdent les propriétés

Center : TLatLng

Coordonnées du centre du cercle

Radius : Integer

Rayon en m

Les rectangles possèdent les propriétés

Height : double

Hauteur en m

Width : double

Largeur en m

TLatLngList

Les polygones et polygones possèdent une propriété **Path** de type **TLatLngList**, liste des points constituant la figure, dont voici les principales caractéristiques.

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

Ajouter un point en fin de liste

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

Insérer un point en Index

procedure **BeginUpdate**;

IMPORTANT : avant d'ajouter une serie de point faite un appel à BeginUpdate pour ne déclencher la reconstruction qu'une fois tous les points insérés.

procedure **Clear**;

Efface tous les points

function **Count**:integer;

Retourne le nombre de points

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

Efface le point Index

procedure **EndUpdate**;

IMPORTANT : à la fin d'une série d'ajout ou d'insertion, faite un appel à EndUpdate pour répercuter les modification

procedure **PanToBounds**;

Faire glisser la carte de manière à ce que la figure soit entièrement visible

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

Retourne l'adresse du point Index

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

Calcule les altitudes des points, vous pouvez lui passer une procédure de type TOnGetAltitude pour pouvoir afficher une barre de progression (voir [demoOverlay](#) et [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;

```

Vous pouvez directement créer un TLatLngList et le remplir avec vos points pour obtenir leur altitude sans avoir à passer par une Polyline.

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

Retourne le TLatLng correspondant à Index

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

Calcule la latitude et la longitude d'un point sur le polyline/polygone en fonction de sa distance en mètres, retourne **True** si on a trouvé un point

SensStartEnd sens du parcours, true pour départ -> arrivée

IMeter la distance en mètre

dLatitude,dLongitude des variables de type **double** qui recevront la latitude et la longitude

idPoint une variable qui contiendra l'indice dans le tableau **Point** où se situe le point, le calcul retourne une approximation car votre polyline/polygone ne comporte pas l'ensemble des points réels

Heading une variable qui contiendra l'angle du point par rapport au nord (de 0 à 360°)

bEnd indique si l'on a atteint ou dépassé la fin du polyline/polygone(ou le début suivant le sens)

Les routes disposent d'une même fonction

Exemple de manipulation d'un Polyline

```
// Delphi map component EMap

// 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;
```

Évènements

Les cercles, rectangles, polylines, polygones et groundoverlays répondent aux évènements suivant

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

simple click sur un overlay

index est l'index de l'overlay dans sa liste

dLatitude,dLongitude les coordonnées géographique du click

OverlayType le type d'overlay (ovCircle, ovRectangle, ovLine, ovPolygone,ovLabel ou ovGroundOverlay)

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

le bouton gauche de la souris est enfoncé sur un overlay

index est l'index de l'overlay dans sa liste

dLatitude,dLongitude les coordonnées géographique du click

OverlayType le type d'overlay (ovMarker,ovCircle, ovRectangle, ovLine, ovPolygone,ovLabel ou ovGroundOverlay)

Les markers répondent à cet événement !

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

le bouton gauche de la souris est relevé sur un overlay

index est l'index de l'overlay dans sa liste

dLatitude,dLongitude les coordonnées géographique du click

OverlayType le type d'overlay (ovMarker,ovCircle, ovRectangle, ovLine, ovPolygone,ovLabel

ou `ovGroundOverlay`)

Les markers répondent à cet événement !

CloudMade ne répond pas aux événements `OnOverlayMouseDown` et `OnOverlayMouseUp`

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

Double click sur un overlay

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

Click droit sur un overlay

CloudMade ne supporte pas le click droit, Alt+click déclenche cet événement (sous Google Map aussi)

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

Déclenché lors du déplacement d'un overlay, par code ou à la souris

Sous Google Map vous pouvez déplacer par glisser-lacher l'overlay pointé par `EditOverlay`, mais sous CloudMade vous devez d'abord faire un CTRL+clic sur la carte, puis déplacer la souris en maintenant CTRL enfoncé pour déplacer votre élément (fonctionne aussi avec Google Map)

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

La souris sort d'un overlay

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

La souris survole un overlay

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

Déclenché lors de tout changement sur un overlay (couleur, point...)

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

Déclenché lorsqu'un des points définissant l'overlay change

PathIndex indique l'index dans Path

Marqueurs interactifs

Les marqueurs interactif de l'overlay assigné à EditOverlay déclenchent aussi des événements

OnEditOverlayPointClick(sender:TObject; const Index:integer)

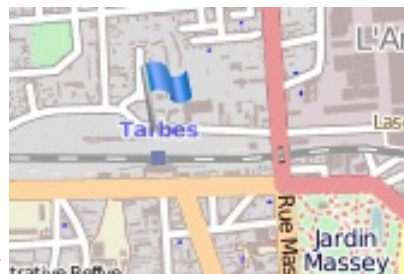
Click sur le marqueur qui représente le point Index dans Path

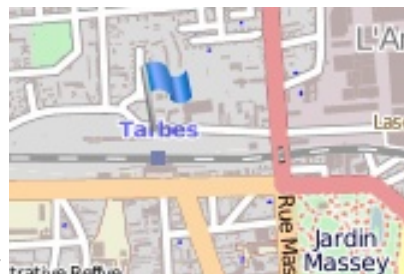
OnEditOverlayPointDragEnd(sender: TObject;const Index:integer;const dLatitude,dLongitude:double)

Déplacement du point Index dans Path et ses nouvelles coordonnées

OnEditOverlayPointRClick(sender:TObject; const Index:integer)

Click sur le marqueur qui représente le point Index dans Path



Par défaut les marqueurs interactif sont représentés par , vous pouvez modifier l'icône au travers de la propriété **IconOverlay** de type string, l'image peut-être une image distante sur internet ou un fichier local.

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

Si vous changez l'icône vous aurez peut-être aussi besoin de retoucher les propriétés

IconSizeOverlay : TPoint

Largeur et Hauteur de l'image

IconOriginOverlay : TPoint

Origine en X et Y de la zone affichée extraite de l'image (une même image peut contenir plusieurs icones)

IconAnchorOverlay : TPoint

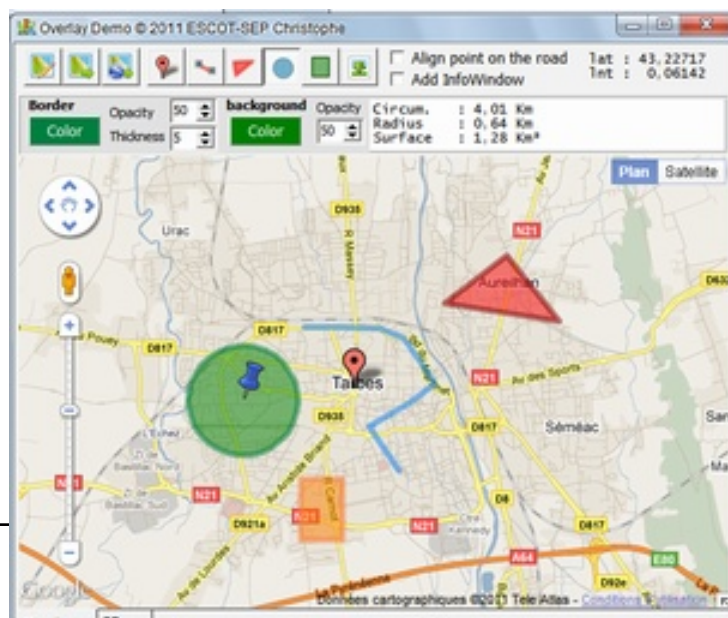
X et Y du point dans la zone qui va correspondre à la latitude et la longitude



Fig. 33 Polygone en mode édition

Démonstration

Le programme DemoOverlay vous montre comment manipuler facilement tous ces éléments



Toutes les notions liées aux overlays y sont employées, la création, l'édition, la suppression, l'[import/export de votre carte](#), y compris l'export vers [Google Earth](#).

Les marqueurs sont gérés par une liste de type **TECMapMarkers** accessible au travers de la propriété **Markers** de TECMap

TECMapMarkers

Cette liste dispose des méthodes et propriétés suivantes :

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

Ajoute un marker au point Latitude,Longitude et retourne son index dans la liste.

Vous pouvez aussi ajouter un marker avec la fonction AddMarker de 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;

Retourne le marker positionné en Latitude,Longitude

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

Retourne le marker en fonction de son nom (propriété Name)

procedure **Clear**;

Efface tous les markers

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

Supprime le marker dont on passe l'index

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

Retourne l'index du marker dans la liste

property **Cluster**:boolean;

Propriété en *lecture/écriture*, regroupe les markers en fonction de leur position et suivant le zoom

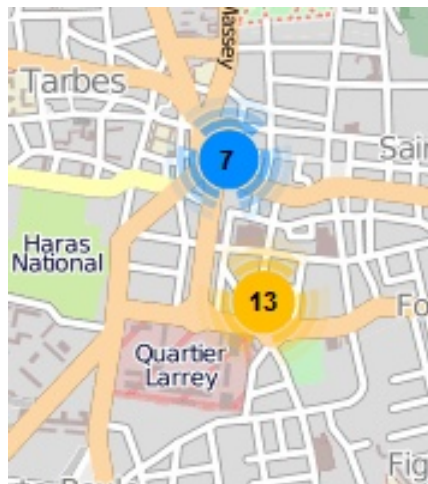


Fig. 35 Activation du regroupement de markers

Inactive sous CloudMade

property **Count**:integer;

Retourne le nombre de marker dans la liste

procédure **fitBounds**

Adapte le zoom de la carte pour afficher l'ensemble des markers

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

Tableau permettant l'accès aux markers, c'est la propriété par défaut donc vous pouvez y accéder directement par **map.Markers[index]** au lieu de **map.Markers.Marker[index]**

property **ToKml** : string;

Propriété en *lecture seule* qui retourne une string au [format Kml](#) contenant la liste des markers

property **ToTxt** : string;

Propriété en *lecture/écriture* qui donne accès à la liste des markers dans un format texte.

*En écriture il s'agit d'un ajout et non d'un remplacement, si vous ne souhaitez pas conserver les anciennes valeurs faites un **Clear** avant l'ajout*

Exemple d'utilisation

```
// 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

Il s'agit de la classe gérant un marker, elle dispose des méthodes et propriétés suivantes :

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

Déplace le marker en Latitude,Longitude, déclenche l'évènement OnMarkerMove

procédure **PanTo**;

Déplace la carte pour que son centre coïncide avec le marker, déclenche l'évènement OnMapMove

Le mouvement sera fluide si le déplacement ne dépasse pas la moitié de la hauteur ou de la largeur de la carte.

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

Copie les options de l'image du marker passé en paramètre

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

_Icon contient le nom du fichier image source, local ou sur internet voir **Icon**

Size définit la Largeur et Hauteur, voir **IconSize**

Origin définit le X et le Y du point d'origine de l'image dans l'image source, voir **IconOrigin**

Anchor définit le X et le Y du point correspondant à la Latitude et Longitude, voir **IconAnchor**

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

Change les options pour l'ombre de l'image

procédure **setShape**;

Application de la zone clickage du marker, voir propriété Shape

property **Adress** : string;

Propriété en **lecture/écriture**, permet d'obtenir l'adresse du marker ou de le positionner à une adresse, déclenche l'évènement OnMarkerAdress

Pour éviter de multiple connexions au serveur l'adresse est mise en cache, une nouvelle connexion ne sera effectuée que si le marker est déplacé

property **Animation** : string;

Propriété en **lecture/écrite**, les valeurs disponibles sont répertoriées dans la propriété de type TStringList **MarkerAnimations** de TECMap

Avec l'api Google vous avez 'BOUNCE' et 'DROP', mettez une chaîne vide pour stopper l'animation

Inactive sous CloudMade

property **Altitude** : double ;

Propriété en **lecture/écrite**, la valeur est mise en cache pour éviter de multiple connexions au serveur, une nouvelle connexion ne sera effectuée que si la position du marker change.

property **Cursor** : string;

Propriété en **lecture/écrite**, défini le curseur de la souris lors du survol, les valeurs disponibles sont répertoriées dans la propriété de type TStringList **Cursors** de TECMap

Inactive sous CloudMade

property **Clickable** : boolean;

Propriété en **lecture/écrite**, défini si le marker reçoit le click de la souris ou non

property **Draggable** : boolean;

Propriété en **lecture/écriture**, défini si le marker est déplaçable à la souris.

Même si un marker n'est pas déplaçable à la souris vous pouvez changer sa position par code au travers de setPosition ou de ses propriétés Latitude et Longitude

property **Dragging** : boolean;

Indique si le marker est entrain d'être déplacé

property **Flat** : boolean;

Propriété en **lecture/écriture**, défini si le marker a une ombre

property **Heading** : integer;

Indique l'angle par rapport au nord

property **Icon** : string;

Propriété en **lecture/écriture**, défini l'image du marker, cela peut-être soit un fichier local soit une url d'un fichier situé sur internet, voir aussi setImage

Avec Google Maps vous pouvez aussi utiliser un

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

property **Shadow** : string;

Propriété en **lecture/écriture**, défini l'image de l'ombre du marker, cela peut-être soit un fichier local soit une url d'un fichier situé sur internet, voir aussi setImageShadow

property **IconSize** : TPoint;

défini la Largeur et Hauteur de l'image

property **IconOrigin** : TPoint;

défini le X et le Y du point d'origine de l'image dans le fichier source

property **IconAnchor** : TPoint;

défini le X et le Y dans l'image du point correspondant à la Latitude et Longitude

property **Index**;

Index du marker dans la liste des markers

property **InfoWindow** : integer;

Index de l'[infoWindow](#) associée au marker dans la liste des InfoWindows

Lors d'un clic sur le marker si une infowindow y est définie, elle sera ouverte

property **Name** : string;

Propriété en *lecture/écrite*, définit le nom de votre marker, voir [ByName](#)

property **Latitude** : double;

Propriété en *lecture/écrite*, définissant la latitude du marker, déclenche l'évènement OnMarkerMove

property **Longitude** : double;

Propriété en *lecture/écrite*, définissant la longitude du marker, déclenche l'évènement OnMarkerMove

property **Tag** : integer;

Propriété en *lecture/écrite*, vous pouvez l'utiliser librement pour stocker ce que vous voulez

property **Title** : string;

Propriété en *lecture/écrite*, définissant le titre de votre marker, il s'affiche sous la forme d'une infobulle lors du survol du curseur

property **Visible** : boolean;

Propriété en *lecture/écrite*, permet d'afficher ou non le marker

property **Zindex** : integer;

Propriété en *lecture/écrite*, attribut un indice de priorité pour l'affichage du marker, un marker ayant un Zindex plus élevé sera affiché par dessus celui ayant un Zindex inférieur

property **ShowOnMap** : boolean;

Propriété en *lecture* indiquant si la position du marker est dans la portion visible de la carte

property **Shape** : TECMapShape;

Permet de définir la zone clickable du 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;
```

Avec SType 'circle', 'poly' ou 'rect'

Voir [aide de google](#)

Inactive sous CloudMade

property **ToTxt** : string;

Propriété en **lecture/écriture** qui donne accès au marker sous la forme d'une chaîne texte

property **ToKml** : string;

Propriété en **lecture seule** qui retourne le marker sous la forme d'une chaîne au [format Kml](#)

Évènements

*Vous pouvez vous connecter sur le composant TECMap pour recevoir les évènements mais vous pouvez aussi vous y brancher directement depuis un marker pour une réponse spécifique, auquel cas **l'évènement global de TECMap ne sera pas appelé***

Les markers déclenchent les évènements:

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

Déclenché lors de tout déplacement d'un marker, par code ou à la souris

Index est l'indice du marker dans la liste Markers

dLatitude et **dLongitude** la nouvelle position

Vous pouvez modifier celle-ci dans l'évènement

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

Déclenché lors d'un click sur le marker

Index est l'indice du marker dans la liste Markers

dLatitude et **dLongitude** position du marker

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

Déclenché lors d'un click droit sur le marker

Index est l'indice du marker dans la liste Markers

dLatitude et **dLongitude** position du marker

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

Déclenché lors d'un double click sur le marker

Index est l'indice du marker dans la liste Markers

dLatitude et **dLongitude** position du marker

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

Déclenché pendant le déplacement à la souris d'un marker

Index est l'indice du marker dans la liste Markers

dLatitude et **dLongitude** position du marker

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

Déclenché au départ du déplacement à la souris d'un marker

Index est l'indice du marker dans la liste Markers

dLatitude et **dLongitude** position du marker

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

Déclenché à la fin du déplacement à la souris d'un marker

Index est l'indice du marker dans la liste Markers

dLatitude et **dLongitude** position du marker

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

Déclenché par l'attribution d'une adresse au marker

sAdresses contiendra toutes les adresses, séparé par #13#10 pouvant correspondre avec votre requête.

Index contiendra l'index de l'adresse retenue, par défaut 0, vous pourrez le modifier dans cet évènement.

IndexMarker indice du marker dans la liste des Markers

En sortie de cet évènement le marker est positionné sur l'adresse retenue

*Les markers répondent aussi aux évènements **OnOverlayMouseDown** et **OnOverlayMouseUp***

Positionner un marker sur un chemin

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

Cette fonction vous permet de positionner un marker sur un chemin (route, polyline ou polygone) en fonction d'une direction et d'une distance en mètre.

iNumMarker est l'indice du marker dans la liste **Markers**

iNumRoute est l'indice du chemin dans **Routes, Polylines ou Polygones**

RouteType détermine le type de chemin, **ovRoute**, **ovLine** ou **ovPolygone**

Direction est le sens de parcour soit **dsStartEnd** soit **dsEndStart**

IMeter la distance en mètre

Dans le chapitre suivant vous découvrirez d'autres fonctionnalités pour [rendre vos markers mobiles](#) plus simplement

Vous pouvez définir une liste de [markers](#) qui auront la faculté de se déplacer le long d'un chemin prédéterminé ([routes](#), [polylines](#) ou [polygones](#)) .

Cette liste de type [TMobileList](#) est disponible au travers de la propriété **Mobiles**.

TECMap dispose en plus de deux propriétés liées aux mobiles

property **MobilesEnabled**:boolean

Propriété en *lecture/écriture* qui autorise ou non le déplacement automatique des mobiles, false par défaut

property **MobilesTiming** : dword

Propriété en *lecture/écriture* qui indique en millisecondes l'intervalle de temps qui sépare chaque déplacement automatique, par défaut **300ms** (l'intervalle minimum étant **100ms**)

TMobileList

Vous avez accès aux méthodes et propriétés suivantes

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

Retourne l'indice du mobile en fonction de l'indice d'un marker

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

Ajoute un mobile

iMarker est l'indice du marker dans la liste [Markers](#)

iRoute est l'indice du chemin à suivre soit dans [Routes](#) (par défaut), soit dans [Polylines](#) ou [Polygones](#)

iSpeed est la vitesse du mobile en **Km/h**

iDistance est la position de départ sur le chemin en **mètre**

```
// 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;

Supprime tous les mobiles

procedure delete(const index:integer);

Efface le mobile d'indice Index

function count:longint;

Retourne le nombre de mobiles

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

Tableau permettant l'accès aux mobiles, c'est la propriété par défaut donc vous pouvez y accéder directement par **map.Mobiles[index]** au lieu de **map.Mobiles.Mobile[index]**

property **Tracking** :integer

Propriété en **lecture/écriture** qui indique quel est le mobile que l'on va suivre, la carte se recentrera automatiquement sur lui lorsqu'il franchira les limites visibles.

Positionnez cette propriété à -1 pour ne suivre aucun mobile

property **ToTxt** : string

Propriété en **lecture/écriture** qui donne accès à la liste des mobiles dans un format texte.

Par défaut les mobiles ne bougent pas, vous pouvez les déplacer soit en mode automatique soit en mode manuel.

En mode automatique ils se déplaceront à la vitesse que vous leur avez assigné, en mode manuel vous pourrez soit manipuler directement la distance soit la faire se calculer en fonction du temps et de la vitesse.

Même en mode automatique vous pouvez modifier manuellement la distance, la vitesse, vous pouvez aussi changer la route sur laquelle le mobile se déplace, sa position est automatiquement réajusté.

Pour cela vous devez accéder à la classe TMobile

TMobile

function **Move**:integer;

Déplace le mobile en calculant sa distance depuis le point de départ en fonction du temps passé depuis le dernier déplacement, de la vitesse et de la direction.

property **Distance** : integer;

Propriété en **lecture/écriture** qui fixe la distance, en mètre, du mobile depuis son point de départ réel, la direction n'est pas prise en compte.

Tout changement de cette propriété entraine un déplacement du marker associé au mobile

property **Direction**: TDirectionSens;

Sens de déplacement, **dsStartEnd** ou **dsEndStart**

Pour une route dsStartEnd correspond à un déplacement du point de départ ou point d'arrivée, pour les polygones et polygones du premier point au dernier.

La distance est toujours calculée en fonction du point de départ réel, la direction ne va donc être déterminante que pour un déplacement automatique ou manuel par appel à **Move**.

property **Index** : integer

Indice du mobile dans la liste des mobiles

property **Marker** : integer

Propriété en *lecture/écriture* qui indique l'indice du **Marker** à déplacer

property **Mobile** : boolean

Propriété en *lecture/écriture* qui définit si le mobile se déplace automatiquement ou non, par défaut fixé à **false** le mobile ne se déplace pas tout seul.

property **Route** : integer;

Propriété en *lecture/écriture* qui fixe l'indice du chemin dans sa liste respective (**Routes**, **Polygones** ou **Polygones** en fonction de **RouteType**)

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

Propriété en *lecture/écriture* définit le type de chemin, **ovRoute**, **ovLine** ou **ovPolygone**

property **Speed** : integer

Propriété en *lecture/écriture* qui détermine la vitesse en km/h

property **Step** : integer;

Indice dans la propriété **Path** de la route, polyline ou polygone où se situe le mobile, le calcul retourne une approximation du point car votre route ne comporte pas l'ensemble des points réels.

property **Heading** : integer;

Angle du mobile en degrés par rapport au Nord (0° à 360°)

property **ToTxt** : string ;

Propriété en *lecture/écriture* qui donne accès au mobile dans un format texte.

Évènements

Deux évènements sont liés aux mobiles

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

Déclenché lors de tout déplacement d'un marker, par code ou à la souris

Index est l'indice du marker dans la liste **Markers**

dLatitude et **dLongitude** la nouvelle position

Vous pouvez modifier celle-ci dans l'évènement

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

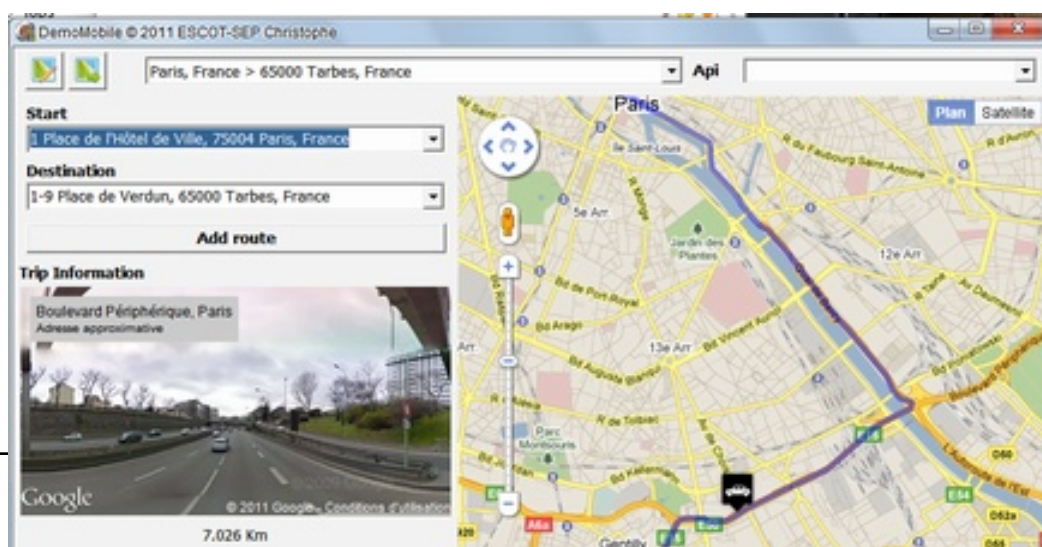
iMobile indice du mobile dans la liste **Mobiles**

En fonction de la direction le mobile est soit arrivé au terminus soit au point de départ

La propriété **Mobile** est passée à **false**, pour relancer votre mobile en mode automatique il vous faut la remettre à **true** et inverser la direction.

Démonstration

Le programme **DemoMobile** vous montre une utilisation des mobiles



Une InfoWindow est une bulle d'information pouvant contenir du texte HTML, elles sont accessibles au travers de la liste InfoWindows de type `TECMapInfoWindows`

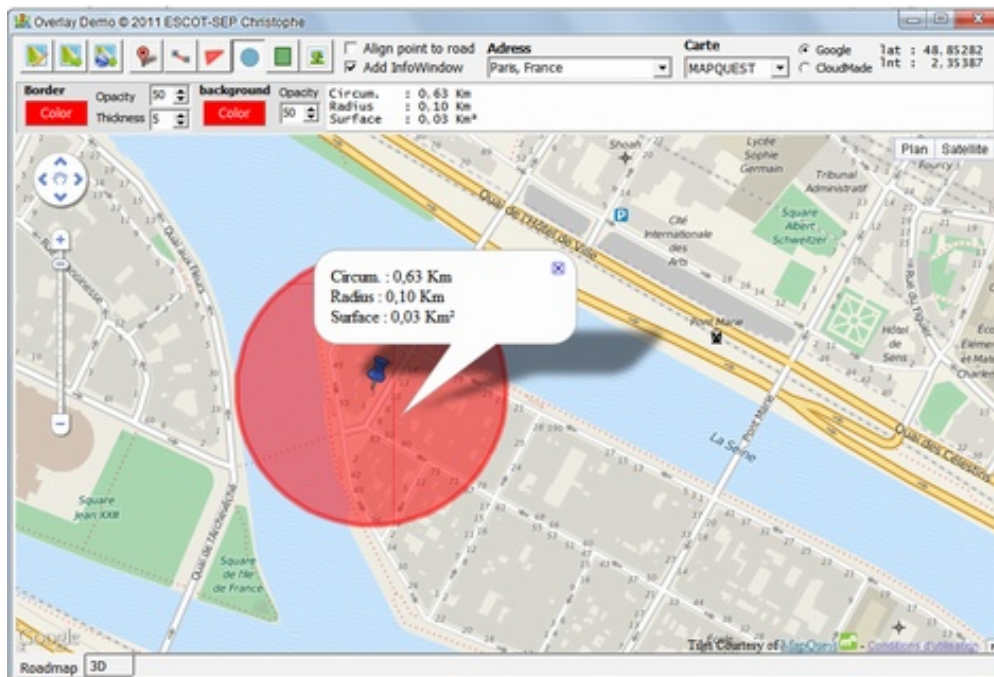


Fig. 37 Une infowindow associée à un cercle

TECMapInfoWindows

Cette classe dispose des méthodes et propriétés

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

Ajoute une infowindow

le contenu peut-être du texte brut ou du Html

*procedur***Clear**;

Efface toutes les infowindows

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

Supprime l'infowindow dont on passe l'index

property **Count**:integer;

Retourne le nombre de marker dans la liste

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

Tableau permettant l'accès aux infowindows, c'est la propriété par défaut donc vous pouvez y accéder directement par **map.InfoWindows[index]** au lieu de **map.InfoWindows.InfoWindow[index]**

property **ToKml** : string;

Propriété en *lecture seule* qui retourne une string au *format Kml* contenant la liste des infowindows

property **ToTxt** : string;

Propriété en *lecture/écriture* qui donne accès à la liste des infowindows dans un format texte.

*En écriture il s'agit d'un ajout et non d'un remplacement, si vous ne souhaitez pas conserver les anciennes valeurs faites un **Clear** avant l'ajout*

TECMapInfoWindow

Les infowindows disposent des méthodes et propriétés

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

Déplace l'infowindow en Latitude,Longitude

Non disponible sous CloudMade

property **Anchor** : integer;

Propriété en **lecture/écrite** qui associe une infowindow à un [marker](#)

Sous CloudMade les infowindows doivent obligatoirement être associées à un marker pour pouvoir s'afficher

property **Content** : string;

Propriété en **lecture/écrite** définissant le contenu de l'infowindow

property **Latitude** : double;

Propriété en **lecture/écrite** donnant accès à la latitude

Non disponible sous CloudMade

property **Longitude**: double;

Propriété en **lecture/écrite** donnant accès à la longitude

Non disponible sous CloudMade

property **MaxWidth** : integer;

Propriété en **lecture/écrite** permettant de définir la largeur maximale que peut atteindre l'infowindow

Non disponible sous CloudMade

property **Open** : boolean

Propriété en **lecture/écrite** permettant d'ouvrir/fermer une infowindow

La fermeture d'une infowindow déclenche l'évènement **OnCloseInfoWindow**

property **Zindex** : integer;

Propriété en **lecture/écrite**, attribut un indice de priorité pour l'affichage de l'infowindow, une infowindow ayant un Zindex plus élevé sera affichée par dessus un Zindex inférieur

property **Index** : integer;

Indice de l'infowindow dans sa liste

property **ToKml** : string;

Propriété en **lecture seule** qui retourne une string au [format Kml](#) contenant l'infowindow

property **ToTxt** : string;

Propriété en **lecture/écriture** qui donne accès à l'infowindow dans un format texte.

OnCloseInfoWindow

Évènement déclenché à la fermeture d'une infowindow

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

Index indice de l'infowindow dans la liste [InfoWindows](#)

Différence entre API

Sous CloudMade pour être affichée une infowindow doit obligatoirement être associée à un [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;
```

Alors que sous Google Maps vous pouvez directement ouvrir une infowindow sans l'associer à un marker

Non disponible sous CloudMade

Les labels sont des conteneurs Html pouvant être déplacé à la souris, ils sont manipulables au travers de la [liste Labels](#).

```
// 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;
```

Ils disposent des propriétés et méthodes

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

Déplace le label en Latitude,Longitude, déclenche l'évènement OnOverlayMove

property **Content** : string

Propriétés en **lecture/écriture** décrivant le contenu HTML du label

property **Css** : string;

Propriétés en **lecture/écriture** styles CSS applicable au label

property **Draggable** : boolean

Propriété en **lecture/écriture**, défini si le label est déplaçable à la souris.

Même si un label n'est pas déplaçable à la souris vous pouvez changer sa position par code au travers de setPosition ou de ses propriétés Latitude et Longitude

property **Id** : integer

Numéro d'index du label dans sa liste

Propriété en **lecture/écriture**, définissant la latitude du label, déclenche l'évènement OnOverlayMove

property **Longitude** : double;

Propriété en **lecture/écriture**, définissant la longitude du label, déclenche l'évènement OnOverlayMove

property **Visible** : boolean;

Propriété en **lecture/écriture**, permet d'afficher ou non le label

property **Zindex** : integer;

Propriété en **lecture/écriture**, attribut un indice de priorité pour l'affichage du label, un label ayant un Zindex plus élevé sera affiché par dessus celui ayant un Zindex inférieur

property **ToTxt** : string;

Propriété en **lecture/écriture** qui donne accès au label sous la forme d'une chaîne texte

```
// 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;';
```

Évènements

Les labels répondent aux évènements:

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

simple click sur un label

index est l'index du label dans sa liste

dLatitude,dLongitude les coordonnées géographique du click

OverlayType le type d'overlay (ovCircle, ovRectangle, ovLine, ovPolygone,ovLabel ou ovGroundOverlay), donc **ovLabel** pour un label

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

Déclenché lors du déplacement d'un label, par code ou à la souris

OnLink(sender: TObject;const Url:string)

Si votre label contient un lien vous interceptez le click dans cet évènement

Url contient la cible du lien

Les routes sont gérées par la liste **Routes** de type **TECMapRoutes**

TECMapRoutes

Cette liste dispose des méthodes et propriétés suivantes :

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

Ajout d'une route, retourne l'indice de la route dans la liste

Déclenche l'évènement [OnRouteChange](#) en cas de succès sinon OnRouteError

Name nom de votre route, peut-être vide

dStartLatitude,dStartLongitude point de départ de la route

dEndLatitude, dEndLongitude point d'arrivée de la route

OptimizeWayPoints force le calcul de la route la plus directe en réorganisant vos points de passages (**n'a aucune action sous CloudMade**)

WayPoints liste de type [TLatLngList](#) des points de passages.

Passez nil si vous ne voulez pas indiquer de points de passage particulier

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

Ajout d'une route, retourne l'indice de la route dans la liste

Déclenche l'évènement [OnRouteChange](#) en cas de succès sinon OnRouteError

Name nom de votre route, peut-être vide

sStartAdress Adresse du point de départ de la route

sEndAdress Adresse point d'arrivée de la route

OptimizeWayPoints force le calcul de la route la plus directe en réorganisant vos points de passages (**n'a aucune action sous CloudMade**)

WayPoints liste de type [TLatLngList](#) des points de passages.

Passez nil si vous ne voulez pas indiquer de points de passage particulier

AddByAdr n'est pas disponible sous CloudMade

procedure **Clear**;

Efface l'ensemble des routes

function **Count**:integer;

Retourne le nombre de routes

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

Efface la route d'indice **Index**

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

Annule le changement de direction effectuée à la souris sur la route d'indice **Index**
Vous pouvez annuler les dix dernières modifications de trajet.

Non disponible sous CloudMade

function **Ready**:boolean;

Indique si vous pouvez ajouter une route, vous ne pouvez pas le faire lorsqu'une route est entrain d'être calculée.

property **TravelMode**: TDirectionsTravelMode

Propriété en **lecture/écriture** qui permet de choisir le type de route que l'on veut

property **Distance** : longint;

Propriété en **lecture seule** qui retourne la longueur de l'ensemble des routes en **mètres**

property **Duration** : longint;

Propriété en **lecture seule** qui retourne le temps qu'il faut, en **secondes**, pour effectuer les trajets.

Non disponible sous CloudMade

property **avoidHighways** : boolean;

Propriété en **lecture/écrite** qui offre le choix d'éviter ou non les autoroutes

property **avoidTolls** : boolean;

Propriété en **lecture/écrite** qui offre le choix d'éviter ou non les routes payantes

property **Color** : TColor;

Propriété en **lecture/écrite** qui offre le choix de fixer la couleur de la route

property **Draggable** : boolean;

Propriété en **lecture/écrite** qui permet la modification à la souris du trajet de la route.

Avec l'api CloudMade vous ne pouvez modifier que les points de départ/arrivée et de passages

property **Opacity** : double;

Propriété en **lecture/écrite** qui ajuste le pourcentage d'opacité de la couleur de la route

property **Weight** : integer;

Propriété en **lecture/écrite** qui fixe la taille du tracé de la route

property **ZIndex** : integer;

Propriété en **lecture/écrite**, attribut un indice de priorité pour l'affichage de la route, une route ayant un Zindex plus élevé sera affiché par dessus celle ayant un Zindex inférieur

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

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

Tableau permettant l'accès aux routes, c'est la propriété par défaut donc vous pouvez y accéder directement par **map.Routes[index]** au lieu de **map.Routes.Route[index]**

property **ToKml** : string;

Propriété en **lecture seule** qui retourne une string au [format Kml](#) contenant la liste des routes

property **ToTxt** : string;

Propriété en **lecture/écriture** qui donne accès à la liste des routes dans un format texte.

*En écriture il s'agit d'un ajout et non d'un remplacement, si vous ne souhaitez pas conserver les anciennes valeurs faites un **Clear** avant l'ajout*

```

        // 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;

```

Évènements

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

idRoute est l'indice de la route dans la liste **Routes**

NewRoute égale **True** s'il s'agit d'un ajout de route, **False** s'il s'agit d'une modification du trajet

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

idRoute est l'indice de la route dans la liste **Routes**

sError explication de l'erreur

TECMapRoute

TECMapRoute est la classe qui manipule une route, elle vous donne accès aux méthodes et propriétés suivantes :

```

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

```


Calcule la latitude et la longitude d'un point sur la route en fonction de sa distance en mètres, retourne **True** si on a trouvé un point

SensStartEnd sens du parcours, true pour départ -> arrivée

IMeter la distance en mètre

dLatitude,dLongitude des variables de type **double** qui recevront la latitude et la longitude

idPoint une variable qui contiendra l'indice dans **Path** où se situe le point, le calcul retourne une approximation car votre route ne comporte pas l'ensemble des points réels

*Les Polylines/Polygones disposent d'une même fonction au travers de leur propriété **Path** de type **TLatLngList***

Heading une variable qui contiendra l'angle du point par rapport au nord (de 0 à 360°)

bEnd indique si l'on a atteint ou dépassé la fin de la route (ou le début suivant le sens)

procédure **fitBounds**

Adapte le zoom de la carte pour afficher l'ensemble de la route

procédure **Update**;

Redéfinit la route pour tenir compte des modifications effectuées sur l'une des propriétés

procédure **OptimizeWaypoints**;

Réorganise les points de passages pour que leur ordre soit optimal

Non disponible sous CloudMade

procédure **updateOptions**;

Redessine la route pour tenir compte des propriétés **Color,Opacity,Weight** et **Draggable**

Propriété en **lecture seule** qui retourne la longueur de la route en **mètres**

property **Duration** : longint;

Propriété en **lecture seule** qui retourne le temps qu'il faut, en **secondes**, pour effectuer le trajet.

Non disponible sous CloudMade

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

Liste de type [TECMapRoutePath](#) contenant les points constituant la route

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

Liste des points de passages

property **Copyright** : string;

L'éventuel copyright associé à cette route

property **StartAddress** : string;

Adresse du point de départ, en *lecture seule*

 *Non disponible sous CloudMade*

property **StartLatitude** : double;

Propriété en *lecture/écriture* Latitude du point de départ

property **StartLongitude** : double;

Propriété en *lecture/écriture* Longitude du point de départ

property **EndAddress** : string;

Adresse du point d'arrivée, en *lecture seule*

 *Non disponible sous CloudMade*

property **EndLatitude** : double;

Propriété en *lecture/écriture* Latitude du point d'arrivée

property **endLongitude** : double;

Propriété en *lecture/écriture* Longitude du point d'arrivée

property **Name** : string;

Propriété en *lecture seule* qui retourne le nom de la route tel qu'il a été indiqué à la création

property **Id** : integer;

Propriété en **lecture seule** qui retourne l'indice de la route dans la liste [Routes](#)

property **NorthEastLatitude** : double;

Propriété en **lecture seule** qui retourne la latitude du coin Nord-Est de la zone encadrant l'intégralité de la route

property **NorthEastLongitude** : double;

Propriété en **lecture seule** qui retourne la longitude du coin Nord-Est de la zone encadrant l'intégralité de la route

property **SouthWestLatitude** : double;

Propriété en **lecture seule** qui retourne la latitude du coin Sud-Ouest de la zone encadrant l'intégralité de la route

property **SouthWestLongitude** : double;

Propriété en **lecture seule** qui retourne la longitude du coin Sud-Ouest de la zone encadrant l'intégralité de la route

property **TravelMode**: TDirectionsTravelMode

Propriété en **lecture/écriture** qui permet de choisir le type de route que l'on veut

property **avoidHighways** : boolean;

Propriété en **lecture/écriture** qui offre le choix d'éviter ou non les autoroutes

property **avoidTolls** : boolean;

Propriété en **lecture/écriture** qui offre le choix d'éviter ou non les routes payantes

property **Color** : TColor;

Propriété en **lecture/écriture** qui offre le choix de fixer la couleur de la route

property **Draggable** : boolean;

Propriété en **lecture/écriture** qui permet la modification à la souris du trajet de la route.

Avec l'api CloudMade vous ne pouvez modifier que les points de départ/arrivée et de passages

property **Opacity** : double;

Propriété en **lecture/écriture** qui ajuste le pourcentage d'opacité de la couleur de la route

property **Weight** : integer;

Propriété en **lecture/écrite** qui fixe la taille du tracé de la route

property **ZIndex** : integer;

Propriété en **lecture/écrite**, attribut un indice de priorité pour l'affichage de la route, une route ayant un Zindex plus élevé sera affiché par dessus celle ayant un Zindex inférieur

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

Propriété qui vous permet de manipuler les markers de départ et d'arrivée en changeant l'icône par exemple, malheureusement on ne peut pas encore les différencier.

Non disponible sous CloudMade

property **Tag** :integer;

Propriété en **lecture/écrite**, vous pouvez l'utiliser librement pour stoker ce que vous voulez

property **ToKml** : string;

Propriété en **lecture seule** qui retourne une string au [format Kml](#) contenant la route

property **ToTxt** : string;

Propriété en **lecture/écriture** qui donne accès à la route dans un format texte.

*En écriture il s'agit d'un ajout et non d'un remplacement, si vous ne souhaitez pas conserver les anciennes valeurs faites un **Clear** avant l'ajout*

TECMapRoutePath

Liste des points de la route, elle vous donne accès aux méthodes et propriétés

function **Distance**:integer;

Distance en mètres

function **Duration**:integer;

Durée en secondes

function **Count**:integer;

Retourne le nombre de point constituant la route

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

Retourne l'index d'un point dont on passe la latitude et la longitude, -1 si pas de point

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

Tableau permettant l'accès aux points, c'est la propriété par défaut donc vous pouvez y accéder directement par **map.Routes[idroute].Path[idPoint]** au lieu de **map.Routes[idroute].Path.Point[idPoint]**

TECMapRoutePathPoint

Classe gérant un point de la route, elle vous donne accès aux propriétés

property **Altitude** : double;

Propriété en *lecture seule* qui retourne l'altitude en mètre du point

property **Instructions** : string;

Propriété en *lecture seule* qui retourne les instructions, en **HTML**, associées au point

property **Latitude** : double;

Propriété en *lecture seule* qui retourne la latitude du point

property **Longitude** : double;

Propriété en *lecture seule* qui retourne la longitude du point

property **Distance** : longint;

Propriété en *lecture seule* qui retourne la distance en mètres

property **Text** : string;

Propriété en *lecture seule* qui retourne les instructions au format texte

AddPolylineFromRoute

TECMap dispose d'une fonction qui vous permet d'ajouter une [polyligne](#) à partir de l'indice d'une route

```
// Delphi map component EMap  
  
// add polyline with info of route number 0  
id := map.AddPolylineFromRoute(0);
```

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

Les polygones permettent de [calculer par lot l'altitude](#) des points alors que pour les routes il faut le faire point par point, DemoRoute utilise AddPolylineFromRoute pour calculer l'élévation.

Fonctions utilitaires

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);

Ces quatre routines permettent d'obtenir les informations d'une route sans effectuer de tracé, vous avez deux procedures asynchrones qui vont s'exécuter en arrière-plan et déclencher l'événement OnRoutePath lorsque les données sont disponibles.

Voir open.mapquestapi.com/directions/ pour le paramètre Params

Si vous ne libérez pas le TECMapRoutePath obtenu, il le sera automatiquement lors de la destruction de TECMap

Vous pouvez créer une polyline à partir d'un 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;

Retourne la distance en KM entre 2 points géographique, utilise la [formule d'haversine](#)

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

Retourne la distance en KM entre 2 adresses en passant par la route

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

Retourne la distance en KM entre 2 points géographique en passant par la route

Voir open.mapquestapi.com/directions/ pour le paramètre Params des fonctions DistanceRouteXXX

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

Ajoute une route, **Adresses** contient les adresses de 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

Le programme DemoRoute vous montre comment exploiter les routes

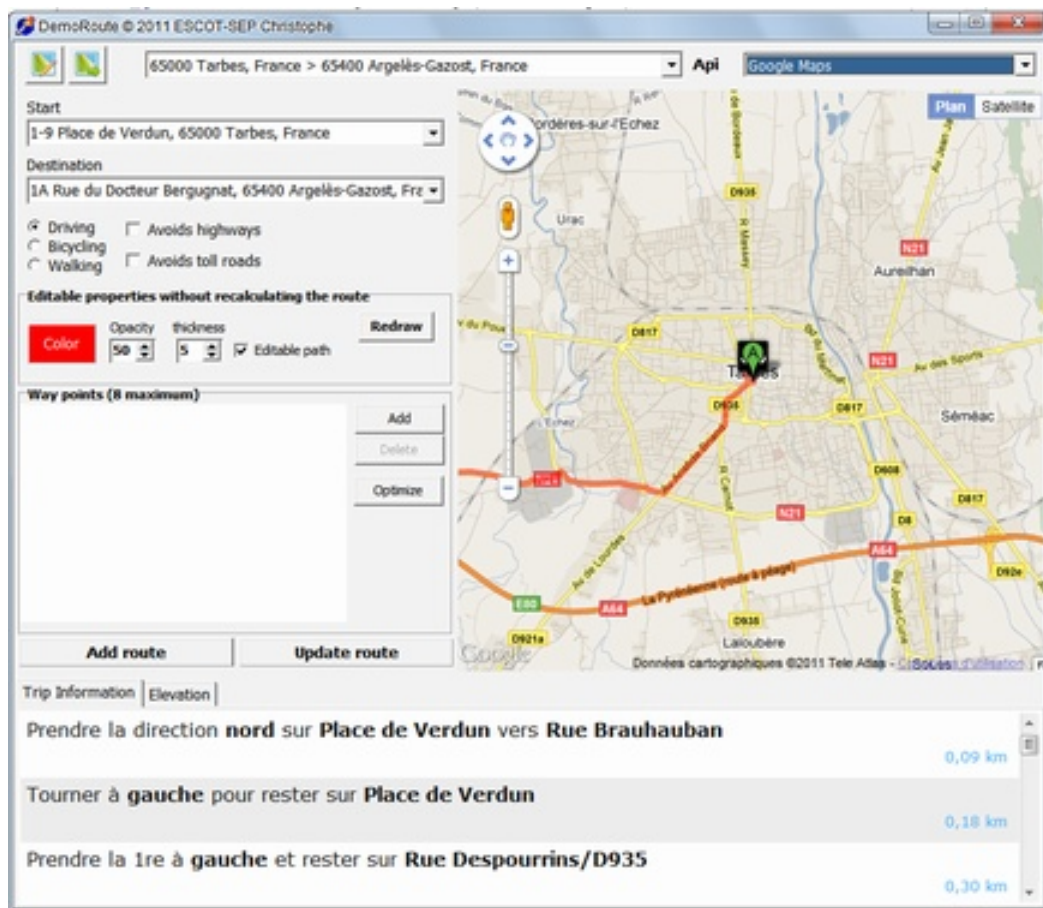


Fig. 38 DemoRoute - Instructions

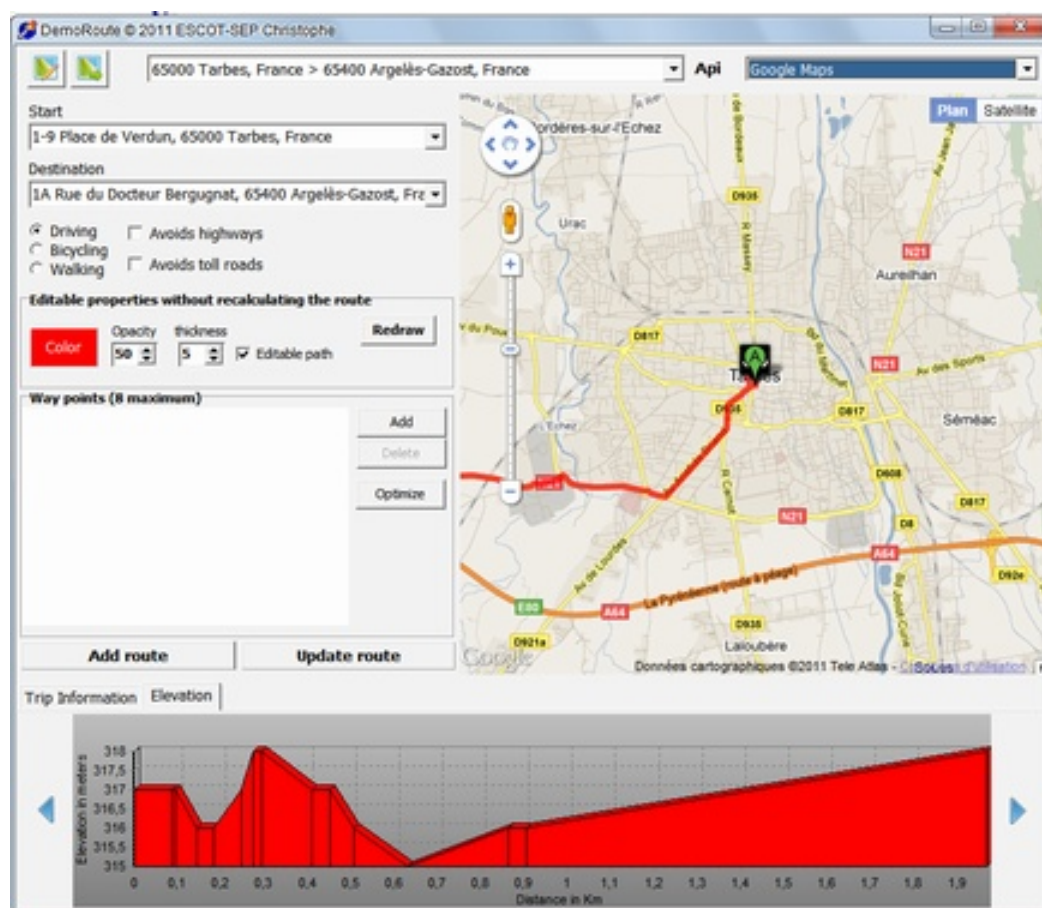


Fig. 39 DemoRoute - Altitudes

BicyclingLayer

Affiche les pistes cyclables uniquement avec l'api Google

```
// Delphi map component EMap  
  
// show bicyclinglayer  
map.BicyclingLayer := true;
```

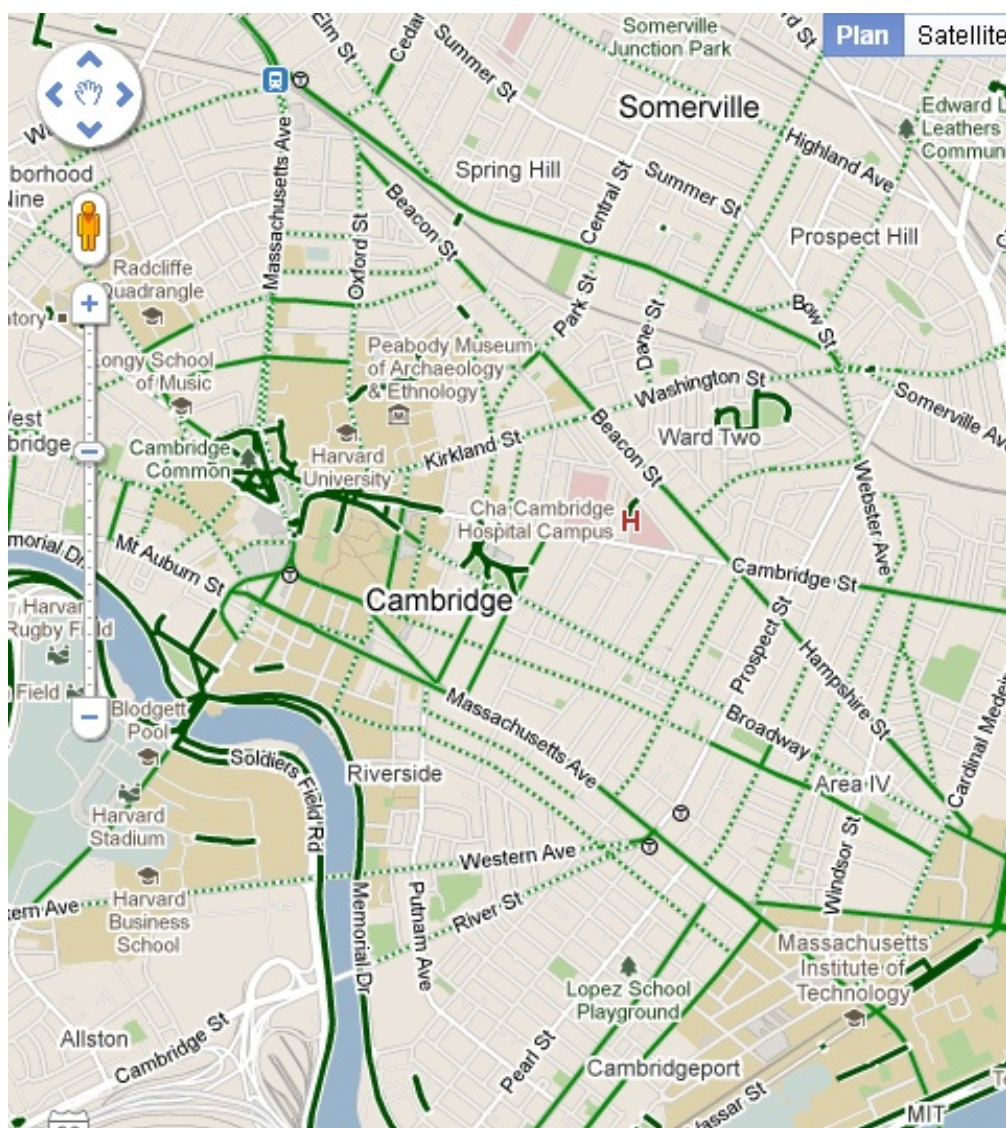


Fig. 40 BicyclingLayer

Pour le moment le layer n'est pas très développé surtout en europe, vous pouvez le remplacer par une carte OpenMapStreet CycleMap qui à l'avantage de fonctionner avec toutes les apis.

```
// Delphi map component EMap

// show bicyclinglayer
map.BicyclingLayer := true;
```

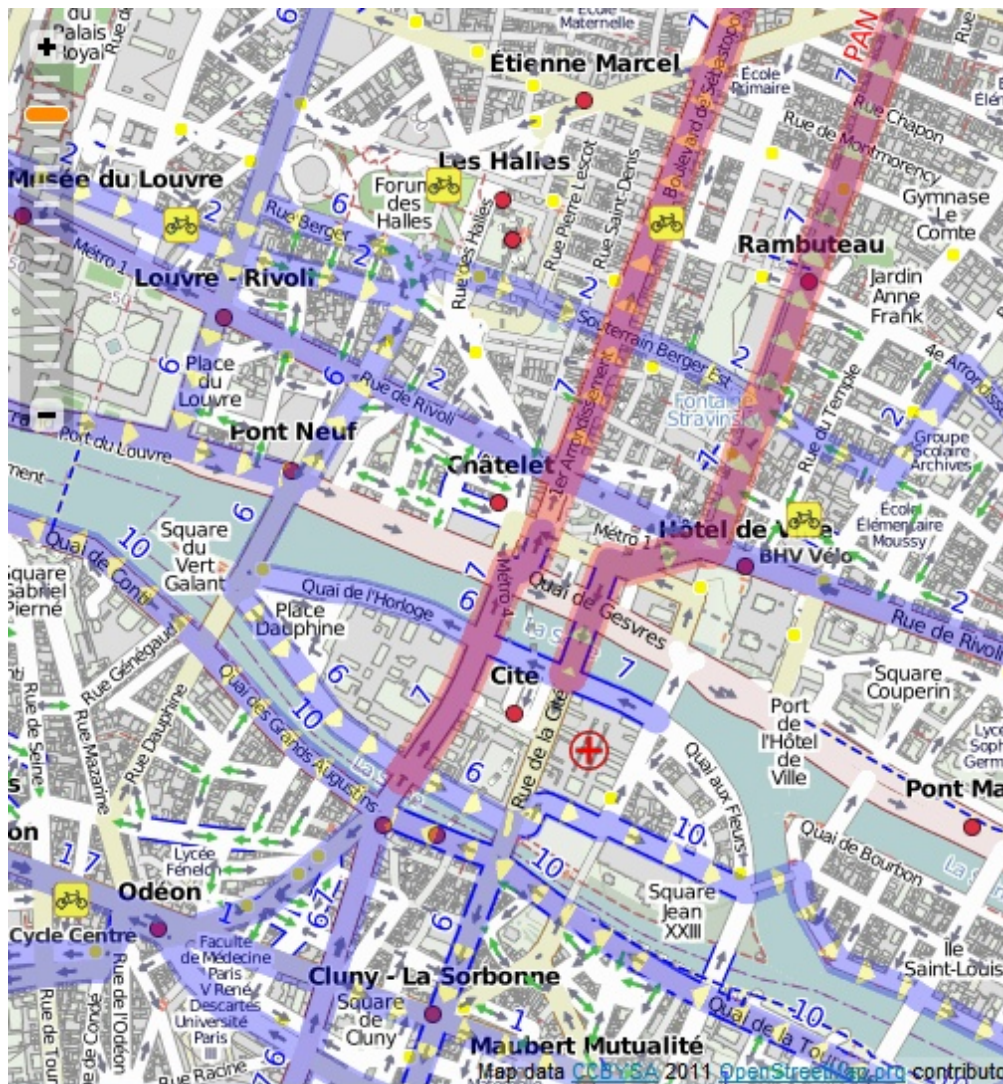


Fig. 41 OpenMapStreet CycleMap

TrafficLayer

Affiche l'état du trafic, ne semble pas activé sur toutes les régions, uniquement avec l'api Google

```
// Delphi map component EMap  
  
// show bicyclinglayer  
map.BicyclingLayer := true;
```

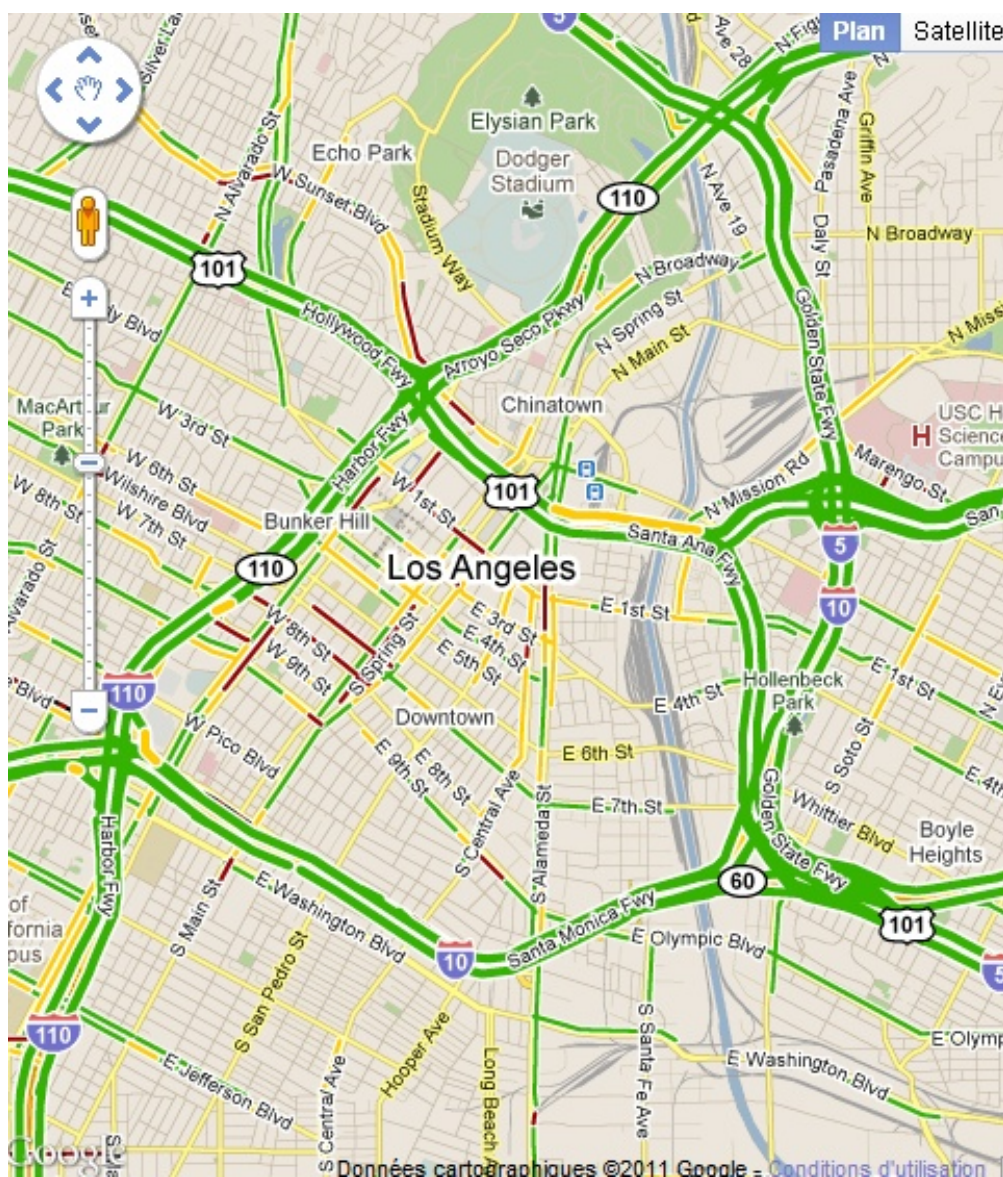


Fig. 42 TrafficLayer

TrafficArea

La procédure **TrafficArea**(const dLatSW,dLngSW,dLatNE,dLngNE:double) détermine les incidents ayant lieux dans une zone donnée.

Vous devez vous brancher sur l'événement **OnTraffic** du composant pour utiliser la réponse

```
// Delphi map component EMap  
  
// show bicyclinglayer  
map.BicyclingLayer := true;
```

Les propriétés **TrafficIconLowImpact**, **TrafficIconMinor**, **TrafficIconModerate**, **TrafficIconSerious** et **TrafficIconRoadClosed** vous permettront d'utiliser vos propres icônes.

TrafficArea utilise les services de [Bing Traffic](#), vous devez [obtenir votre propre clef Bing](#) pour utiliser cette fonction.

FusionTablesLayer

Affiche des données depuis une [table Google Fusion](#) uniquement avec l'api Google

Vous gérer vos layers au travers de la propriété **FusionTablesLayers** de type **TECMapFusionTablesLayers**

```
// Delphi map component EMap  
  
// show bicyclinglayer  
map.BicyclingLayer := true;
```

TECMapFusionTablesLayers

Voir la [documentation de google sur FusionTablesLayer](#)

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

Ajoute un layer et retourne son index

Column est la colonne de la table, correspond à **select** dans la doc google

Id est l'identifiant de la table, correspond à **from** dans la doc google

Style est le style applicable au layer

Clause est la requête SQL qui vous permet de filtrer vos données, correspond à **where** dans la doc google

procedure **Clear**;

Efface l'ensemble des layers

function **Count**:integer;

Retourne le nombre de layers

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

Supprime le layer dont on passe l'index

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

Tableau permettant l'accès aux layers, c'est la propriété par défaut donc vous pouvez y accéder directement par **map.FusionTablesLayers[index]** au lieu de **map.FusionTablesLayers.FusionTablesLayer[index]**

property **ToTxt**: string ;

Propriété en **lecture/écriture** qui donne accès à la liste des FusionTablesLayer dans un format texte.

*En écriture il s'agit d'un ajout et non d'un remplacement, si vous ne souhaitez pas conserver les anciennes valeurs faites un **Clear** avant l'ajout*

TECMapFusionTablesLayer

Classe contrôlant un FusionTablesLayer

procedure **update**;

Mise à jour des propriétés, vous devez y faire appel pour répercuter les modifications des propriétés

property **TableId** : string

identifiant de la table, correspond à **from** dans la doc google

property **LocationColumn** : string

colonne de la table, correspond à **select** dans la doc google

property **Clause** : string

Requête SQL qui vous permet de filtrer vos données, correspond à **where** dans la doc google

property **Style** : string

style applicable au layer

property **Heat** : boolean

correspond à **heatmap** dans la doc google

property **SuppressInfoWindow** : boolean

Permet d'activer ou non l'affichage d'une infoWindow lors d'un clic sur un élément du layer

property **Visible** : boolean

Affiche ou non le layer

property **ToTxt** : string

Propriété en **lecture/écriture** qui donne accès au layer dans un format texte.

OnFusionTablesLayerClick

TECMap dispose d'un évènement pour intercepter un click sur un élément du layer.

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

idLayer le numéro du layer dans la liste FusionTablesLayers

htmlInfoWindow une chaîne contenant le contenu Html de l'info bulle de l'élément, s'il existe

dLatitude,dLongitude coordonnées du point cliqué

KmlLayer

Vous pouvez importer des fichiers Kml/Kmz dans vos cartes et les afficher en surcouche, ne fonctionne pas sous Leaflet.

*Les fichiers Kmz ne sont pas supportés par CloudMade,
par contre il supporte le format GeoXml (extension .xml)*

Les layers Kml disposent des propriétés et méthodes

Id : integer

Index du layer, accessible en lecture seule

Url : string

Url du fichier kml, accessible en lecture seule

Visible : boolean

Visibilité du layer, accessible en lecture/écriture

ToTxt : string

propriété en lecture/écriture permettant d'exporter/importer au format texte l'ensemble des éléments

procédure **Show**;

Affiche la zone par défaut du layer si elle a été spécifiée

OnKmlLayerClick

TECMap dispose d'un évènement pour intercepter un click sur un élément du layer.

CloudMade ne répond pas à cet évènement

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

idKmlLayer le numéro du layer dans la liste KmlLayers

htmlDescription une chaîne contenant le contenu Html de l'élément, s'il existe

htmlInfoWindow une chaîne contenant le contenu Html de l'info bulle de l'élément, s'il existe

dLatitude,dLongitude coordonnées du point cliqué

```
// Delphi map component EMap
// show bicyclinglayer
map.BicyclingLayer := true;
```

TECMap permet d'exporter la majorité de ses données, y compris les layers kml, au format KML.

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

```
// show bicyclinglayer  
map.BicyclingLayer := true;
```

WeatherLayer

Disponible uniquement avec l'api Google

```
// Delphi map component EMap  
  
// show bicyclinglayer  
map.BicyclingLayer := true;
```

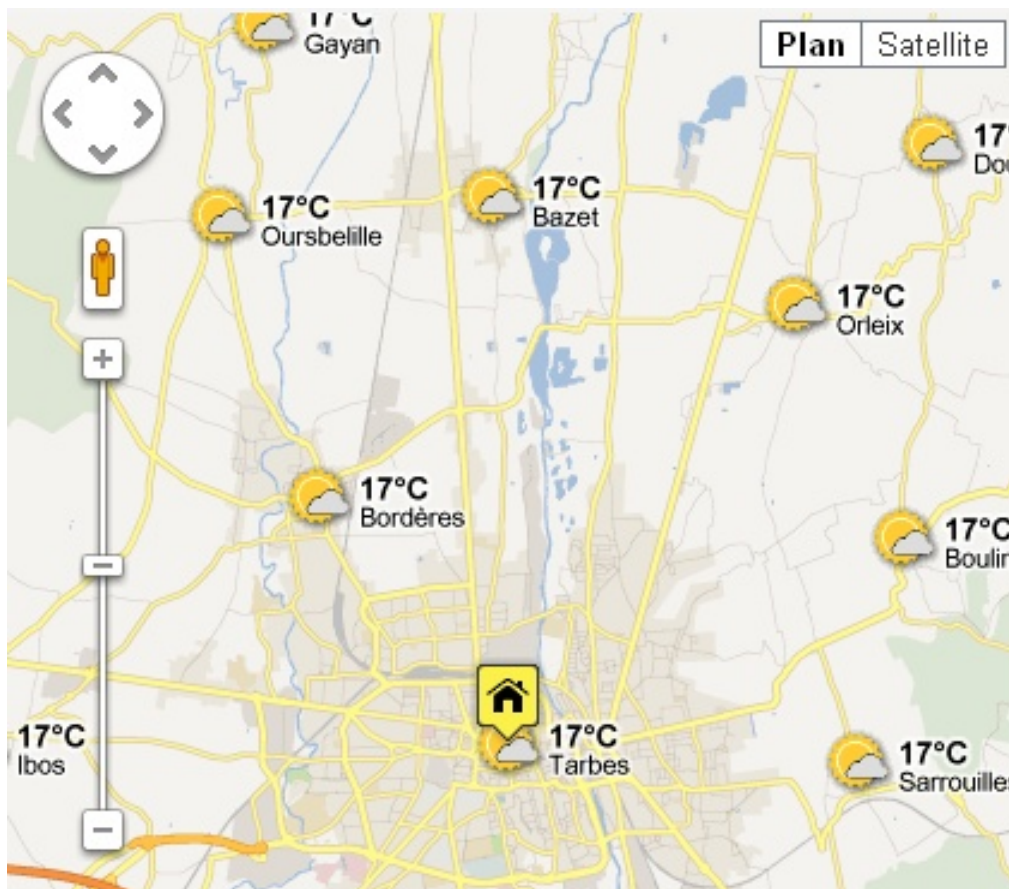


Fig. 43 WeatherLayer

OnWeatherLayerClick

TECMap dispose d'un évènement pour intercepter un click sur un élément du layer.

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

htmlInfoWindow une chaîne contenant le contenu Html de l'info bulle de l'élément, s'il existe

dLatitude,dLongitude coordonnées du point cliqué

DemoLayer

Le programme **DemoLayer** vous permet de voir comment utiliser ces différents layers ainsi que [Panoramio](#) qui est lui aussi une surcouche

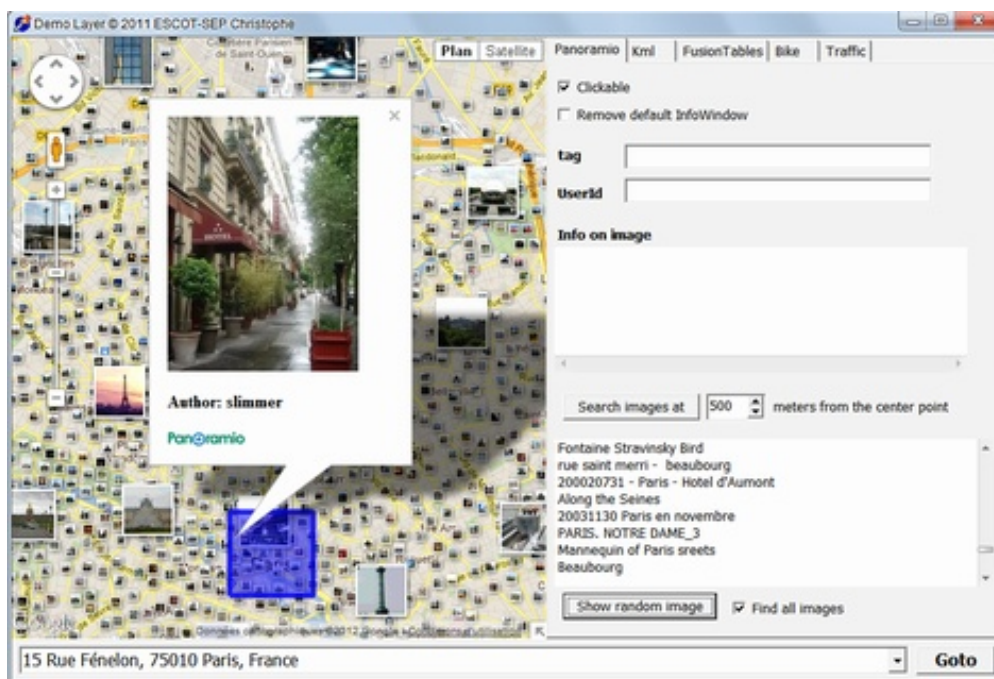


Fig. 44 DemoLayer

Vous avez 4 formes prédéfinies (Ellipse, rectangle, triangle et étoile) mais vous pouvez aussi prendre en charge vous même leur dessin.

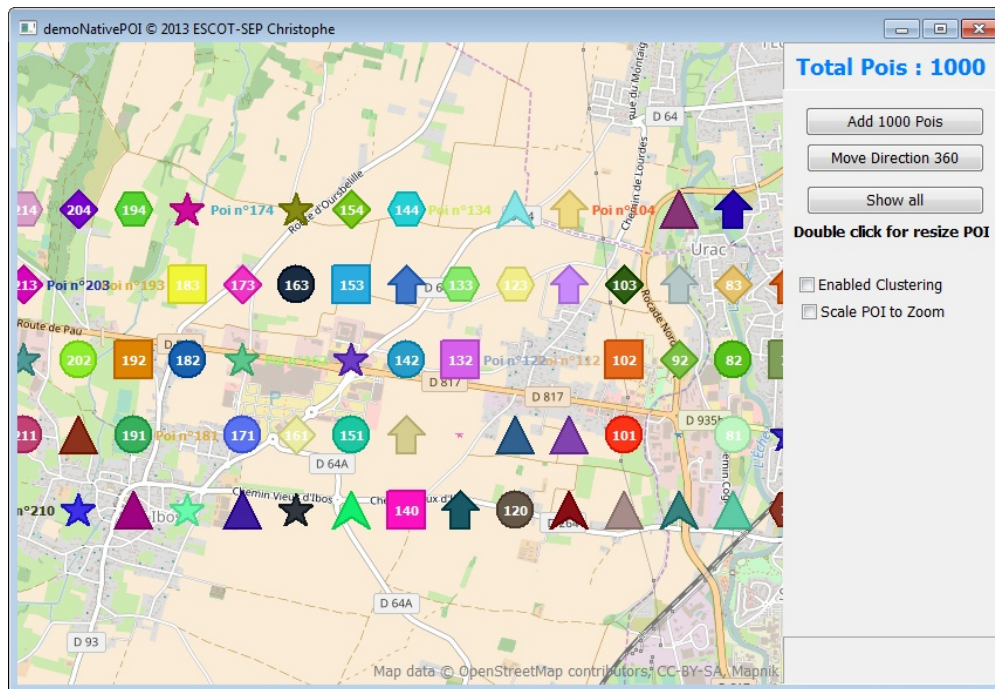


Fig. 45 DemoPOI

Ne fonctionne pas sous Chromium !

Les POIs sont gérés par une liste de type **TECMapPois** accessible au travers de la propriété **Pois** de TECMap

TECMapPois

Cette liste dispose des méthodes et propriétés suivantes :

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

Ajoute un POI au point Latitude,Longitude et retourne son index dans la liste.

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

Retourne le POI positionné en Latitude,Longitude

procedure **Clear**;

Efface tous les Pois

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

Supprime le POI dont on passe l'index

property **Count**:integer;

Retourne le nombre de POI dans la liste

procedure **fitBounds**

Adapte le zoom de la carte pour afficher l'ensemble des Pois

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

Tableau permettant l'accès aux POIs, c'est la propriété par défaut donc vous pouvez y accéder directement par **map.Pois[index]** au lieu de **map.Pois.Poi[index]**

property **ToKml** : string;

Propriété en *lecture seule* qui retourne une string au [format Kml](#) contenant la liste des Pois

property **ToTxt** : string;

Propriété en *lecture/écriture* qui donne accès à la liste des Pois dans un format texte.

*En écriture il s'agit d'un ajout et non d'un remplacement, si vous ne souhaitez pas conserver les anciennes valeurs faites un **Clear** avant l'ajout*

property **ShowHint** : boolean

Indique s'il faut afficher le caption d'un POI sous la forme d'une bulle lors du survol à la souris

property **OnOwnerDrawPOI** : TOnOwnerDrawPOI

Permet d'indiquer une procedure pour prendre en charge le dessin des Pois de type **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

Si un POI est déplacé à la souris cette propriété le référence

TECMapPOI

Il s'agit de la classe gérant un POI, elle dispose des méthodes et propriétés suivantes :

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

Déplace le POI en Latitude,Longitude, déclenche l'évènement OnOverlayMove

procédure **PanTo**;

Déplace la carte pour que son centre coïncide avec le POI, déclenche l'évènement OnMapMove

Le mouvement sera fluide si le déplacement ne dépasse pas la moitié de la hauteur ou de la largeur de la carte.

property **Caption**: string

Libellé

property **Color**: TColor

Couleur du POI

property **HoverColor**: TColor

Couleur au survol de la souris

property **Latitude**: double

Propriete en lecture seule, utilisez SetPosition

property **Longitude**: double

Propriete en lecture seule, utilisez SetPosition

property **X** : integer

Position horizontale en pixels, calculée automatiquement en fonction de la latitude

property **Y** : integer

Position verticale en pixels, calculée automatiquement en fonction de la longitude

property **Width** : integer

Largeur en pixels

property **height** : integer

Hauteur en pixels

property **Draggable** : boolean

Indique si le POI est déplaçable à la souris

property **Shape** : TPOIShape

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

property **Hover** : boolean

Indique si le POI est survolé par la souris

property **OnBeforeDrawPOI** : TOnOwnerDrawPOI

Permet d'indiquer une procedure pour dessiner par dessus des Pois de type **poiEllipse**,

poiRectangle, poiTriangle et poiStar

Exemple, écrive son numéro sur un 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 TFormDemoEMap.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;
```

Événements

Les Pois répondent aux mêmes événements que les overlays (**OnOverlayXXX**), leur TOverlayType est **ovPoi**

Les groupes vous permettent de gérer un ensemble d'éléments, vous pourrez les charger, les sauver, les afficher, les cacher, les déplacer d'une seule commande.

```
// 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);
```

Les éléments que vous pouvez regrouper sont les [markers](#), les [polylines](#), les [polygones](#), les [circles](#), les [rectangles](#), les [groundoverlays](#) et les [labels](#)

la propriété **Groups[]** retourne un **TECMapItemGroup**, tout accès à **Groups** va créer un groupe si celui-ci n'existe pas.

*Le nom des groupes n'est pas sensible à la casse,
map.Groups['group'] est équivalent à 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**;

Procédure à utiliser avant toute modification multiple sur le groupe pour optimiser la vitesse

procedure **EndUpdate**;

Procédure à utiliser, coupler avec **BeginUpdate**, avant toute modification multiple sur le groupe pour optimiser la vitesse

procedure **Clear**;

Vide le groupe de ses éléments, mais ceux-ci ne sont pas supprimés de la carte

procedure **Add**(const item: [TECMapItem](#));

Ajoute un élément dans le groupe

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

Ajoute l'ensemble des TECMapItem situés dans la région délimitée par les point bas (Sud Ouest) et haut (Nord Est).

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

Supprime un élément du groupe, l'élément n'est pas supprimé de la carte

procedure **Delete**;

Supprime tous les éléments du groupe ET de la carte, les éléments n'existent plus

Si vous supprimez un groupe par `map.Groups['Group'].Free` les éléments ne sont pas supprimés de la carte

procedure **Show**;

Rend visible l'ensemble des éléments du groupe

procedure **Hide**;

Cache l'ensemble des éléments du groupe

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

Enregistre le groupe dans un fichier texte.

Utilisez les extensions .gpx , .kml et .json pour spécifier un format, autrement c'est le format interne d'ECMap qui sera employé

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

Charge le contenu d'un fichier texte au format interne d'ECMap

Utilisez les extensions .gpx , .kml et .json pour spécifier un format, autrement c'est le format interne d'ECMap qui sera employé

LoadFromFile peut télécharger les fichiers sur internet

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

Indique si le point situé en Lat,Lng est situé dans la zone englobante le groupe

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

Déplace l'ensemble des éléments du groupe pour que son centre soit situé au point Lat,Lng

```
procedure PanToBounds;
```

Fait glisser la carte pour que le groupe soit visible.

```
procedure fitBounds;
```

Ajuste la vue pour que l'ensemble des éléments du groupe soit visible

```
property Visible:boolean ;
```

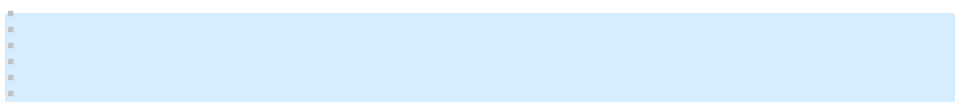
Cette propriété est utilisé par **Show / Hide**

property **MinZoom**:byte ;

Limite minimale du zoom pour que le groupe s'affiche, laissez à 0 pour ignorer le niveau de zoom

property **MaxZoom**:byte ;

Limite maximale du zoom pour que le groupe s'affiche, laissez à 0 pour ignorer le niveau de zoom



property **OnChange** : TNotifyEvent

Déclenché lorsque le périmètre du groupe change ou par l'ajout d'un élément

property **SouthWestLat**: double ;

Indique la latitude du coin bas gauche de la zone qui englobe le groupe

property **SouthWestLng**: double ;

Indique la longitude du coin bas gauche de la zone qui englobe le groupe

property **NorthEastLat**: double ;

Indique la latitude du coin haut droit de la zone qui englobe le groupe

property **NorthEastLng**: double ;

Indique la longitude du coin haut droit de la zone qui englobe le groupe

property **CenterLat** : double ;

Indique la latitude du centre du groupe, utilisez **setCentre** pour la modifier

property **CenterLng** : double ;

Indique la longitude du centre du groupe, utilisez **setCentre** pour la modifier

property **Name** : string;

property **toGPX** : string;

Propriété en **lecture/écriture** qui donne accès aux données de la carte dans le format GPX.

property **ToKml** : string;

Propriété en **lecture / écriture** qui retourne les [overlays](#) (tous sauf les [Labels](#)) au [format Kml](#)

property **ToGeoJSON** : string

Propriété en **lecture / écriture** qui retourne/importe les [overlays](#) (juste les markers, les polylines, les polygones, les cercles et les rectangles) au [format GeoJSON](#)

property **ItemsToTxt** : string;

Propriété en **lecture / écriture** qui retourne/importe les [overlays](#) au format texte interne de TECMAP

property **Count** : integer

Retourne le nombre d'élément contenu dans le groupe

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

Retourne un élément du groupe

La propriété **StreetView** de type `TECMapStreetView` vous permet de gérer l'affichage StreetView

Non disponible sous CloudMade



Fig. 46 Vue StreetView

- 1 Barre de contrôles
- 2 Panneau d'adresse

3 Bouton de fermeture

4 Liens directionnels

TECMapStreetView

Elle donne accès aux propriétés et méthodes suivantes

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

Change la position du point de vue

Déclenche les évènements **OnStreetViewPosition** et **OnStreetViewAvailable**

procédure **ReDraw**;

Redessine la vue pour prendre en compte les options autres que celles relative à la caméra et à la position

property **Visible** : boolean;

Propriété en *lecture/écriture* pour afficher ou non la vue StreetView

Déclenche l'évènement **OnStreetViewVisible**

property **Adress** : string;

Propriété en *lecture* retourne l'adresse du lieu

property **Available** : boolean;

Propriété en *lecture* pour déterminer si StreetView est disponible à cet endroit

property **Latitude** : double;

Propriété en *lecture/écriture* pour définir la latitude, voir **SetPosition**

property **Longitude**: double;

Propriété en *lecture/écriture* pour définir la longitude, voir **SetPosition**

property **Heading** : integer;

Propriété en *lecture/écriture* pour définir la direction de visée de la caméra, 0° = **Nord**, 90° = **Est**, 180° = **Sud** et 270° = **Ouest**

Déclenche l'évènement **OnStreetViewPOV**

property **Pitch** : integer;

Propriété en *lecture/écriture* pour définir la rotation verticale de la caméra, de 90° à -90°

Déclenche l'évènement **OnStreetViewPOV**

property **Zoom** : integer;

Propriété en *lecture/écriture* pour définir le niveau de zoom

Déclenche l'évènement **OnStreetViewPOV**

property **NavigationControl** : boolean;

Propriété en *lecture/écriture* pour afficher ou non la barre de contrôles

property **NavigationPosition** : TControlPosition;

Propriété en *lecture/écriture* permet de définir la position de la barre de contrôles

Les valeurs possibles sont :

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

- cpLeftTop

property **NavigationStyle** : string;

Propriété en *lecture/écriture* pour styler la barre de contrôles

Vous avez le choix entre :

- DEFAULT
- ANDROID
- LARGE
- SMALL

property **AdrControl** : boolean;

Propriété en *lecture/écriture* pour afficher ou non le panneau d'adresse

property **AdrPosition** : TControlPosition;

Propriété en *lecture/écriture* permet de définir la position de l'adresse, voir NavigationPosition pour les valeurs

property **AdrCss** : string;

Propriété en *lecture/écriture* pour attribuer un style CSS à l'adresse

property **CloseBtnVisible** : boolean;

Propriété en *lecture/écriture* pour afficher ou non le bouton de fermeture de la vue StreetView

property **LinkVisible** : boolean;

Propriété en *lecture/écriture* pour afficher ou non les liens directionnels

property **ToTxt** : string;

Propriété en *lecture/écriture* qui donne accès aux paramètres de StreetView sous la forme d'une chaîne texte

```
// red color between 1.2 km and 3.8 kilometer
line := map.shapes.lines[0].Slice(1.2,3.8);
line.Color := claRed;
```

Évènements

OnStreetViewAvailable(sender: TObject; const bVisible: Boolean);

Déclenché lorsque la disponibilité de la vue StreetView change, **bVisible** est à true si StreetView est disponible, false sinon

OnStreetViewPosition(sender: TObject; const dLatitude, dLongitude: Double);

Déclenché lors du changement de position en latitude,longitude

OnStreetViewPOV(Sender: TObject);

Déclenché lors du changement d'axe de la caméra ou du changement zoom

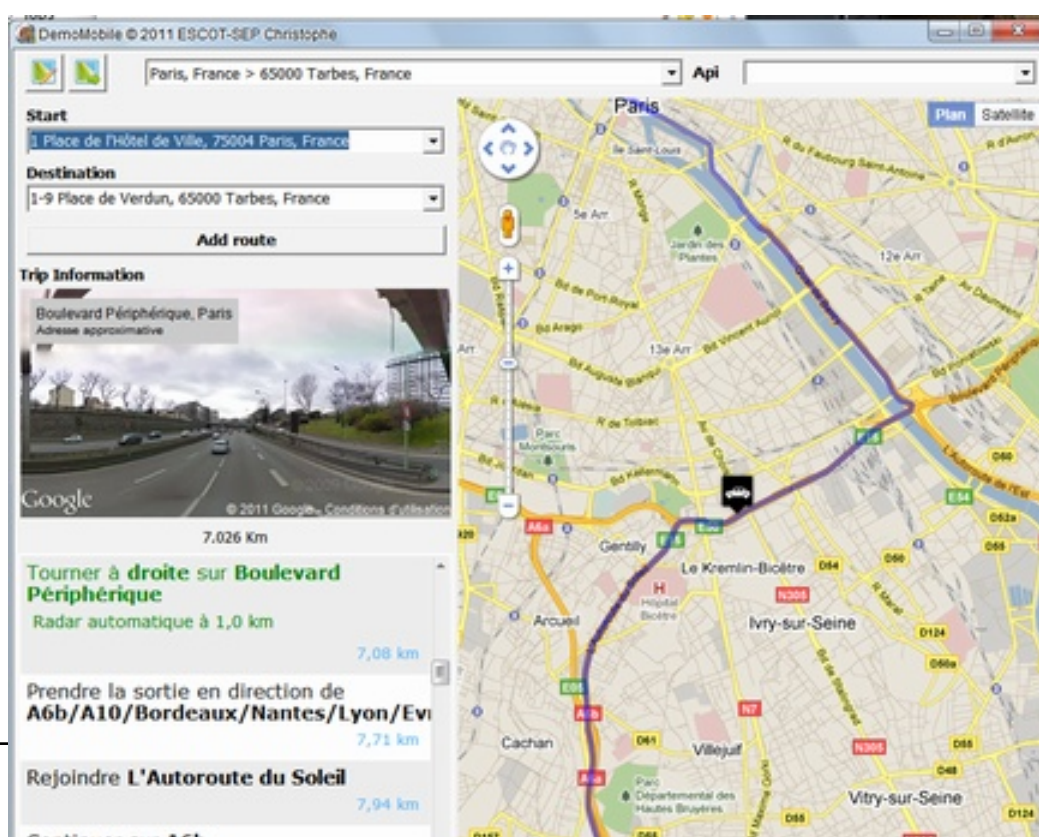
OnStreetViewVisible(sender: TObject;const bVisible: Boolean);

Déclenché lorsque la vue StreetView devient soit visible soit non visible

La fermeture de la vue par le bouton X déclenche cet évènement avec bVisible à false

Démonstration

Le programme **DemoMobile** vous montre une utilisation de StreetView



La propriété **EarthView** de type `TECMapEarthView` vous permet de gérer l'affichage [3D Google Earth](#) ou de [Cesium](#)

Google Earth n'est disponible qu'avec l'api Google

Le plugin Google Earth sera abandonnée par Google en décembre 2015

La suite `TECMap` vous propose le composant indépendant `TECCesiumMap` pour gérer vos vues 3D, ce composant est aussi intégré dans `EarthView` pour remplacer à terme Google Earth.

L'intégration de **Cesium** dans **EarthView** est partielle, le logo  indique la compatibilité avec Cesium

*Il est recommandé d'utiliser directement la propriété **Cesium** qui donne un accès complet au composant gérant Cesium.*

TECMapEarthView

Elle donne accès aux propriétés et méthodes suivantes

property **Cesium**; 

Accès au composant `TECCesiumMap` qui vous permet un contrôle maximal sur le moteur Cesium

property **UseCesium**; 

Choix du moteur Cesium pour gérer la 3D

procédure **Clear**; 

Efface les données

Les données ne sont effacées que dans la vue 3D elles sont toujours disponible dans la vue 2D standard

procédure **EnableLayer**(const sLayer:string;const bShow:boolean) 

Activer/Désactiver un layer

Vous avez le choix parmi

- LAYER_BORDERS
- LAYER_BUILDINGS
- LAYER_BUILDINGS_LOW_RESOLUTION
- LAYER_BUILDINGS_LOW_RESOLUTION.
- LAYER_ROADS
- LAYER_TERRAIN
- LAYER_TREES

Sous Cesium seul LAYER_TERRAIN est fonctionnel

LAYER_TERRAIN est activé par défaut

```
// Delphi map component EMap  
  
// active LAYER_BUILDINGS and LAYER_TREES  
map.EarthView.EnableLayer( 'LAYER_BUILDINGS',true);  
map.EarthView.EnableLayer( 'LAYER_TREES',true);
```


procédure **Camera**(const dLatitude,dLongitude:double) 

Positionne la caméra en dLatitude,dLongitude et affiche la vue en accord avec les propriétés **Altitude, Roll, Tilt** et **Heading**

procédure **LookAt**(const dLatitude,dLongitude:double) 

Regarde le point situé en dLatitude,dLongitude et affiche la vue en accord avec les propriétés **Altitude, Range, Tilt** et **Heading**

fonction **getGroundAltitude**(const dLatitude,dLongitude:double):double;

Retourne l'altitude du point situé en dLatitude,dLongitude

procédure **LoadKml**(const UrlKmlKmz:string); 

Chargement d'un fichier Kml/Kmz distant

UrlKmlKmz url du fichier à importer

Les données sont ajoutées aux données déjà présentes, faites un Clear avant si vous souhaitez un remplacement total

procédure **PlacemarkCamera**(const id:integer)

Positionne la caméra sur la marque d'indice id et affiche la vue en accord avec les propriétés **Altitude, Roll, Tilt** et **Heading**

procédure **PlacemarkLookAt**(const id:integer)

Regarde la marque d'indice id et affiche la vue en accord avec les propriétés **Altitude, Range, Tilt** et **Heading**

procédure **setPlacemarkLatLngAlt**(const id:integer;const Lat,Lng,Alt:double);

Positionner la marque d'indice id en Latitude,Longitude et Altitude

procédure **getPlacemarkLatLngAlt**(const id:integer;var Lat,Lng,Alt:double);

Obtenir la latitude, longitude et altitude de la marque d'indice id

procédure **FromXYToLatLngAlt** (const iX, iY: double; var dLatitude, dLongitude, altitude: double)

Donne la latitude, longitude et l'altitude du point X,Y (pixels)

property **Altitude** : double 

Propriété en **lecture/écriture** pour déterminer son altitude

property **MouseAltitude** : double 

Propriété en **lecture** indique l'altitude du point situé sous le curseur de la souris

property **AltitudeMode**: TAltitudeMode

Propriété en **lecture/écriture** pour fixer la manière dont est calculée l'altitude

Vous avez le choix entre

- amAbsolute
- amRelativeToGround
- amClampToGround
- amRelativeToSea
- amClampToSea

property **FlyToSpeed** : double

Propriété en **lecture/écriture** pour fixer la vitesse de déplacement, de 0 à 5.0

Utilisez 6.0 pour un téléportage immédiat

property **Loaded** : boolean

Propriété en **lecture seule** pour déterminer si le plugin a bien été chargé

property **Latitude** : double 

Propriété en **lecture/écriture** pour déterminer la latitude

property **Longitude** : double 

Propriété en **lecture/écriture** pour déterminer la longitude

property **Heading** : double 

Propriété en **lecture/écriture** pour déterminer l'orientation par rapport au Nord (de 0° à 360°)

property **Tilt** : double 

Propriété en **lecture/écriture** pour déterminer l'angle de la vue (de 0° à 90° pour **LookAt** et de 0° à 180° pour **Camera**)

property **Range** : double

Propriété en **lecture/écriture** pour déterminer la distance par rapport au point observé (non utilisé par **Camera**)

property **Ready** : boolean

Propriété en **lecture seule** pour déterminé le moment où le plugin a fini son rendu écran, correspond à l'évènement OnEarthViewFrameEnd

property **Visible** : boolean 

Propriété en **lecture/écriture** pour activer/désactiver la vue 3D

property **Atmosphere** : boolean

Propriété en **lecture/écriture** pour afficher ou non l'atmosphère

property **Grid** : boolean

Propriété en **lecture/écriture** pour afficher ou non les lignes de latitudes et longitudes sur le globe

property **MouseNavigationEnabled** : boolean 

Propriété en **lecture/écriture** pour activer ou non le déplacement à la souris

property **NavigationControlVisibility** : TNavigationControlVisibility

Propriété en **lecture/écriture** pour afficher ou non les contrôles, vous avez le choix entre **ncvShow**, **ncvHide** et **ncvAuto**

```
// Delphi map component EMap
map.EarthView.NavigationControlVisibility := nvcAuto;
```

property **StatusBar** : boolean

Propriété en **lecture/écriture** pour afficher ou non la barre d'information

property **ScaleLegend** : boolean

Propriété en **lecture/écriture** pour afficher ou non l'échelle

property **Sun** : boolean

Propriété en **lecture/écriture** pour afficher ou non le soleil

property **OverviewMap** : boolean

Propriété en **lecture/écriture** pour afficher ou non une mini carte du monde

property **TerrainExaggeration** : double

Propriété en **lecture/écriture** qui est un facteur d'accroissement de l'altitude de 1.0 à 3.0

property **ToKml** : string; 

Propriété en **lecture/écriture** qui permet l'import/export au [format Kml](#)

TECMap dispose de la fonction toKml pour exporter la quasi totalité des overlays au format Kml

```
// Delphi map component EMap
// import overlays (markers, rectangles, circles,...routes)
map.EarthView.toKml := map.toKml;
```

property **ToTxt** : string;

Propriété en **lecture/écriture** qui donne accès aux propriétés dans un format texte.

Sous Google Earth les donnée Kml ne sont pas prises en compte

property **PlaceMark**[const index:integer]: string

Propriété en **lecture/écriture** qui détermine le contenu de la marque d'indice **id**

La chaîne est au format Kml, du genre

```
<Placemark>
<name>Placemark from KML string</name>
<Point>
<coordinates>-122.448425,37.802907,0</coordinates>
</Point>
</Placemark>
```

property **PlacemarkCount** : integer

Retourne le nombre de marques

property **Models** : [TECMapEarthModels](#)

Liste des modèles de type `TECMapEarthModel`

Un modèle est défini par la balise Kml **<Model>**

property **CesiumModels** : `TECCesiumShapeModelList` 

Liste des modèles 3D de type `TECCesiumShapeModel`

property **StreamPercent** : integer

Propriété en **lecture** qui indique le pourcentage de chargement de la vue

property **ApiVersion** : string

Retourne la version de l'api Google Earth

property **PluginVersion**

Retourne le numéro de version du plugin 3D

*Pour que les modifications sur les propriétés **Altitude, Heading, Range, Roll et Tilt** soient visibles vous devez déclencher un appel à **LookAt** ou à **Camera***

TECMapEarthModels

Cette classe possède les méthodes et propriétés

procedure **LoadKml**(const filename:string);

charger un fichier Kml/Kmz définissant un modèle 3D, déclenche l'évènement

OnEarthViewUpdateModels

```
// Delphi map component EMap
// load 3d model
map.EarthView.models.LoadKml(
)http://earth-api-samples.googlecode.com/svn/trunk/demos/drive-simulator/
```

*Si des modèles sont présent dans le Kml chargé par
TECMapEarthView.ToKml ou par TECMapEarthView.LoadKml ils seront
aussi accessible dans la liste Models*

function **IndexOf**(const idModel:string):integer;

Retourne l'indice du modèle dont on passe l'identifiant

Il s'agit de la propriété ID de la balise Model

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

Supprime le modèle index

function **Count**:integer;

Retourne le nombre de modèle 3D

procedure **Clear**;

Efface tous les modèles 3D

property **Model**[Index:integer]:TECMapEarthModel

Tableau permettant l'accès aux modèles, c'est la propriété par défaut donc vous pouvez y accéder directement par **map.EarthView.Models[index]** au lieu de

`map.EarthView.Models.Model[index]`

TECMapEarthModel

Classe gérant un modèle 3D, elle possède les méthodes et propriétés

procédure **setLatLngAlt**(const Lat,Lng,Alt:double);

Position le modèle en latitude, longitude, altitude

procédure **LookAt**;

Regarder le modèle

property **Id** : string ;

Propriété en **lecture** qui donne l'identifiant du modèle

*Il s'agit de la propriété **ID** de la balise **Model***

property **Index** : integer

Indice du modèle dans la liste **Models**

property **Latitude** : double

Propriété en **lecture/écriture** pour déterminer la latitude

property **Longitude** : double

Propriété en **lecture/écriture** pour déterminer la longitude

property **Heading** : double

Propriété en **lecture/écriture** pour déterminer l'orientation par rapport au Nord (de 0° à 360°)

property **Tilt** : double

Propriété en **lecture/écriture** pour déterminer l'angle de la vue (de 0° à 90°)

property **Range** : double

Propriété en **lecture/écriture** pour déterminer la distance par rapport au point observé)

property Roll : double

Propriété en **lecture/écriture** pour déterminer la rotation du modèle

property Altitude : double

Propriété en **lecture/écriture** pour déterminer l'altitude du modèle

Visite guidée (Tour)

Google Earth vous permet de jouer des visites guidées, nommées **tour** (voir [la documentation de google earth api](#))

EarthView vous offre la possibilité de les gérer de manière simple.

property Tour : string

Propriété en **lecture/écriture** pour charger un tour

- Vous pouvez charger un fichier kml/kmz depuis internet ou en local
- Vous pouvez aussi directement passer une chaine contenant les données kml

```
// 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>...'
```

L'événement OnLoadTour est déclenché lorsque le tour est chargé et disponible

procédure **PlayTour**;

Démarre le tour

L'événement OnEndTour est déclenché lorsque le tour est terminé

procédure **PauseTour**;

Met en pause le tour

procédure **ResetTour**;

Positionne le tour au départ, en attente d'être joué

procédure **ExitTour**;

Quitte le tour

property **CurrentTimeTour** : integer;

propriété en **lecture/écriture** qui donne/fixe la position du tour dans le temps (en secondes)



property **Duration** : integer

propriété en lecture qui donne la durée du tour en secondes

property **StateTour** : TEarthStateTour

propriété en lecture qui donne l'état du tour (stNone,stPlay,stPause,stEnd)

Évènements

*En plus des événements ci-dessous vous avez aussi accès à **OnMapClick** , **OnMapDbClick**, **OnMouseMove** , **OnOverlayMouseDown**, **OnOverlayMouseUp** et **OnOverlayMouseOut***

property **OnEarthViewInit** : TNotifyEvent; 

Déclenché quand le plugin est chargé, cela se produit lorsque vous basculez **TECMapEarthView.Visible** à true

property **OnEarthViewChange** : TNotifyEvent;

Déclenché lorsque la vue change

property **OnEarthViewFrameEnd** : TNotifyEvent;

Déclenché lorsque le rendu 3D est terminé, c'est le bon endroit pour déplacer les modèles 3D

pour avoir une animation fluide

property **OnEarthViewUpdateModels**: TNotifyEvent;

Déclenché par l'ajout d'un modèle

property **OnEarthViewUpdatePlacemarks** : TNotifyEvent;

Déclenché par l'ajout d'une marque

property **OnEarthViewClickPlacemark**(sender: TObject;const index:integer;const KmlString:string)

Déclenché par un click sur une marque

index est l'indice de la marque dans la liste **PlaceMark**

KmlString le contenu de la marque au format Kml, du genre

```
<Placemark>
<name>Placemark from KML string</name>
<Point>
<coordinates>-122.448425,37.802907,0</coordinates>
</Point>
</Placemark>
```

Démonstrations

Demo3D vous montre en particulier comment déplacer un modèle 3D en suivant un trajet précis.



Le programme DemoOverlay vous montre comment transférer des données vers une vue 3D

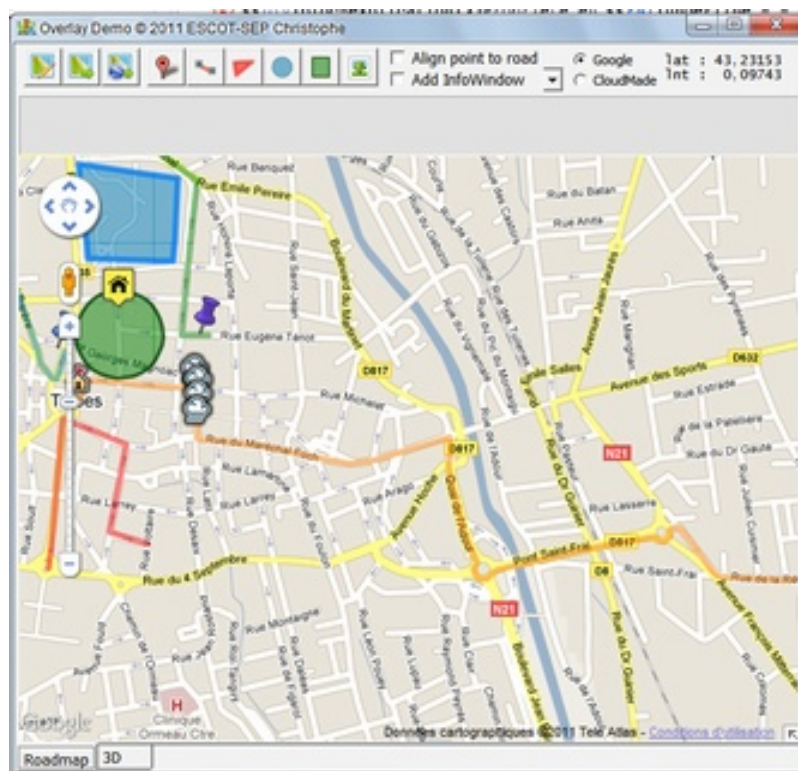
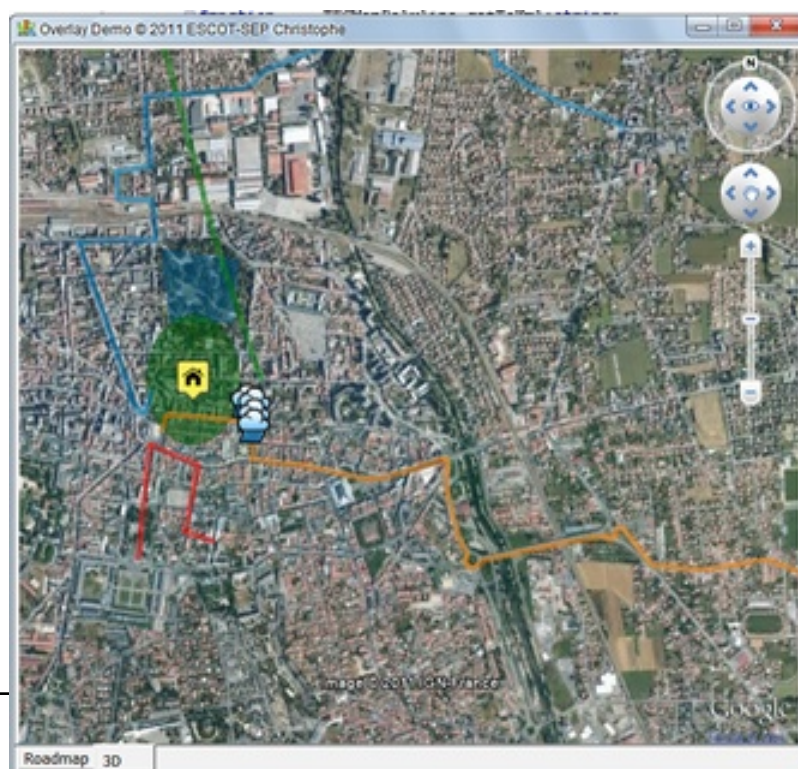


Fig. 49 DemoOverlay



La propriété **PanoramioView** de type **TECMapPanoramioLayer** vous permet entre autre d'afficher les images en provenance de [Panoramio](#)

Cela n'est disponible que sous Google Maps mais une [émulation](#) est possible.



Fig. 51 Activation de la vue Panoramio

```
// Delphi map component EMap  
  
// show panoramio layer  
map.PanoramioView.visible := true;
```

TECMapPanoramioLayer

Cette classe dispose des propriétés

property **Clickable** : boolean;

Propriété en *lecture/écrite* qui rend clickable ou non les images

Déclenche l'évènement [OnPanoramioLayerClick](#)

property **SuppressInfoWindows** : boolean;

Propriété en *lecture/écrite* qui permet ou non l'affichage de l'infowindow associée à la miniature de l'image

property **Tag** : string;

Propriété en *lecture/écrite* qui permet un filtrage par mot clef

property **UserId** : string;

Propriété en *lecture/écrite* qui permet un filtrage par UserId, identifiant Panoramio du posteur des images.

property **Visible** : boolean;

Propriété en *lecture/écrite* qui permet d'afficher ou non la vue panoramio

property **ToTxt** : string;

Propriété en *lecture/écriture* qui donne accès à **PanoramioView** sous la forme d'une chaîne texte

procedure **Search**;

Lance une recherche d'images dans une zone rectangulaire de coordonnées LatSW, LngSW, LatNE, LngNE

procedure **SearchAt**(const Lat,Lng,Radius:double);

Lance une recherche d'images dans une zone rectangulaire centrée sur les coordonnées Lat,Lng dans un rayon Radius exprimé en Km

La fonction de recherche d'images est disponible sous CloudMade et Google Maps

La recherche n'est pas bloquante, l'évènement OnPanoramioSearch est déclenché lorsque les résultats sont disponibles.

Chaque requête retourne au maximum un bloc de 100 images, pour savoir si d'autres images sont disponibles vous devez tester la propriété **HasMore** et relancer la recherche par **NextSearch**, le bon endroit pour le faire est dans l'évènement OnPanoramioSearch.

procedure **NextSearch**;

Relancer la recherche pour obtenir les images suivantes, l'évènement OnPanoramioSearch est aussi déclenché par cette procedure

function **IsSearch**:boolean;

Indique si une recherche est en cours

function **Count**:integer;

Retourne le nombre d'images trouvé

function **HtmlPhoto**(const index:integer;size:TECPanoramioImageSize):string;

Retourne une balise html IMG permettant d'afficher une image dans une taille déterminée (pimMini_square,pimSquare,pimThumbnail,pimSmall,pimMedium ou pimOriginal)

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

Retourne un contenu html indiquant le copyright de l'image

Exemple d'utilisation afficher l'image en miniature avec son titre et copyright dans une 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);
```

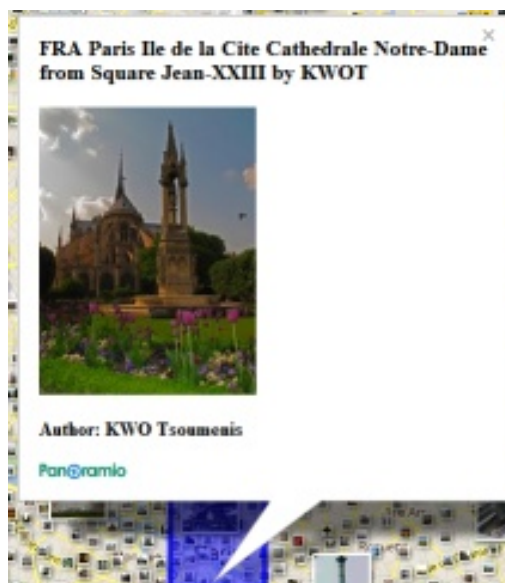


Fig. 52 Affichage d'une InfoWindows avec les informations d'une image Panoramio

property **HasMore**:boolean

Indique si d'autres images sont disponibles dans la zone de recherche

Les propriétés suivantes sont associées aux images trouvées

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

Les propriétés suivantes déterminent le point Sud-Ouest et le point Nord-Est de la zone de recherche

property **LatSW** : double

property **LngSW** : double

property **LatNE** : double

property **LngNE** : double

OnPanoramioLayerClick

Évènement déclenché par un clic sur une miniature Panoramio

procedure **OnPanoramioLayerClick**(sender : TObject; const Author,PhotoId,title,Url,UserId,Html:string;const dLatitude,dLongitude : double)

Author auteur de la photo

PhotoID identifiant Panoramio de la photo

Title titre de la photo

Url url de la photo

UserId identifiant Panoramio du posteur de la photo

Html contenu Html de l'infowindow associée à la photo

dLatitude,dLongitude Latitude et Longitude de la photo

OnPanoramioSearch

Évènement déclenché par une recherche d'images que cela soit avec **SearchAt** ou **NextSearch**

procedure OnPanoramioSearch(sender: TObject; const First, Last: Integer);

First et **Last** indiquent les bornes des images trouvées, en effet chaque recherche ne peut retourner au maximum que 100 images

DemoLayer

Le programme **DemoLayer** vous permet de voir comment utiliser **Panoramio** ainsi que les autres types de **layers**

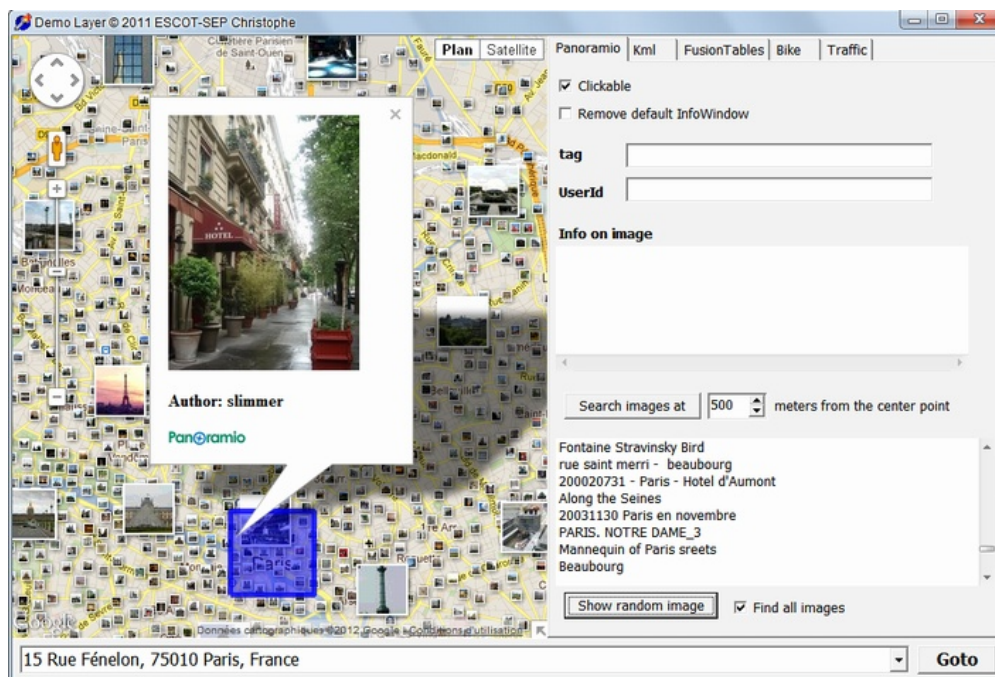


Fig. 53 DemoLayer

La propriété **DistanceMatrix** de type **TECDistanceMatrix** vous permet d'obtenir les distances et temps de voyages pour chaque couple d'adresses départ/arrivée.

Non disponible sous CloudMade



Fig. 54 DemoMatrix démonstration de l'utilisation de DistanceMatrix

Le programme de démonstration DemoMatrix est une adaptation en Delphi de l'[exemple javascript de google](#)

*L'exemple de google ne fonctionne pas sous IE car il y a une virgule en fin des arrays **origins** et **destinations** qui fait planter le moteur javascript d'IE*

TECDistanceMatrix

Elle donne accès aux propriétés et méthodes suivantes

fonction **Count**:integer;

Retourne le nombre de couples Départ/Arrivée

procédure **Update**;

Lancer le calcul des distances et temps pour chaque couple.

l'évènement OnDistanceMatrix est déclenché lorsque le calcul est terminé

Property **Origins**:TStringList;

Liste des points de départ

Property **Destinations**:TStringList;

Liste des points d'arrivée

property **TravelMode** : TDirectionsTravelMode;

Choix du type de route **tmDriving** ou **tmWalking**

property **UnitSystem** : TUnitSystem;

Choix du système d'unité, **usMetric** (kilometre) ou **usImperial** (miles)

property **AvoidHighWays** : boolean;

Éviter ou non les autoroutes

property **AvoidTolls** : boolean;

Éviter ou non les péages

property **Cells**[indexOrigine,indexDestination:integer]:TECDistanceMatrixItem; default;

Tableau permettant l'accès aux données, c'est la propriété par défaut donc vous pouvez y accéder directement par **map.DistanceMatrix[indexOrigine,indexDestination]** au lieu de **map.DistanceMatrix.cells[indexOrigine,indexDestination]**

indexDestination est l'index du point d'arrivé dans la liste **Destinations**

indexOrigine est l'index du point de départ dans la liste **Origins**

Retourne nil si les indices sont en dehors des bornes

```
// 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 vous permet d'obtenir vos données de manière séquentielle, utilisez Count pour connaître le nombre total de couples.

Ils sont rangés dans l'ordre suivant

(Départ1-Arrivée1), (Départ1-Arrivée2),..., (Départ2-Arrivée1), (Départ2-Arrivée2),...

Retourne nil si l'indice est en dehors des bornes

TECDistanceMatrixItem

Cette classe représente un couple Départ/Arrivée, elle vous donne accès aux propriétés

property **Origin** : string

Adresse de départ

property **Destination** : string

Adresse d'arrivée

property **Distance** : integer

Distance en metres

property **Duration** : integer

Durée en secondes

property **DistanceText** : string

Distance sous la forme d'un texte qui tient compte de l'unité système choisie (**tmMetric** ou **tmImperial**)

property **DurationText** : string

Durée sous forme texte

property **Status** : string

Indique une éventuelle erreur, **OK** si pas d'erreur et si un résultat a été trouvé

OnDistanceMatrix

Évènement déclenché après l'appel à **DistanceMatrix.Update** lorsque les résultats sont disponibles

TECNativeMap disponible en version **VCL et Firemonkey** (**Windows** , **Mac OS X** , **iOs** et **Android**) pour [le même prix](#) !

Vous profitez d'un [mode hors-ligne](#), et vous gardez le contrôle sur vos données en évitant Google Maps et autre Bing Maps.

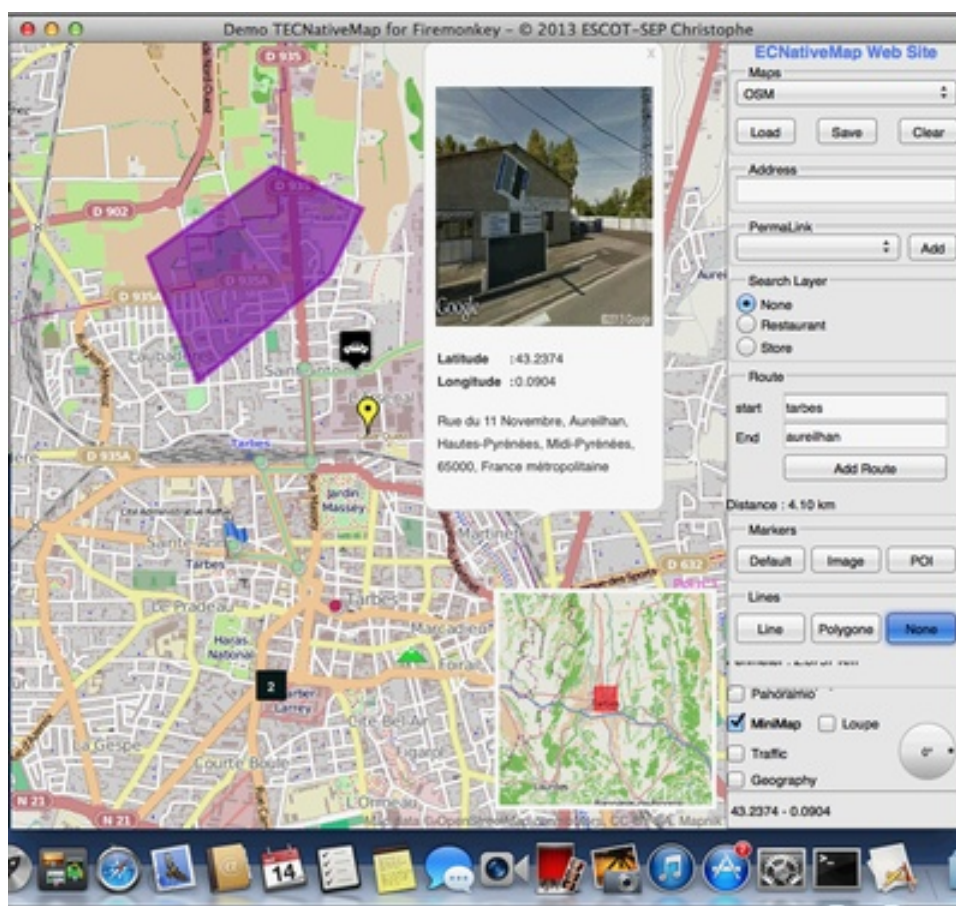


Fig. 55 ECNativeMap on Mac OS X

[testez une démonstration pour Android](#)

TECNativeMap vous permet d'utiliser les cartes d'OpenStreetMap, de MapQuest, ArcGIS World, d'OpenCycleMap, de MapBox, d'OPNV mais

aussi vos propres serveurs, [ces cartes sont aussi disponibles avec TECMap](#).

Pour sélectionner un fournisseur de cartes utilisez la propriété **TileServer**

```
map.TileServer := tsOsm;
```

Bing

En renseignant la propriété **BingKey** avec [votre clef Bing](#) vous pourrez utiliser les tuiles de Bing Maps (tsBingRoad, tsBingAerial, tsBingAerialLabels)

```
map.BingKey      := YOUR_BING_KEY
map.TileServer   := tsBingRoad;
```

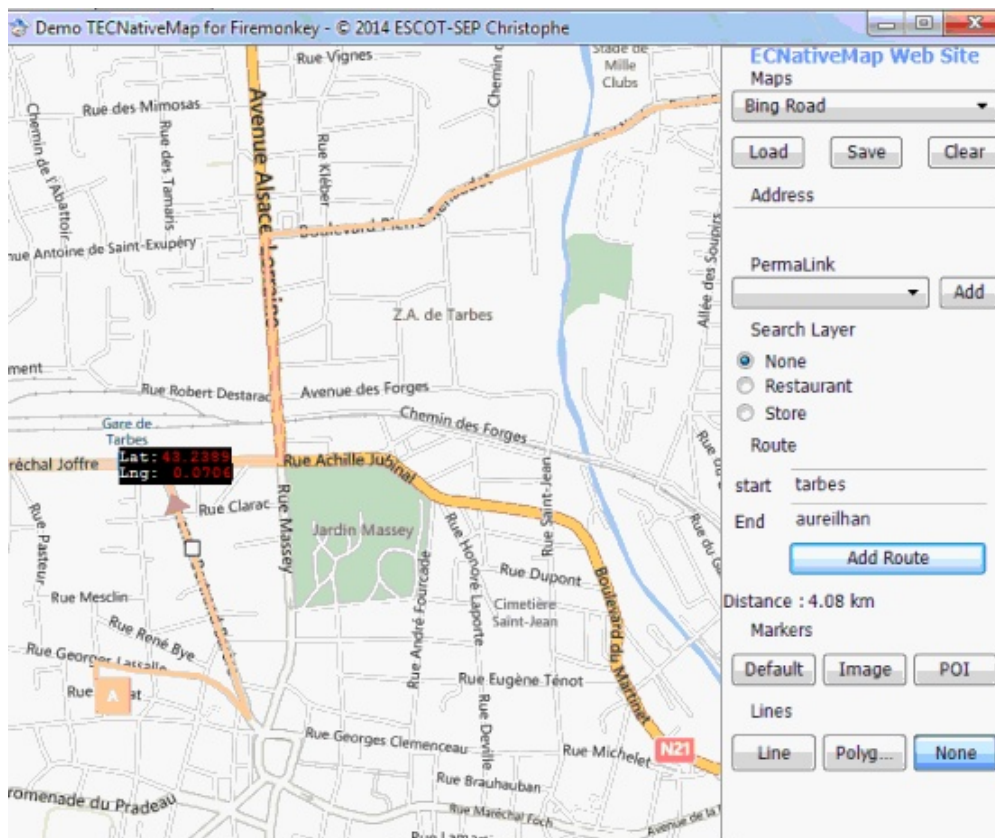


Fig. 56 Bing Maps

Here

En renseignant les propriétés **HereID** et **HereCode** vous pouvez utiliser les **tuiles de Here** (**tsHereNormal**, **tsHereTerrain**, **tsHereSatellite**, **tsHybrid**, **tsHereMobile**, **tsHereHiRes**, **tsHereTransit**, **tsHereTraffic**, **tsHereFlow**, **tsHereTruck**, **tsHereTruckTransparent**)

```
map.HereID      := YOUR_HERE_ID
map.HereCode    := YOUR_HERE_CODE
map.TileServer  := tsHereTraffic;
```

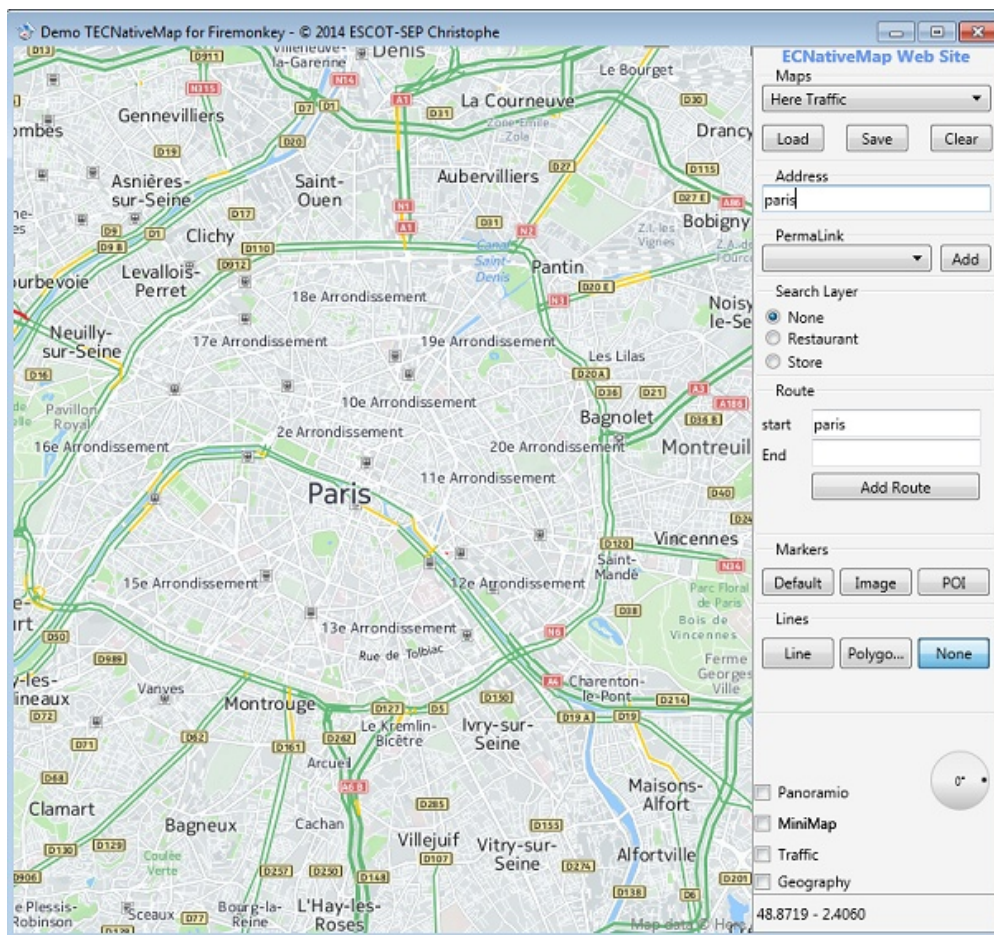


Fig. 57 Here Traffic

MapBox

Pour utiliser les tuiles de [MapBox](#) (`tsMapBoxSatellite`, `tsMapBoxStreets`, `tsMapBoxStreetsSatellite`, `tsMapBoxStreetsBasic`) il vous suffit d'entrer votre token dans **MapBoxToken**.

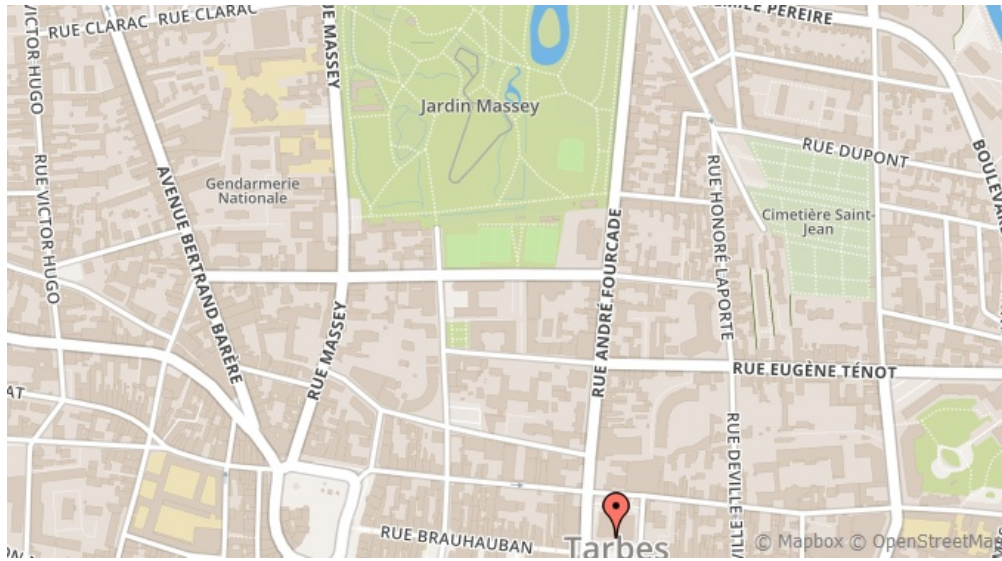


Fig. 58 MapBoxStreets

Yandex

`tsYandexNormal`, `tsYandexSatellite` et `tsYandexHybrid` sont aussi utilisables (merci alexey t.) et vous permettent d'utiliser les tuiles de Yandex.

Yandex utilise la projection Mercator elliptique différente de la projection par défaut (Mercator sphérique style google maps), cela est pris en charge par TECNativeMap vous n'avez donc rien à faire.

Vous ne pourrez pas utiliser tsYandexHybrid avec une base de tuiles qui n'utilise pas la même projection !

Utilisez votre propre serveur de tuiles

En plus des cartes précédente vous pouvez utiliser n'importe quel autre fournisseur utilisant le même format, l'exemple ci-dessous ne doit pas être employé en production pour des raisons légales.

```

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;

```

*Pour des questions de licences vous ne pouvez pas utiliser directement les tuiles de Google, mais vous le pouvez au travers du composant **TECMap**.*

Inscrutation de tuiles

Vous pouvez utiliser autant de serveurs de tuiles que vous le voulez pour les afficher par dessus votre carte de base, l'exemple ci-dessous incruste une carte des nuages.

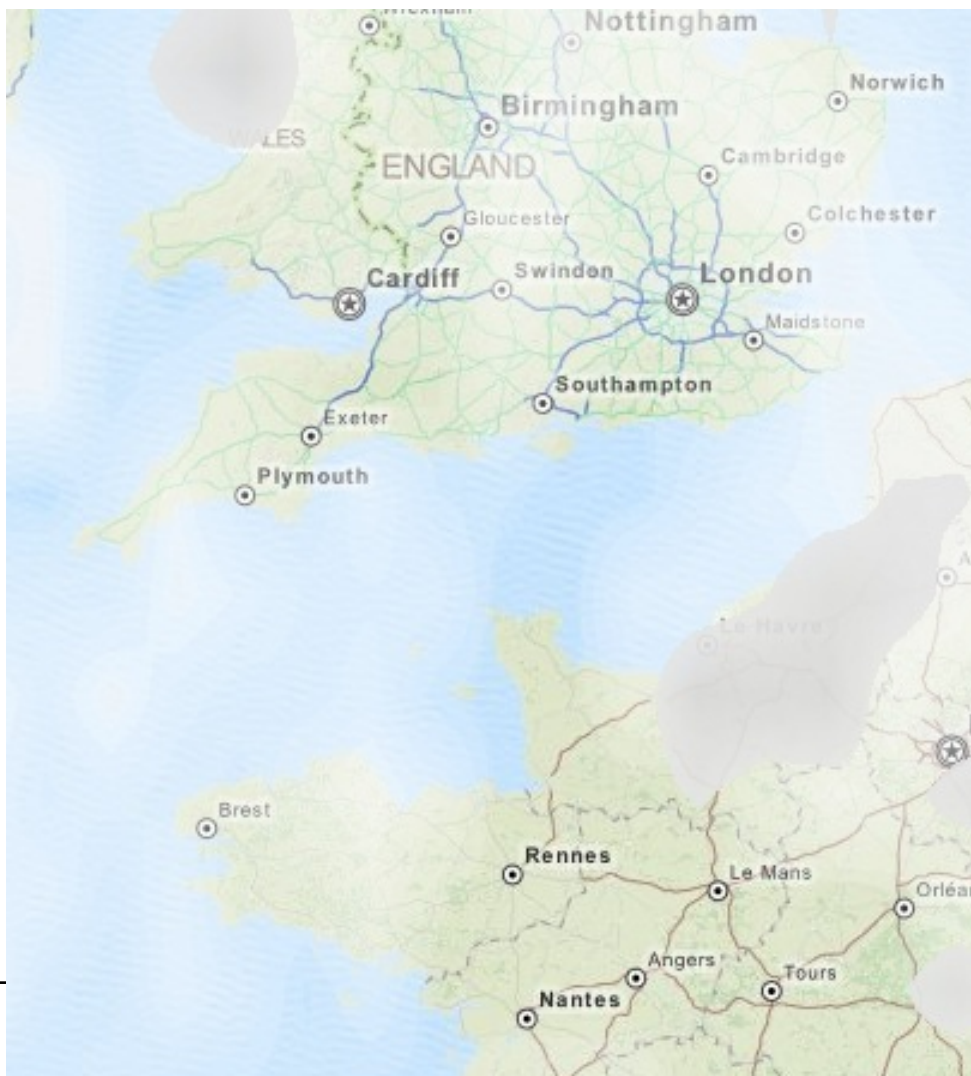
```
var overlay:TECOverlayTileLayer;
...
// overlay clouds tiles
overlay := map.AddOverlayTiles(GetWeatherTile);

// remove overlay

map.RemoveOverlayTiles(overlay);

...

procedure TForm.GetWeatherTile(var TileFilename:string;
const x,y,z:integer);
begin
    TileFilename := format(
    'C:\Program Files\TECNativeMap\clouds/%d/%d/%d.png',
    x,y,z);
end;
```



Vous pouvez aussi utiliser un stream

```

procedure TForm.getOverlayTileStream(var
  TileStream: TMemoryStream;const x, y, z: integer);
begin
  // here fill the stream with your tile

end;

...
map.AddOverlayStreamTiles(getOverlayTileStream);

```

Ou directement utiliser un serveur prédéfini transparent comme [tsHereFlow](#) ou [tsHereTruckTransparent](#)

```

// mix Bing Aerial Labels and Here Flow
map.TileServer      := tsBingAerialLabels;
map.AddOverlayTileServer(tsHereFlow);

```

En mode VCL Votre overlay doit utiliser des pngs pour qu'il s'inscrute sur votre carte, sous FMX vous pouvez utiliser la propriété opacity.

```

var overlay:TECOverlayTileLayer;
...
// overlay clouds tiles
overlay := map.AddOverlayTiles(GetWeatherTile);

// set opacity 0..1
overlay.opacity := 0.5;

```

Utilisez **RemoveAllOverlayTiles** pour supprimer tous les overlays

*Vous pouvez aussi
à l'échelle*

seule si elle est

Positionnement sur la carte

Vous pouvez déterminer les coordonnées du centre de la carte au travers des propriétés **Latitude** et **Longitude**, elles sont de type **double** et accessible en **lecture/écriture**

Vous pouvez directement modifier les coordonnées au travers de la procédure **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;
```

Position de la souris

Les latitude et longitude du point situé sous le curseur de la souris sont renvoyés par la propriété **MouseLatLng**

Vous pouvez vous brancher sur l'évènement **OnMapMouseMove** pour connaître en temps réel votre position

Réagir au changement de position

Lorsque le centre de la carte est déplacé, que cela soit par code ou directement à la souris, l'évènement **OnMapMove(sender: TObject;const dLatitude,dLongitude:double)** est déclenché

Sender représente le composant EMap qui a été modifié, dLatitude et dLongitude les nouvelles coordonnées, qui sont aussi accessible au travers de **Latitude** et **Longitude**

Lorsque la carte commence à bouger l'évènement **OnMapDragStart** est déclenché, puis **OnMapDrag** pendant le déplacement et **OnMapDragEnd** en fin de déplacement.

Vous pouvez aussi réagir lors du déplacement de la souris en vous branchant sur **OnMapMouseMove**

Zone affichée

Vous pouvez déterminer les coordonnées du point Nord Est (en haut à droite) et du point Sud Ouest (en bas à gauche) de la zone affichée par le composant au travers des propriétés **NorthEastLatitude**, **NorthEastLongitude**, **SouthWestLatitude** et **SouthWestLongitude**, elles sont de type double et accessible *uniquement en lecture*.

La procédure **BoundingBox(maxLatitude,maxLongitude,minLatitude,minLatitude)** limite la zone accessible de votre carte, l'évènement **OnOutOfBounds** est déclenché si vous tentez d'accéder à une zone interdite.

*Un appel à **BoundinBox** sans paramètre lève les limitations.*

La procédure **fitBounds(const dLatlo,dLnglo,dLathi,dLnghi:double)** vous permet de d'ajuster la vue aux coordonnées passées.

la procédure **boundingCoordinates(const lat, lng, radius: double;**

var latSW, lngSW,latNE, lngNE: double); retourne les coordonnées d'une zone rectangulaire en fonction du point central et d'une distance en Km.

La fonction

ContainsLatLng(var dLatitude,dLongitude:double):boolean vous indique si le point en dLatitude,dLongitude est dans la portion visible de la carte.

Dés que la vue change l'évènement

OnChangeMapBounds(sender: TObject) est déclenché.

La fonction **ScreenShot** retourne un TBitmap contenant l'image de la carte

Zoom

La propriété **Zoom** vous permet de contrôler la définition de votre carte, elle est de type integer et est accessible en **lecture/écriture**

Vous avez accès aussi au propriété MaxZoom et MinZoom qui vous permettent de déterminer les limites du zoom

Utilisez la procedure **ZoomAround(const LatLngZoom:TLatLng; const NewZoom:Integer)** pour zoomer tout en restant centré sur un point précis (comme à la souris)

Le changement de zoom déclenche l'évènement **OnChangeMapZoom(sender: TObject)**

En positionnant la propriété `dragRect` sur `drZoom` vous pouvez sélectionner une zone avec la souris en maintenant le bouton droit enfoncé, lorsque vous relâchez le bouton un zoom est effectué sur la région sélectionnée.

ZoomScaleFactor

Cette propriété vous permet d'émuler les zooms intermédiaires

```
// emulate zoom 16.35  
  
map.zoom := 16;  
map.zoomScaleFactor := 35;
```

*Faites Click droit + molette pour changer
ZoomScaleFactor à la souris*



Fig. 60 ZoomScaleFactor

ZoomScaleFactor peut varier de 0 à 999, si le zoom

*maximum n'est pas atteint 100 correspond à zoom+1
(et inversement)*

Utilisez la procédure **ZoomScaleFactorAround(const LatLngZoom:TLatLng; const NewZoomScaleFactor:Integer)** pour zoomer tout en restant centré sur un point précis (comme à la souris)

Adapter les markers au Zoom

Utilisez la propriété **ScaleMarkerToZoom** pour que la taille des markers changent en fonction du Zoom.



Fig. 61 ScaleMarkerToZoom

```
map.ScaleMarkerToZoom := true;
```

Sélection

La propriété **Selected** vous permet de sélectionner des éléments.

```

    // 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;

```

En positionnant **DragRect** sur drSelect vous pouvez effectuer une sélection avec la souris en maintenant le bouton droit enfoncé puis en le relachant une fois la zone délimitée.

Vous pouvez redéfinir l'apparence du rectangle de sélection en utilisant [les styles](#), exemple pour avoir une épaisseur de ligne de 10 pixels, une couleur verte et une ligne en pointillé

```

map.styles.addRule(
'#_DRAGZOOM_.line {weight:10;color:green;penStyle:dash}');

```

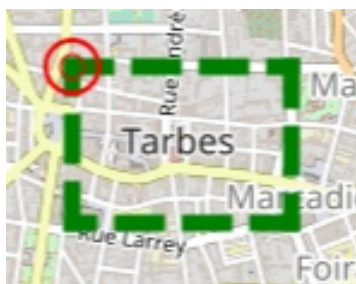


Fig. 62 sélection stylisée

Par défaut les éléments sélectionnés sont dessinés en utilisant la couleur définie pour le survol de la souris, vous pouvez ajouter une marque distinctive en vous branchant sur leur événement OnAfterDraw.

Cet exemple ajoute une étoile rouge en haut à droite des éléments sélectionnés

```
// 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;
```



Rotation de la carte

Disponible seulement dans la version Firemonkey

Placez votre composant dans un conteneur (TPanel, TRectangle) qui servira à délimiter la zone visible puis activez la propriété

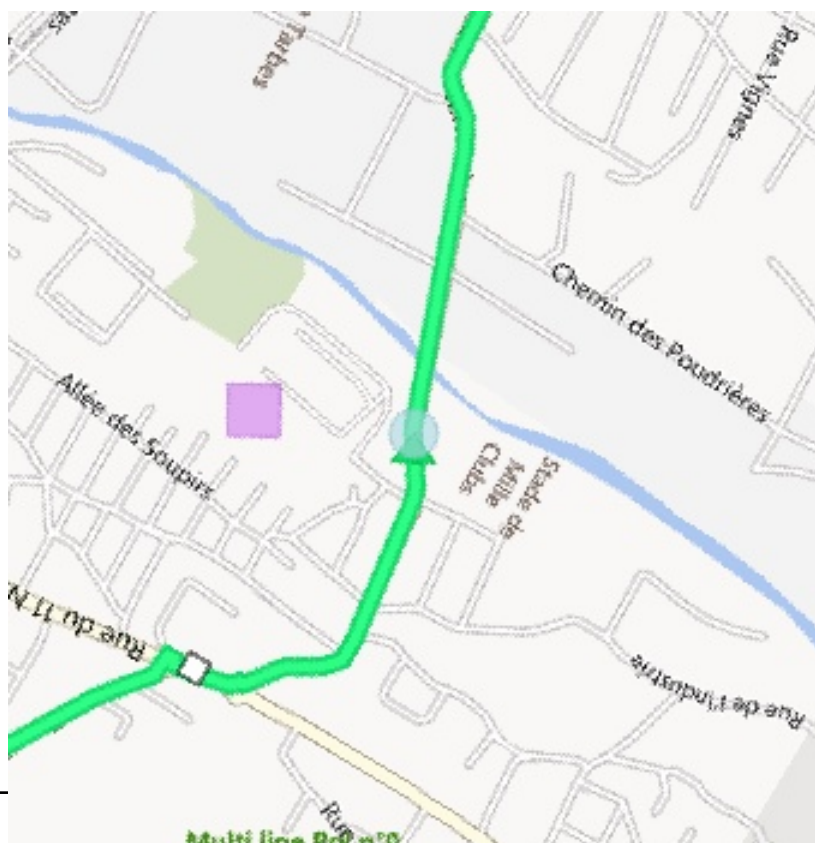
OverSizeForRotation

```
map.OverSizeForRotation := true;

// To allow or not the rotation gesture use the property EnableTouchRotation

map.EnableTouchRotation := true;
```

EnableTouchRotation *permet d'autoriser ou non la rotation par geste.*



Url

La propriété **Url** retourne/accepte une chaîne au format '**#zoom/Latitude/Longitude**'

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

L'assignation d'une valeur à cette propriété déclenche l'événement **OnBeforeUrl**(sender : TObject; var Url:string) qui peut vous permettre de modifier l'url, et même d'annuler le changement en retournant une chaîne vide.

Vous pouvez passer un tel lien dans une **InfoWindow**

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

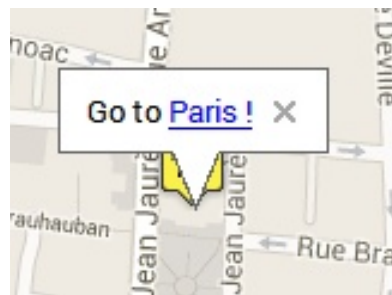


Fig. 65 Url Link

Si vous passez une url "classique" elle s'ouvrira dans votre navigateur par défaut.

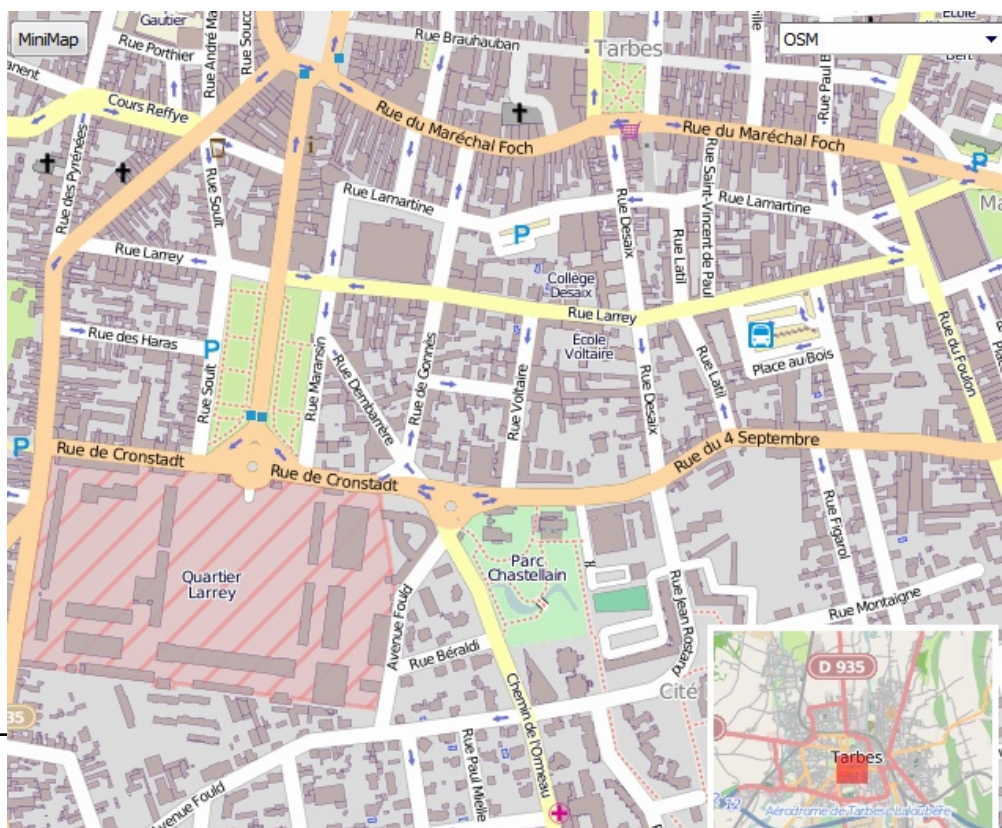
Distance

La fonction **DistanceFrom**(const dLatitudeStart,dLongitudeStart,dLatitudeEnd,dLongitudeEnd:double):double permet de calculer la distance entre 2 points, le résultat est en Kilomètres.

Angle par rapport au Nord

La fonction **Bearing**(const dLatitudeStart,dLongitudeStart,dLatitudeEnd,dLongitudeEnd:double):integer vous donne l'angle, de 0° à 360°, pour la direction allant du point **dLatitudeStart,dLongitudeStart** au point **dLatitudeEnd,dLongitudeEnd**

MiniMap



Pour afficher une mini carte navigable dans votre carte principale il vous suffit d'ajouter l'unité **uecNativeMiniMap** à votre fiche ainsi que les lignes suivantes.

```
// 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;
```

Vous n'avez pas à libérer votre minimap, elle le sera automatiquement lors de la destruction de la carte à laquelle elle est rattachée.

Vous pouvez modifier le coin d'ancrage avec la propriété **ancragePosition** (apTopLeft, apTopRight, apBottomLeft, apBottomRight)

XMargin et **YMargin** permettent d'ajuster la position par rapport aux bords

BorderColor et **BorderSize** modifie la couleur et taille de la bordure entourant la mini carte.

Barre d'échelle

Pour afficher l'échelle il vous faut ajouter l'unité **uecNativeScaleMap** et

les lignes suivantes.

```

procedure TForm1.FormCreate(Sender: TObject);
begin

    FECNativeScaleMap := TECNativeScaleMap.create ;
    FECNativeScaleMap.Map := map;

end;

procedure TForm1.FormDestroy(Sender: TObject);
begin

    FECNativeScaleMap.free;

end;

```



Fig. 67 Scale bar

Les propriétés suivantes sont disponible :

AnchorPosition (apTopLeft, apTopRight, apBottomLeft, apBottomRight)

XMargin et **YMargin** pour ajuster par rapport aux coins

BarSize finesse de la barre (defaut 2)

Color couleur (defaut black)

MaxWidth longueur maximale de la barre en pixels (defaut 80)

MesureSystem système de mesure (msMetric, mslImperial)
defaut msMetric

Shadow ajoute une ombre (defaut true)

Outil de mesure

Vous pouvez utiliser un outil de mesure en ajoutant l'unité **uecNativeMeasureMap** et les lignes

```

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;
  
```



Les propriétés suivantes sont disponible :

Color couleur de la ligne default noir

MeasureSystem unité de mesure (msMetric, msImperial)

default msMetric

Distance : double la distance dans l'unité choisie

DistanceLabel : string la distance en texte (celle qui s'affiche à l'écran)

ShowDistance permet d'afficher ou non la distance à l'écran (default true)

Hint le texte d'information lorsqu'on laisse la souris sur la ligne

OnChange : TNotifyEvent déclenché lorsque la distance change

Observateur

Un observateur vous permet d'être averti lorsqu'un événement particulier à lieux que cela soit un déplacement de la carte ou d'un élément.

Cela a l'avantage de ne pas bloquer les événements principaux de la carte et des shapes tout en les démultipliant.

```
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 OnMapMouseClick : 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;

```

Vous pouvez enregistrer autant d'observateurs que vous le souhaitez.

```

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;

```

Les **layers** utilisent ce système pour réagir au changement de position.

Les tuiles sont d'abord recherchées dans le cache mémoire (les tuiles affichées il y a peu), le cache local, l'archive locale puis sur internet si besoin.

Cache local

En attribuant un répertoire à la propriété **LocalCache** les tuiles téléchargées depuis internet sont enregistrées en local et sont disponibles hors connexion.

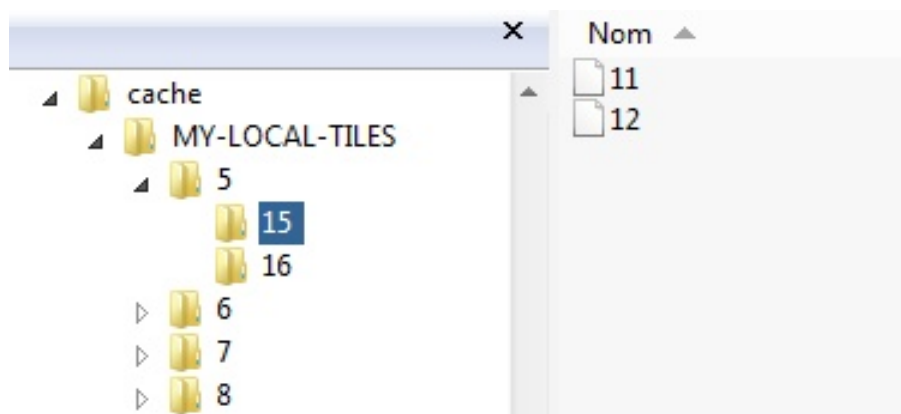


Fig. 69 My local tiles

Utilisez la propriété **MaxDayInCache** pour spécifier la durée de retention dans le cache, 0 pour un cache infini, par défaut 30 jours.

```
map.MaxDayInCache := 7; // max 7 days
```

Archive Locale

Une archive locale correspond au cache local mis dans un Zip, cela simplifie le déploiement des tuiles et des autres fichiers.

Pour améliorer la vitesse les tuiles sont extraites de l'archive et placées dans le cache locale lors de la première demande, vous devez donc définir un cache local pour utiliser une archive.

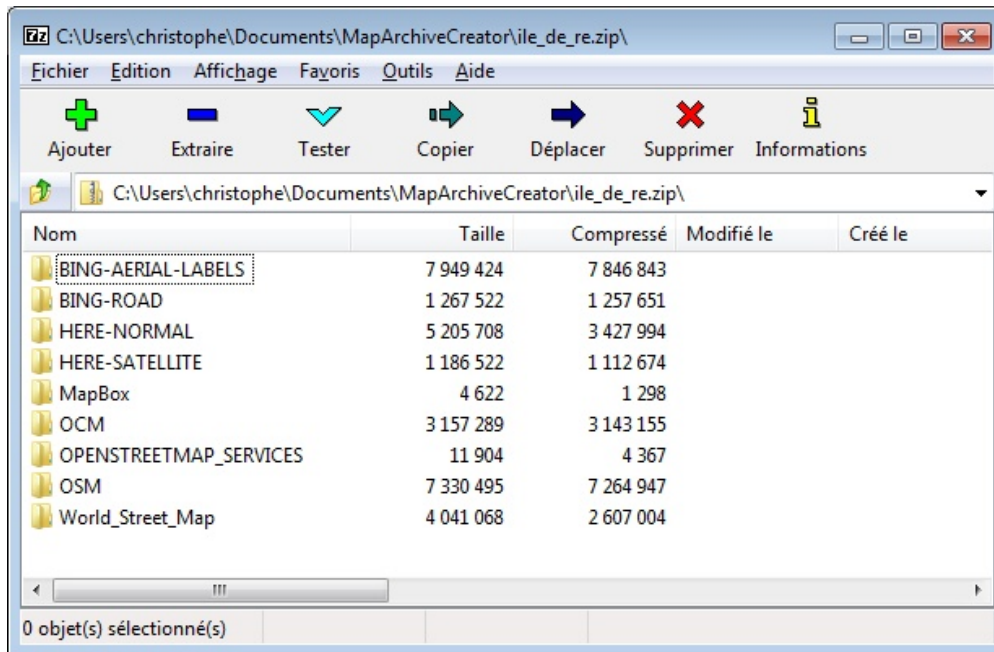


Fig. 70 Archive ile de ré

```
map.LocalCache
:= TPath.Combine(TPath.GetSharedDocumentsPath, 'cache');
map.TileServer := tsOSM;
map.LocalArchive := ExtractfilePath(ParamStr(0))+
'ile_de_re.zip';
```

Si des routes ou des géolocations d'adresses sont enregistrées dans l'archive la récupération est totalement transparente

```
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');
```

Vous pouvez y stocker vos images ou des fichiers de données (kml, geojson etc...), pour les charger il suffira de faire débiter le nom du fichier par /

```
// load data
map.Shapes.LoadFromFile('/DATA/tdf.kml');
map.LoadFromFile('/DATA/tarbes.txt');
// load image in marker
marker.filename := '/IMAGE/node.png';
```

Utilisez MapArchive pour manipuler directement votre archive et extraire vos autres données.

```
m := TMemoryStream.Create;
try

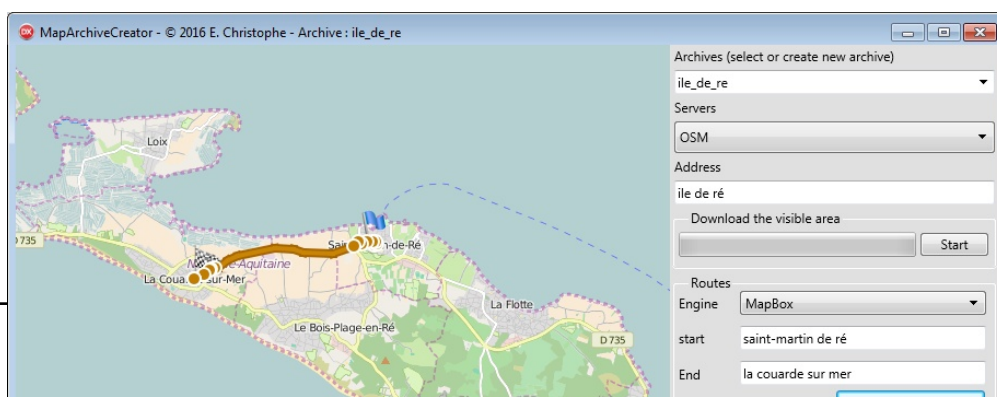
    map.MapArchive.ReadStream('DATA/mydata.txt',m);

    memol.Lines.LoadFromStream(m);

finally
    m.Free;
end;
```

MapArchiveCreator

Ce petit utilitaire permet de créer vos archives, vous pouvez télécharger des zones de tuiles, enregistrer des routes, ajouter des images et des fichiers.



Vous pouvez interdire toute connexion internet pour les tuiles et le geocodage en utilisant la propriété `OnlyLocal`

```
// tiles and geocoding only uses local cache and local archive
map.onlyLocal := true;
```

Télécharger une zone complète

La classe **TECDownloadTiles** vous permet de télécharger en arrière-plan une zone complète, la demo **EcNativeMapFiremonkeyDemo** vous montre comment précharger la zone visible à l'écran.

```
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 ECNativeMapFiremokeyDemo for complete use
```

Assurez-vous que votre fournisseur de tuiles autorise cela !

Remplir le Stream des tuiles

Vous pouvez aussi directement retourner un stream contenant le jpeg ou le png de vos tuiles, utile si vous avez vos tuiles dans une base de données.

```
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;
```

Vous pouvez aussi vous en servir pour utiliser un de tuiles distant .

Support des tuiles vectorielles geoJSON

MapZen fournit gratuitement des [tuiles vectorielles](#) à partir des données d'[OpenStreetMap](#)

Vous devez obtenir une clef gratuite puis renseigner la propriété `VectorMapZenKey` de votre carte.

ATTENTION: MapZen n'existe plus, mais `TECNativeMap` supporte toujours leurs formats car leurs outils sont disponibles en Open Source.

```
// connect your key for use vector tiles
map.VectorMapZenKey := 'vector-tiles-XXXXX';
```

TECNativeMap va donc ajouter un nouveau [serveur](#) `tsVectorMapZen`, les tuiles seront au format [geoJSON](#)

Dans ce format les tuiles peuvent rapidement être très lourdes, 300Ko voir +500ko, surtout dans les zoom <16

Pour limiter le poids on sélectionne [les layers](#) en fonction du 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'

Bien entendu cela n'est pas encore définitif, et vous pourrez toujours faire un autre choix

L'affichage et le téléchargement des tuiles n'est pas plus rapide que des tuiles bitmaps, au contraire, mais le grand avantage c'est que vous avez accès à beaucoup plus de données, certains [tags OSM](#) sont disponibles en plus des données géométriques, par exemple la population d'une ville, le numéro des maisons etc.

Vous pouvez aussi utiliser le layer XAPI pour afficher les données d'OpenStreetMap.

Propriétés

Pour pouvoir gérer ces nouvelles données **Properties** et **PropertyValue** sont ajoutés aux [TECShape](#)

Properties est rempli avec le champ "properties" des tuiles, au format "nom:valeur", chaque couple est séparé par un retour chariot.

PropertyValue[Name] vous permet d'accéder à la valeur de la propriété **Name**

Vous pouvez les utiliser librement pour stocker vos propres données.

Feuille de styles

Pour pouvoir afficher correctement toutes ses données vectorielles il devient obligatoire de pouvoir utiliser des styles.

TECNativeMap dispose donc de la propriété **Styles**

```
map.Styles.LoadFromFile(your_filename);
map.Styles.Rules := all_your_rules;
map.Styles.addRule(your_rule);
```

Une règle se décompose en un sélecteur et une liste de propriétés placées entre { }

```
Selecteur {prop1:value1;prop2:value2}
```

Sélecteur de style

Le sélecteur peut être le nom d'un **Groupe**, un type d'élément (**marker**, **poi**, **line** or **polygone**), le nom d'une propriété de l'élément ou un ensemble de Property:value

Vous pouvez associer ces diverses possibilités (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 {}
```

Vous pouvez appliquer les mêmes propriétés à plusieurs sélecteurs en les séparant par une virgule

```
Select1 , Select2, Select3 {...}
```

Propriété de style

Vous pouvez utiliser les propriétés :

```
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);
```

Vous pouvez définir les couleurs avec la syntaxe #RRGGBB ou \$int_value ou leur nom (red, black, green etc.)

Vous pouvez aussi composer une couleur a partir de deux et d'un pourcentage de mélange

```
// create a color by mixing
map.styles.addRule('.line {color:gradient(Red,Yellow,0.6)'});
```

Exemple d'une feuille de styles

```
.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;}
```



Propriété conditionnelle

Vous pouvez appliquer une règle en fonction de la valeur d'une propriété, pour cela utilisez la syntaxe **if:property**

Avec les règles suivantes, les villages avec une population de moins de 5000 habitants apparaissent en vert, ceux avec une population supérieure à 6000 en bleu.

```
.kind:village {if:population<5000;fontsize:12;color:green;}  
.kind:village {if:population>6000;fontsize:12;color:blue;}
```



Fig. 73 green < 5000 - blue > 6000

Les comparateurs = , !=, > et sont disponibles.

Vous pouvez aussi tester des propriétés "spéciales"

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) permet de détecter si un polygone est un trou dans un autre polygone, TECNativeMap ne prend pas en charge les polygones à trous, ce test permet au minimum d'attribuer une couleur pour les trous.

```
.polygone.kind:building {if:hole=true;fcolor:silver;fopacity:100;zindex:0;}
```

Règle en fonction du Zoom

Vous pouvez définir le ou les zooms pour lesquels la règle s'applique, il suffit d'ajouter en fin de sélecteur [zoom valeur] ou [zoom valeur1,valeur2] , [zoom *] indique que la règle s'applique pour tous les 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];}
```

Multiples valeurs en fonction du zoom

Pour les propriétés **StyleIcon**, **FontSize**, **Scale** et **Weight** vous pouvez définir plusieurs valeurs qui dépendent du niveau de zoom

La syntaxe est du type **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}
```

Définir un nom pour une valeur

Vous pouvez nommer vos valeurs et utiliser le nom à la place de la valeur.

```
@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}
```

Vous pouvez utiliser les styles sans utiliser les tuiles vectorielles, ils s'appliquent à tous vos éléments

L'unité **uecOSMStyles_standard** déclare la constante **UEC_OSM_STYLESHEET**, ce sont les styles utilisés dans la demo **OSMViewer**

```
// see uecOSMStyles_standard
map.styles.Rules := UEC_OSM_STYLESHEET;
```

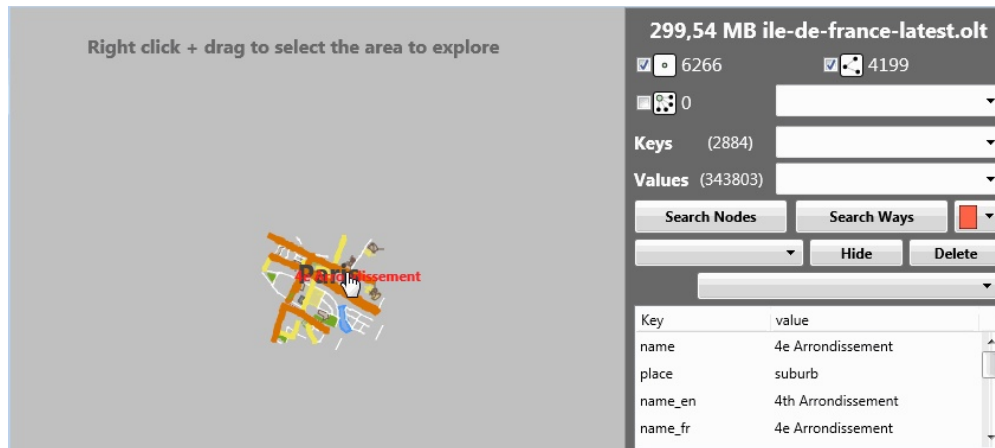


Fig. 74 Demo OSMViewer

Éléments clickables

Chaque élément des tuiles est clickable, vous pouvez intercepter le click avec les mêmes évènements que pour vos propres éléments (OnShapeClick, ShapeRightClick et OnShapeLongClick).

Ces évènements ne sont pas restreints au niveau des tuiles, ils sont répercutés à votre carte et vous avez un OnMapClick après un OnShapeClick !

Recherche

Vous pouvez utiliser les mêmes [fonctions de recherche](#) que pour vos propres éléments.

```
// 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;

```

Cacher les tuiles vectorielles

Vous pouvez ne pas afficher les tuiles vectorielles mais continuer d'utiliser les données (click et recherche)

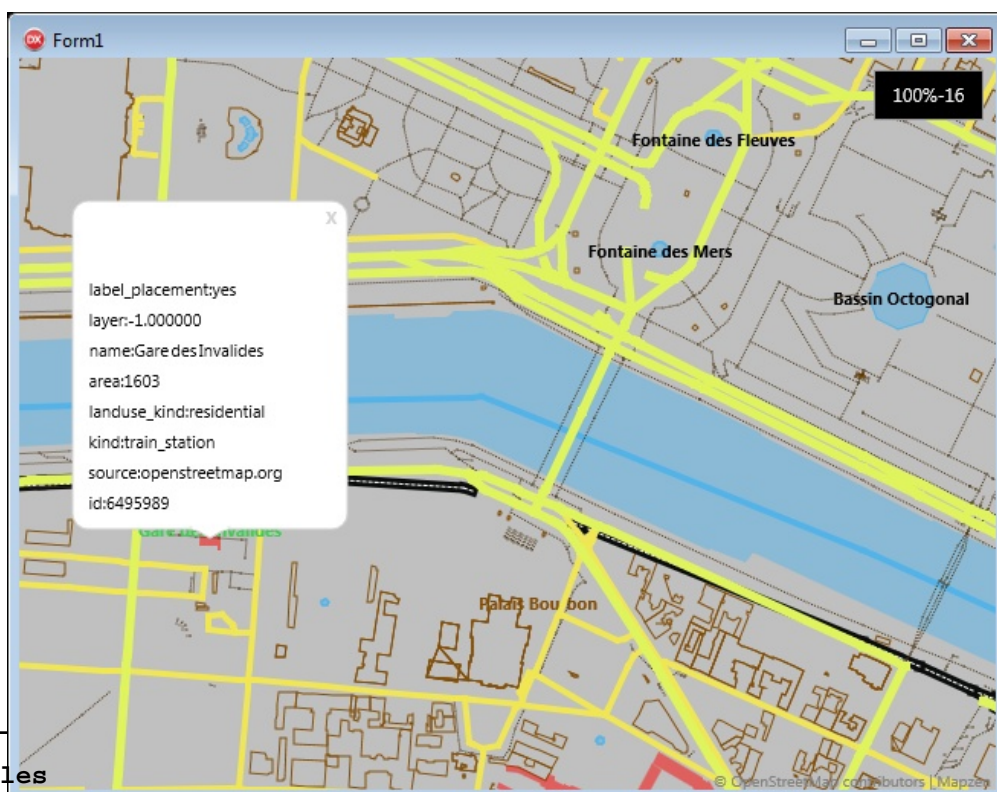
```

// add vector tiles overlay
map.AddOverlayTileServer(tsVectorMapZen);
// hide the tiles but the data is always available
map.DrawVectorTiles := false;

```

Demos

Pour vous faire une idée du résultat vous pouvez télécharger [une demo pour Windows](#) et une [pour Android](#) (dézippez et installez l'apk)



Tous les éléments ne sont pas stylés, c'est juste le minimum pour que cela soit lisible (je n'ai pas dit beau !)

vous avez la possibilité de connecter plusieurs TECNativeMap sur votre carte principale, vous pourrez alors obtenir autant de vue différente.

Les vues sont indépendantes les unes des autres mais elles utilisent toutes les éléments de la carte principales, les données ne sont pas dupliqués elles n'existent que dans la carte principale.

Si vous ajoutez des éléments dans une des vues secondaires, ils sont en fait intégrés dans la vue principale.

Toute modification des données d'une des vues est répercutés immédiatement sur l'ensemble des cartes

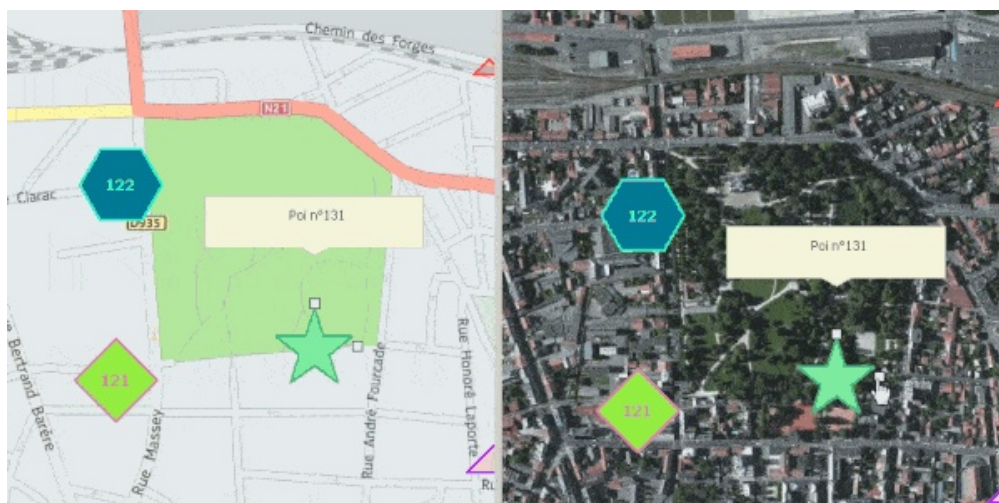


Fig. 76 A gauche la carte principale - à droite une vue secondaire

```
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

Branchez-vous sur cet événements si vous souhaitez filtrer les éléments affichés dans la vue

Exemple, on ne veux voir que des éléments de type **TECShapePOI** en forme d'étoile

```
// 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;
```


Par défaut TECNativeMap utilise son propre serveur de tuiles mais nous pouvons le déplacer sur un poste distant.

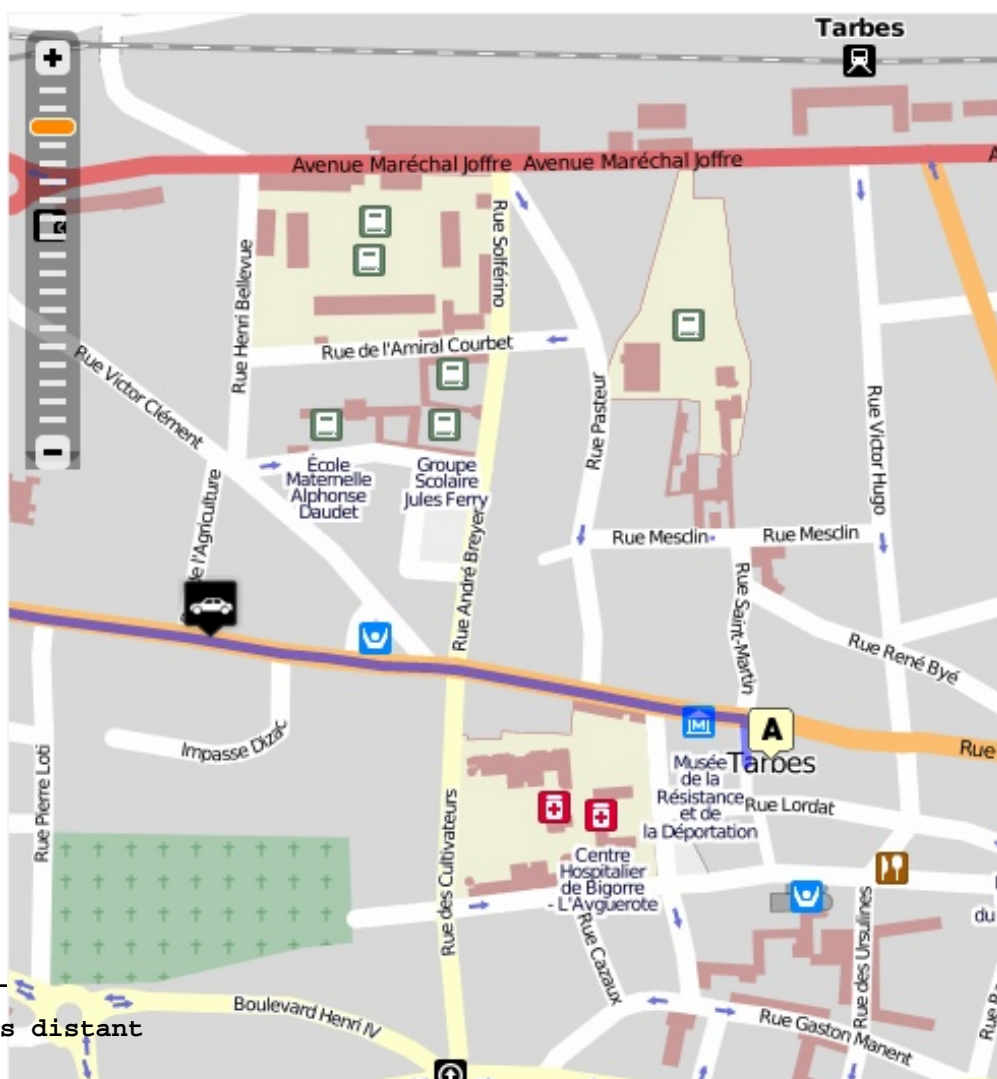
Pour notre démo nous utilisons les [WebSockets d'Andre Mussche](#)

C'est une démo minimaliste qui n'a pas la prétention d'être un tutorial pour l'utilisation des WebSockets !



Télécharger les sources et les
executables des démos

Serveur de tuiles



Vous pouvez indiquer un cache pour limiter les connexions internet

Notre serveur n'a pas besoin du composant `TECNativeMap` juste de la partie serveur pour cela nous allons utiliser la classe **`TNativeMapServer`** de l'unité **`uecNativeTileServer`**

```

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;

// 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;
end;

procedure TForm1.FormCloseQuery(Sender: TObject; var CanClose:
Boolean);
begin
    if CanClose then
        begin
            FMapServer.Free;
        end;
    end;
end;

```

Client

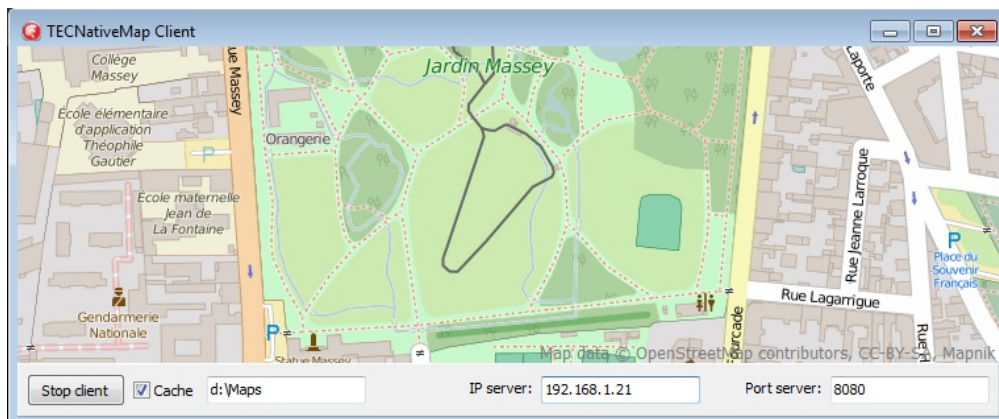


Fig. 78 Client

Le client n'a pas besoin d'avoir un accès internet, il doit juste pouvoir se connecter à un poste local qui lui aura accès à internet et sur lequel

tournera notre serveur.

```

procedure TForm2.FormCreate(Sender: TObject);
begin

    // manual management tiles as stream
    map.TileServerInfo.GetTileStream := GetTileStream;
    // the name will be used 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;
```



Le client peut lui aussi avoir son propre cache de tuiles.

Clefs pour les API

Par défaut TECNativeMap utilise les services d'OpenStreetMap.

Si vous souhaitez utiliser MapQuest vous devez [obtenir une clef auprès de MapQuest](#)

Faites de même pour utiliser les services de [MapZen](#) et [MapBox](#).

Mode hors-ligne

Si vous indiquez un répertoire dans la propriété [LocalCache](#), toutes vos données de géolocalisation seront mise en cache et pourront être réutilisées en mode hors-ligne.

Format de l'adresse

vous bénéficiez des services de géolocalisation au travers de la propriété **GeoLocalise** de type **TECGelocalise**

La propriété **Address** vous donne l'adresse du centre de la carte, elle est de type **string** et est accessible en **lecture/écriture**

Pour rechercher une adresse vous pouvez utiliser un format libre du genre "adresse,ville, code postal".

TECGeolocalise utilise les services d'OpenStreetMap et d'[OpenMapQuest](#)

Vous pouvez aussi utiliser ArcGis au travers des fonctions

```
function TECGeolocalise.ArcGisReverse(const Lat,Lng:double):string;
function TECGeolocalise.ArcGisFind(const data:string;var Lat,Lng:double):boolean;
```

Geolocalisation

Pour obtenir l'adresse d'un point précis vous avez la fonction **GetAddressFromLatLng**
dLatitude,dLongitude:double):string;

Vous pouvez obtenir les diverses parties de l'adresse en utilisant la propriété **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'];
...
```

Vous pouvez obtenir les coordonnées d'une adresse avec la fonction **GetLatLngFromAddress**
sAddress:string;var dLatitude,dLongitude:double):boolean;

En attribuant une valeur à la propriété **Address** vous allez changer la position du centre de votre carte pour la faire correspondre à l'adresse indiquée

Places

TECGeolocalise vous permet d'effectuer des recherches sur des lieux spécifiques dans une zone données, par exemple trouver des

restaurants dans un rayon de 500 mètres.

C'est [Overpass-api](#) qui utilise les données d'[OpenStreetMap](#) qui est utilisé

La propriété **TECGeolocalise.Places.XapiServer** vous permet d'utiliser un autre serveur Xapi que celui de MapQuest

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

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

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

Lancement d'une recherche, tags contient la requête

[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;

```

Lorsque la recherche est terminée, l'évènement **Geolocalise.OnSearch** est déclenché

Chaque lancement de Search efface les résultats d'une précédente recherche

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

Propriété en **lecture/écriture** qui indique le point central de la zone de recherche, elle a la priorité sur les propriétés **Latitude** et **Longitude**

property **Status** : string;

Propriété en **lecture seule** qui retourne une string indiquant le status de la recherche, 'OK' si tout c'est bien passé

Le status est consultable dans l'évènement **OnPlacesSearch**

property ItemDetail : integer;

Propriété en **lecture seule** qui contient l'index du résultat dont on cherche les détails supplémentaires

property **Latitude** : double;

Propriété en **lecture/écriture** qui indique la latitude du point central de la zone de recherche, cela invalide le contenu de la propriété **Adress**

property **Longitude**: double;

Propriété en **lecture/écriture** qui indique la longitude du point central de la zone de recherche, cela invalide le contenu de la propriété **Adress**

property **Radius** : integer;

Propriété en *lecture/écriture* qui indique le rayon de recherche **en mètres**

property **maxResult** : integer;

limite le nombre de résultat pour une recherche dans les données d'OpenStreetMap

property **Searching**: boolean;

Propriété en *lecture seule* qui indique si une recherche est en cours

property **Results**:TECPlaceResults;

Liste des resultats, l'événement **Geolocalise.OnSearch** est déclenché lorsque les résultats sont disponibles

TECPlacesResults

Cette classe gère la liste des resultats retournés par **Search**

procedure **Clear**;

Efface l'ensemble des résultats

function **Count**:integer;

Retourne le nombre de résultat correspondant à la requête

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

Efface le résultat dont on passe l'index

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

Tableau permettant l'accès aux résultats, c'est la propriété par défaut donc vous pouvez y accéder directement par **map.Places.Results[index]** au lieu de **map.Places.Results.Result[index]**

TECPlaceResult

Classe gérant un résultat correspondant à une recherche

procedure **getDetails**;

Effectue une requête supplémentaire sur le résultat pour obtenir des détails

Lorsque les détails sont disponibles l'évènement **OnPlacesDetail** est déclenché.

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

Retourne la valeur de la clef que l'on passe en paramètre

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

Retourne la valeur de la clef que l'on passe en paramètre pour les détails

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

Retourne la clef d'un résultat en fonction de son index

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

Retourne la clef d'un détail en fonction de son index

property **CountResult**:integer;

Retourne le nombre d'élément (clef=valeur) d'un résultat

property **CountDetail**:integer read getCountDetail;

Retourne le nombre d'élément (clef=valeur) d'un détail

property **Latitude** : double;

Latitude du résultat

property **Longitude**: double;

Longitude du résultat

property **RawResult** : string;

Retourne l'ensemble des éléments du resultat sous la forme d'une chaîne constituée de lignes
clef=valeur

property **RawDetail** : string;

Retourne l'ensemble des éléments du détail sous la forme d'une chaîne constituée de lignes
clef=valeur

DemonNativeLocalise

le programme **DemonNativeLocalise** vous montre comment gérer **Places**

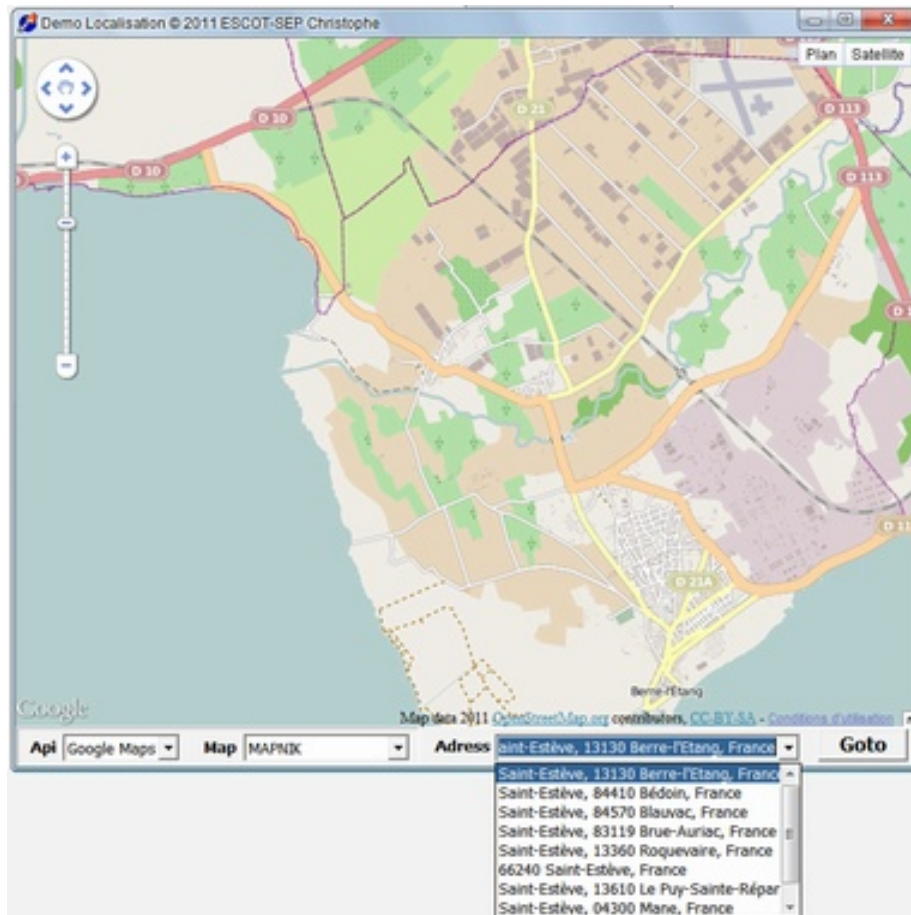


Fig. 79 DemoLocalise

Geofences

Geofence désigne une zone virtuelle qui déclenche une alerte lorsqu'on y entre ou qu'on en sort.

```
function Add(const Lat,Lng:double;const RadiusMeter:integer;const Name:string=""):integer;overload;
```

Ajoute une zone circulaire définie par son point central et son rayon en mètres.

```
map.geofences.add(43.231572,0.080853,50);
```

function **Add**(const dLatLngs: array of double;const Name:string=""):integer;overload;

Ajoute une zone polygonale définie par un tableau contenant les divers points

```
map.geofences.add([lat1,lng1,lat2,lng2,lat3,lng3],"triangle");
```

function **Add**(const Polygone:TECShapePolygone;const Name:string=""):integer;overload;

Ajoute une zone polygonale définie à partir d'un polygone

```
map.geofences.add(map.shapes.polygones[0]);
```

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

Supprimer une geofence

function **Count**:integer;

Retourne le nombre de geofences

function **IndexOf**(value:TECBaseGeofence):integer;

Retourne l'index de la geofence

procedure **Clear**;

Efface toute les geofences

property **Active** : boolean ;

Activer ou non la détection des geofences

property **toTxt**:string ;

Importer / exporter les geofences sous forme textuelle

Par défaut lorsque vous [sauvegardez la carte dans le format texte](#) les geofences sont enregistrés et peuvent être rechargés

property **Geofence**[index : integer]:TECBaseGeofence ;

Retourne une geofence

TECBaseGeofence

property **TECBaseGeofence.Name**:string ;

Nom de la geofence

property **TECBaseGeofence.Active** : boolean

Activer ou non la geofence (active par défaut)

property **TECBaseGeofence.ActiveDuration** : integer

Temps maximal d'activation (millisecondes) 0 pour un temps infini (0 par défaut)

```
// Activate for 3 seconds
map.geofences.geofence[0].ActiveDuration := 3000;
```

property **Color** : TColor ;

Couleur de la zone

property **HoverColor** : TColor ;

Couleur de la zone lors du survol par la souris

property **Item** : TObject;

Vous pouvez utiliser cette propriété pour stocker vos données

property **Tag** : integer;

Vous pouvez utiliser cette propriété pour stocker vos données

property **TECBaseGeofence.Shapes** : TECShapesList

Liste des éléments qui sont actuellement dans la geofence

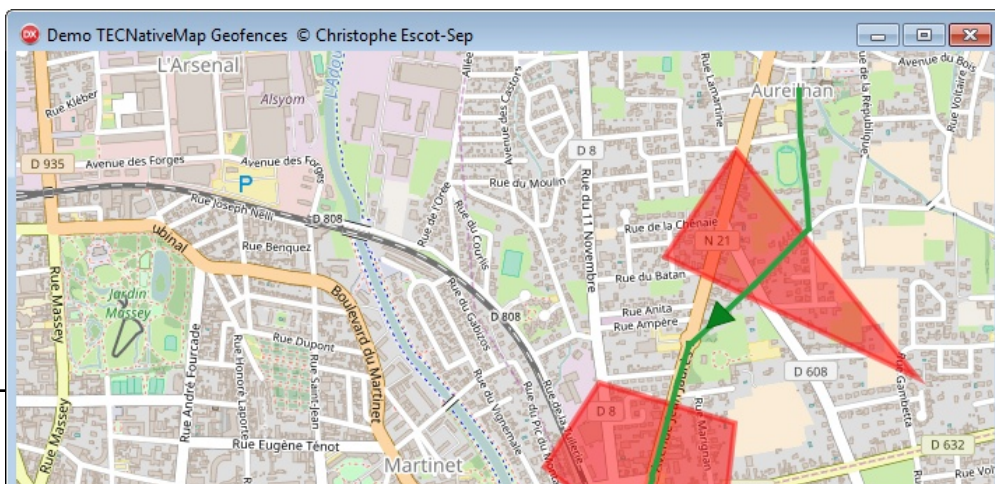
Événements Geofences

Pour réagir branchez-vous sur les événements **OnEnterGeofence** et **OnLeaveGeofence** de TECNativeMap.

```
...
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;
```

La détection se fait lorsqu'un **shape** est déplacé, le test est effectué sur le point de localisation (sa latitude et sa longitude) pas sur la surface réelle de l'objet.



Boundary

Boundary permet d'obtenir le polygone et les données d'une zone juste en indiquant un point géographique.

function

Administrative(const lat,lng:double;Level:integer=0):boolean;

Trouve la zone administrative dont on passe le niveau (de 2 à 10), en indiquant 0 le niveau est déterminé en fonction du zoom

Retourne true si trouvé

```

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**

Permet de contrôler le niveau administratif en fonction du zoom

Par défaut :

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;**

Fonction générique qui vous permet de définir la zone à trouver.

```
// like administrative level 8
map.boundary.Filter(43.2332,0.0736, 'admin_level'="8" )
);
```

property **Id**:int64;

Id OSM de la zone trouvée

property **Tags**:TStringlist

Contient la liste des tags OSM de la zone, sous la forme key=value

property **OverPassUrl**:string;

OverPass est utilisé pour trouver la zone (par défaut <https://overpass-api.de/api/interpreter>)

Pour obtenir le polygone à partir de l'id de la zone, on se connecte à polygons.openstreetmap.fr

Pour modifier cela assignez une procédure à **GetPolygoneFromID**

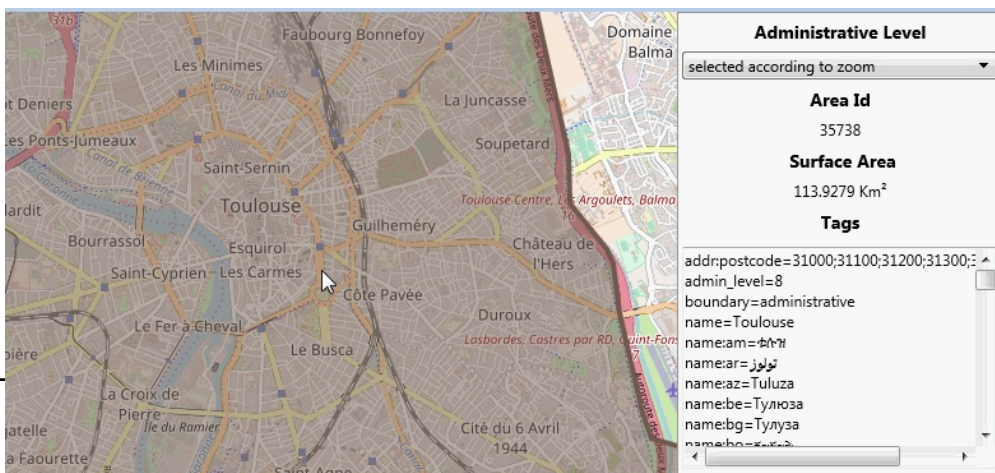
```
// set nil for use polygons.openstreetmap.fr
map.Boudary.GetPolygoneFromID := myGetPoygonFromId;

procedure TForm.myGetPoygonFromId(const id:int64;var JSON:
string);
begin

    // here retourne yours polygones in json format

    JSON := ...
end;
```

Consultez DemoNativeBoundaryArea pour une démo



GeoHash

Geohash est un système de géocodage du domaine public inventé par Gustavo Niemeyer, qui code un emplacement géographique en une courte chaîne de lettres et de chiffres.

Plus d'informations sur [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' );
```

TECNativeMap permet d'enregistrer et de recharger la totalité de ses données dans un simple fichier texte, cela comprend non seulement les paramètres du composant et des vues mais aussi les données cartographiques

Vous pouvez aussi importer/exporter vos données aux formats [GPX](#) , [GeoJSON](#) et [KML](#)

Sauvegarde/Restauration

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

Enregistre la carte dans un fichier texte.

Utilisez les extensions .gpx , .kml pour spécifier un format, autrement c'est le format interne d'ECNativeMap qui sera employé

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

Charge la carte avec un fichier texte

Utilisez les extensions .osm, .olt, .gpx, .json , .kml pour spécifier un format, autrement c'est le format interne d'ECNativeMap qui sera employé

LoadFromFile peut télécharger les fichiers sur internet

property **toGPX** : string;

Propriété en *lecture/écriture* qui donne accès aux données de la carte dans le format GPX.

Cette propriété est utilisée par **SaveToFile**, **LoadFromFile**.

property **toTxt** : string;

Propriété en **lecture/écriture** qui donne accès aux données de la carte dans un format texte.

Cette propriété est utilisée par **SaveToFile**, **LoadFromFile**.

property **ToKml** : string;

Propriété en **lecture / écriture** qui retourne les éléments géographique de la carte au [format Kml](#)

property **toGeoJson** : string;

Propriété en **lecture/écriture** qui donne accès aux données de la carte dans le format GeoJson.

Vous accédez aux données du champ "properties" avec **TECShape.Properties**, reportez vous au chapitre sur les [Tuiles vectorielles](#) pour apprendre à styler vos éléments.

Cette propriété est utilisée par **SaveToFile**, **LoadFromFile**.

Événements

Pendant le chargement de la carte l'événement **OnLoadShapes**(sender: TObject;const ShapeType:string;const index,max:integer;var cancel:boolean) se déclenche pour chaque élément ajouté.

ShapeType peut prendre les valeurs '[TECShapePOI](#)', '[TECShapeMarker](#)', '[TECShapeLine](#)' , '[TECShapePolygone](#)' et '[TECShapeInfoWindow](#)'.

Index indique le numéro de l'élément actuellement chargé et **Max** le nombre total d'éléments à charger.

Vous pouvez abandonner le chargement en basculant **cancel** à true

Lorsque le fichier a été chargé l'événement **OnLoad**(sender: TObject;const GroupName:string;const FinishLoading : boolean)

GroupName contient le nom du groupe qui a été chargé, vide pour un chargement direct de la carte (groupe par défaut)

FinishLoading vaut true si le fichier a été chargé en totalité et false s'il a été abandonné en basculant **cancel** à true

OSM XML

Vous pouvez directement charger un fichier au format OSM XML ou olt avec LoadFromFile.

Les groupes disposent en plus des fonctions

LoadFromOSMStream et **LoadFromOSMString**.

La classe **TOSMFile** de l'unité **uecOSM** vous permet de travailler avec des fichiers au format **OSM XML** (format de base d'OpenStreetMap).

Vous pouvez télécharger des régions du monde dans ce format sur le site geofabrik.de, prenez les fichiers **.osm.bz2** et dézippez-les

Mais vous pouvez aussi obtenir ces données directement depuis **TECNativeMap** en utilisant **OverPass API**

OverPassApi

La propriété `OverPassApi` vous permet d'utiliser [Overpass API](#) pour obtenir des données d'OpenStreetMap

Par défaut le serveur utilisé est <https://overpass-api.de/api/interpreter>, pour le changer utilisez la propriété `Url`

```
map.OverPassApi.Url :=
  'https://overpass.kumi.systems/api/interpreter';
```

Les requêtes ne sont pas bloquantes, branchez vous sur l'événement **OnData** pour récupérer les données lorsqu'elles sont disponibles.

Utilisez **Query** pour lancer votre requête.

```
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;
```

Vous pouvez utiliser `{{bbox}}` pour limiter la recherche à la zone visible de votre carte, et `{{center}}` pour indiquer le point central de votre carte.



Pour plus d'information sur le langage de script d'OverPassAPI consultez :

wiki.openstreetmap.org/wiki/Overpass_API/Language_Guide

wiki.openstreetmap.org/wiki/Overpass_API/Overpass_QL

Quelques requêtes sont disponibles en standard

StreetNames retourne juste une liste contenant le nom des rues

```
procedure StreetNames(const bbox:string="";const  
OnDataEvent:TECOnOverPass=nil); overload;
```

Utilisez cette version pour définir une zone de recherche différente de la vue actuelle

```
procedure StreetNames(const SWLatitude,SWLongitude,NELatitude,NELongitude:double;const  
OnDataEvent:TECOnOverPass=nil); overload;
```

```
map.OverPassApi.StreetNames;
```

Streets retourne les données OSM pour les rues de la zone choisi (c'est l'équivalent de l'exemple plus haut)

```
procedure Streets(const bbox:string="";const OnDataEvent:TECOnOverPass=nil); overload;  
procedure Streets(const SWLatitude,SWLongitude,NELatitude,NELongitude:double;const  
OnDataEvent:TECOnOverPass=nil); overload;
```

Amenity permet de trouver des points spécifiques

```
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 retourne l'ensemble des données OSM disponibles pour la zone spécifiée

```
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;
```

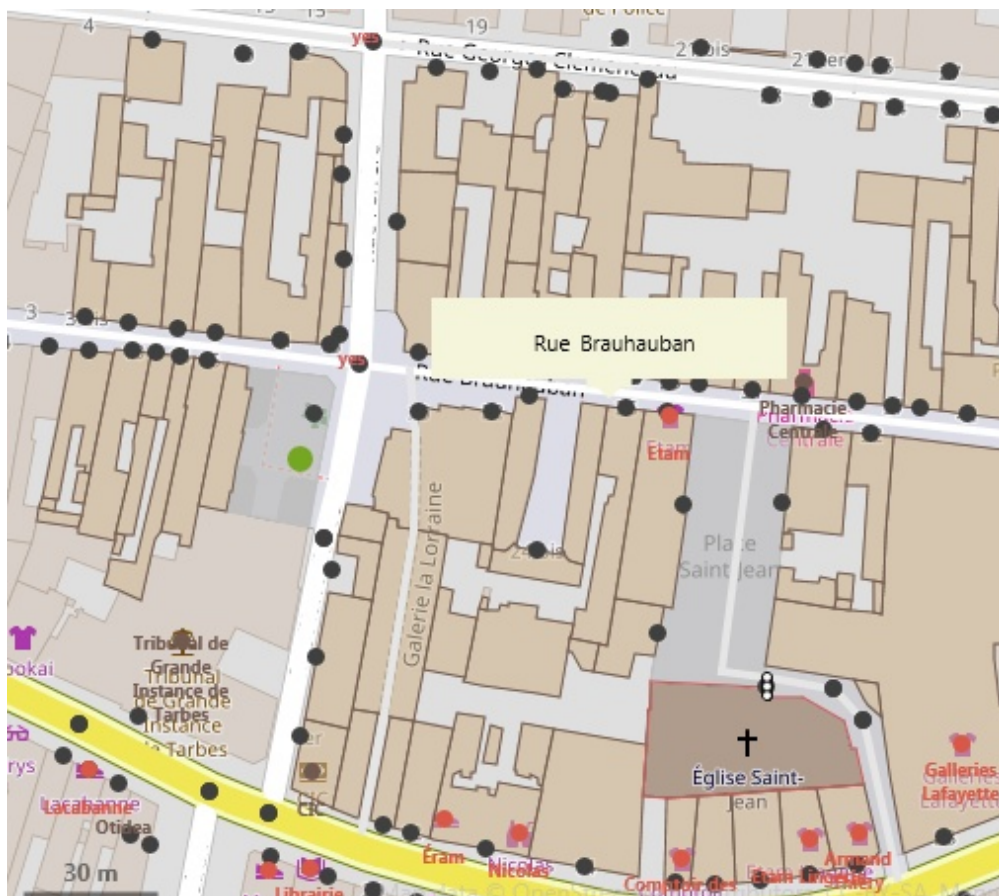


Fig. 83 OverPassApi.Data

OSM XML est volumineux, certains fichiers font plusieurs Giga,
TOSMFile peut utiliser un format interne (.olt) jusqu'a 19 fois plus léger,

10 fois plus rapide à traiter.

Pour convertir simplement un fichier .osm en .olt utiliser la procedure **OSMFileToOLT**
OSMFilename:string;const Notify:TNotifyEvent=nil);

La conversion est effectuée dans un thread et Notify est appelé lorsque le travail est terminé.

```

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 savec in .olt

end;

```

Vous devez utiliser directement TOSMFile pour convertir des portions de fichiers OSM

TOMSFile

function **LoadFromFile**(const sFilename : string) : boolean;
function **LoadFromStream**(const OSMStream : TStream) : boolean;
function **LoadFromString**(const OSMDData : string) : boolean;

Charge un fichier OSM ou OLT, le traitement a lieu dans un Thread donc il ne bloque pas votre programme

property **FilterPrimitive** : TSetPrimitiveOSM ;

Il y a 3 primitives, les nodes, les ways et les relations.

```
// handle all primitive
FOSMFile.FilterPrimitive := [poNode,poWay,poRelation];
```

Si vous souhaitez connaître simplement la zone géographique couverte par le fichier sans charger les données, filtrez tout !

```
FOSMFile.FilterPrimitive := [];
```

property **FilterBounds** : TFilterBounds;

Détermine les zones géographiques qui nous intéressent, les données situées en dehors sont ignorées.

```
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;

Filtre les éléments en fonction de leur clefs et valeurs, les filtres descendent de la classe **TFilterOSM**

ExcludeKeyValue(*const Key, Value : string*),
OnlyKeyValue(*const Key, Value : string*),
ExcludelfKeyExist(*const Key:string*) et **OnlyIfKeyExist**(*const Key:string*) vous simplifie la création de filtres.

```
FOSMFile.FilterWay.clear;
FOSMFile.FilterWay.ExcludeIfKeyExist('building');
FOSMFile.FilterWay.ExcludeKeyValue('leisure','park');
FOSMFile.FilterWay.ExcludeKeyValue('natural','coastline');
```

function Reload : boolean;

Retraite un fichier après avoir modifié les divers filtres, les clefs et les valeurs ne sont pas rechargées cela permet de gagner quelques secondes.

procedure saveToFile(const Filename : string; NotifyEvent : TNotifyEvent = nil);

Vous pouvez sauvegarder dans le format propriétaire .olt ou au [format GeoJSON](#)

```
// save in geojson
FOSMFile.SaveToFile('path_myfile.geojson');
// save in olt
FOSMFile.SaveToFile('path_myfile.olt');
```

Pour sauvegarder dans un thread indiquez un 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;

Listes des éléments contenu dans le fichier après filtrage.

Chaque liste dispose de fonctions permettant d'effectuer des recherches sur ces éléments.

Find(idElement) retourne un élément en fonction de son id

Les diverses fonctions **Search** remplisse la liste SearchResult avec les éléments remplissant les conditions

function **Search**(const Lat, Lng, radiusKM : double) : integer; overload;

Trouve tous les éléments situés dans une zone centrée sur le point Lat,Lng et de rayon radiusKM

function **Search**(const SWLat, SWLng, NELat, NELng : double) : integer; overload;

Trouve tous les éléments de la zone.

function **Search**(const SWLat, SWLng, NELat, NELng : double; const Key, Values : string) : integer; overload;

Trouve tous les éléments de la zone avec une Clef contenant les valeurs spécifiées.

function **Search**(const Key, Values : string) : integer; overload;

Trouve tous les éléments dont la clef contient les valeurs indiquées.

```
// 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;

Trouve tous les éléments possédant cette clef, quelque soit la valeur.

procedure **Clear**;

procedure **Abort**;

Abandonne un chargement

procedure **ToShapes**

(Shapes:TECShapes;NotifyEvent:TNotifyEvent=nil);

Charge les données dans un groupe.

```
// load in default group
FOSMFile.ToShapes(map.Shapes);
```


procedure **FindToShapes**

(Shapes:TECShapes;NotifyEvent:TNotifyEvent=nil);

Charge les données recherchées dans un groupe.

```
// 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;

Retourne le nombre de **tag** pour un élément

function **getKey**(Node:TBaseOSM;const index:integer):string;

Retourne la key dont on passe l'index

function **ReadKey**(Node : TBaseOSM; const KeyName : string) :

string; overload;

Retourne la valeur de la Key

function **ReadKey**(Node : TBaseOSM) : string; overload;

Retourne l'ensemble des paires key/value sous la forme :

Key1=value1#13#10Key2=value2#13#10Keyx=ValueX

function **ReadValuesForKey**(const KeyName : string; valuesList :

TStringList): integer;

Rempli ValuesList avec l'ensemble des valeurs pour une Key

property **Aborted** : boolean;

Indique si le chargement a été interrompu par **Abort**

property **Bounds** : TBoundsLatLng ;

Contient la zone géographique décrite par le fichier

property **FileFormat**: TFileTypeOSM;

Retourne le type de fichier (ftOSM ou ftOLT)

property **Filename** : string ;

property **FileSize** : int64;

property **Keys** : TUniqueStringList;

Liste de la totalité des Key du fichier

property **Values** : TUniqueStringList;

Liste de la totalité des Value du fichier

property Role[Member : TMemberOSM]:string;

Retourne le [role](#) d'un membre d'une [relation](#)

property **Timestamp** : string ;

Retourne la date de la dernière mise à jour du fichier

property **OnRead** : TOnOSMBlockRead;

Pour pouvoir traiter des fichiers de plusieurs Giga la lecture est faite par bloc, cet événement est déclenché à chaque chargement d'un bloc.

property **OnLoaded** : TNotifyEvent;

Déclenché lorsque la totalité du fichier a été traité.

*Consultez **DemoOSMViewer** pour voir une utilisation réelle de TOSMFile*

OSMViewer

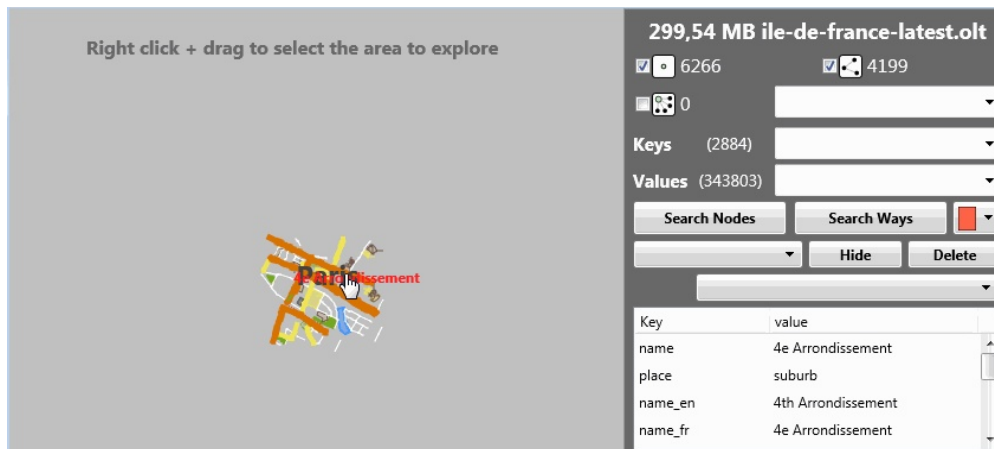
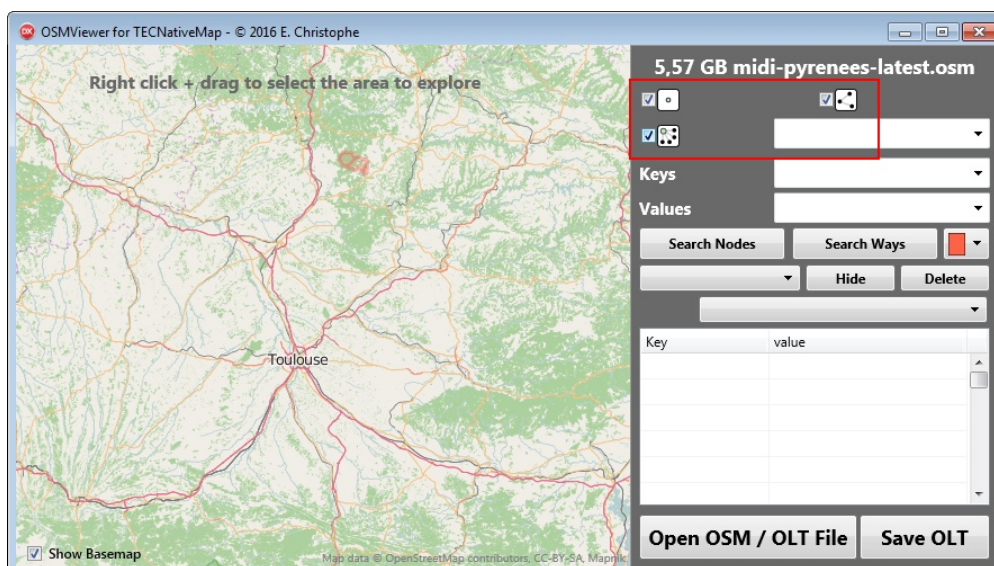


Fig. 84 Demo OSMViewer

Convertir un fichier OSM en OLT

Ouvrez le fichier OSM dans OSMviewer, la zone couverte s'affiche sur la carte

Cochez les primitives que vous souhaitez exporter (nodes, ways et relation) puis cliquez sur **Save OLT** pour lancer la conversion



Suivant la taille du fichier OSM le travail peut durer quelques minutes.

Pour exporter les ways toutes les nodes sont conservée en mémoire il faut avoir 1Mo de mémoire par Go de fichier.

Si vous ne convertissez que les nodes vous pourrez travailler avec des fichiers de 100 Go même si vous n'avez que 8Mo de mémoire.

Faites un clic droit et tout en maintenant le bouton enfoncé déplacez la souris pour définir une zone à explorer.

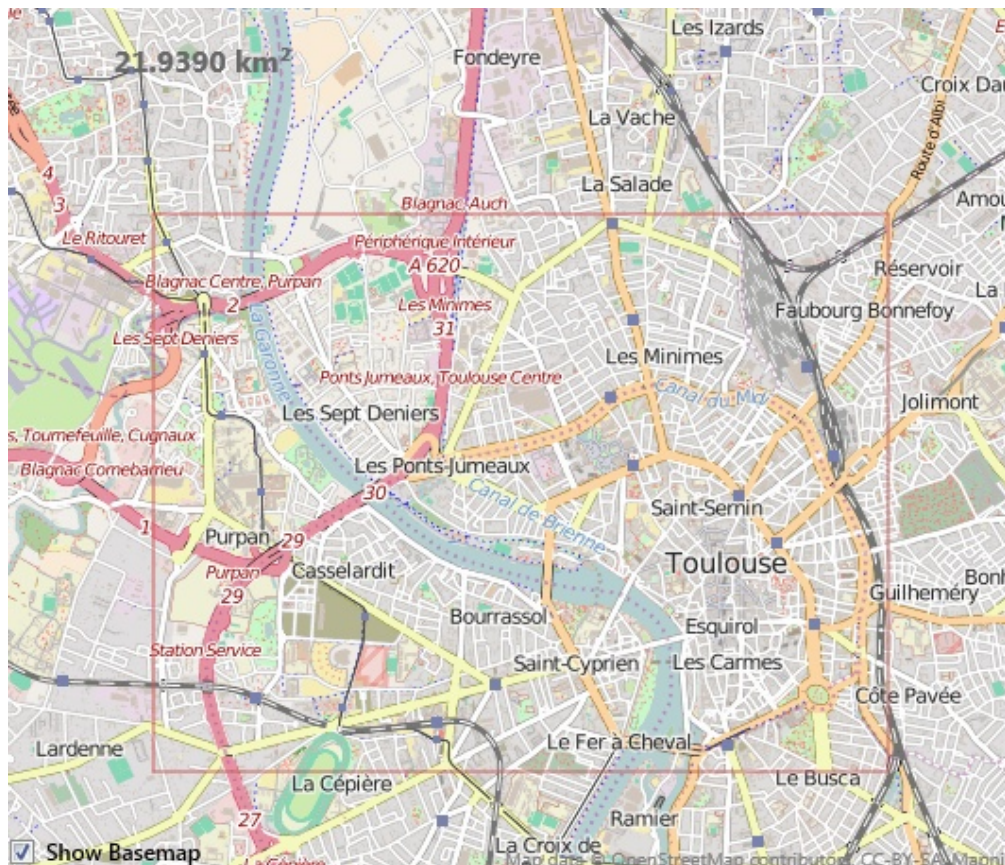


Fig. 86 Sélection d'une zone

Une fois la zone chargée vous aurez le choix entre affichée les données sur la carte ou les sauvegarder en .olt

En chargeant les données vous pouvez effectuer des recherches en fonction des paires key/value

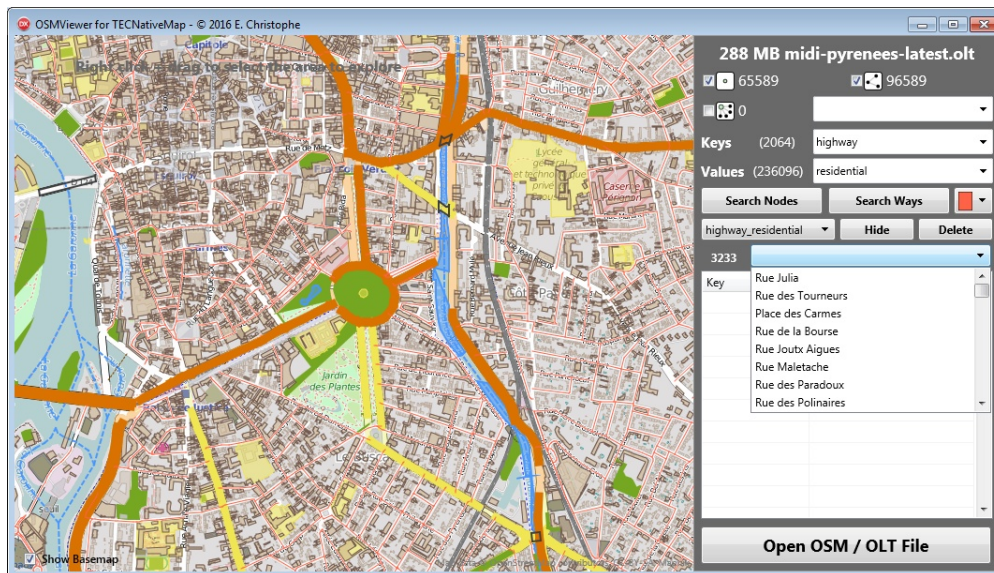


Fig. 87 search highway=residential

Télécharger OSMViewer

Vous pouvez [télécharger un apk pour android](#) d'une mini demo qui affiche les données OSM d'une petite zone.



Capture Image

Vous pouvez enregistrer la zone visible de l'écran avec la propriété **ScreenShot** qui est un TBitmap

```
// Delphi map component TECNativeMap

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

Vous pouvez aussi capturer n'importe quelle zone géographique, même hors-écran, avec la propriété
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
    Screenshot:TBitmap);
begin
    // don't free ScreenShot !
    screenshot.SaveToFile(YourPath+name+'.bmp');
end;

```

La demo Photographer pour Firemonkey vous montre un exemple d'utilisation, vous lancez 8 "drones" pour photographier le terrain tous les x mètres.

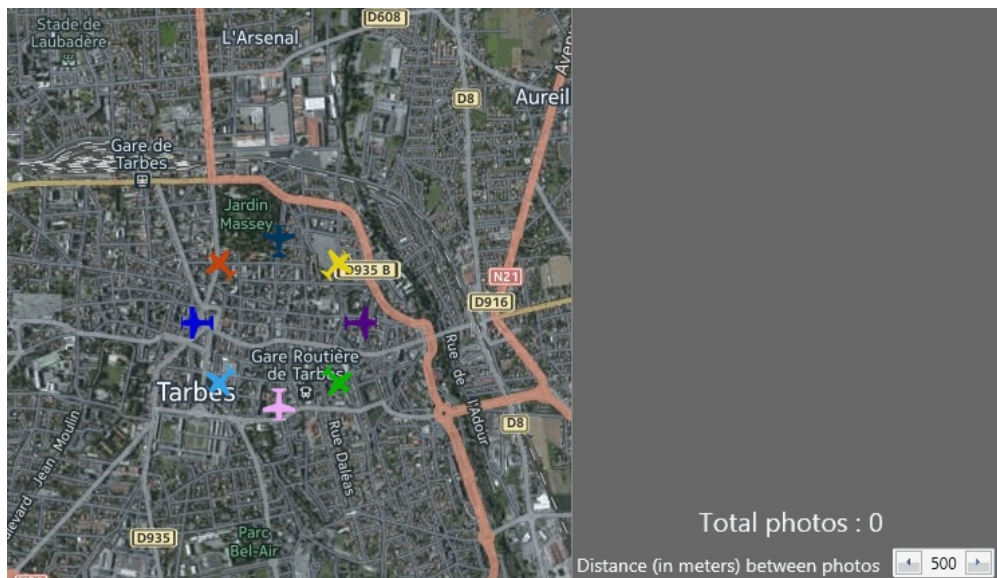


Fig. 89 Photographer

Avec la VCL ou Firemonkey vous pouvez travailler en haute résolution en ajustant la propriété **ScaleFactor**

```
// "retina" mode
map.ScaleFactor := 2.0
```

Si vous travaillez dans une résolution doublée, doublez aussi la taille des images de vos markers et mettez leur propriété Scale à 0 pour qu'ils soient automatiquement mis à la bonne échelle.

```
map.ScaleFactor := 2.0;

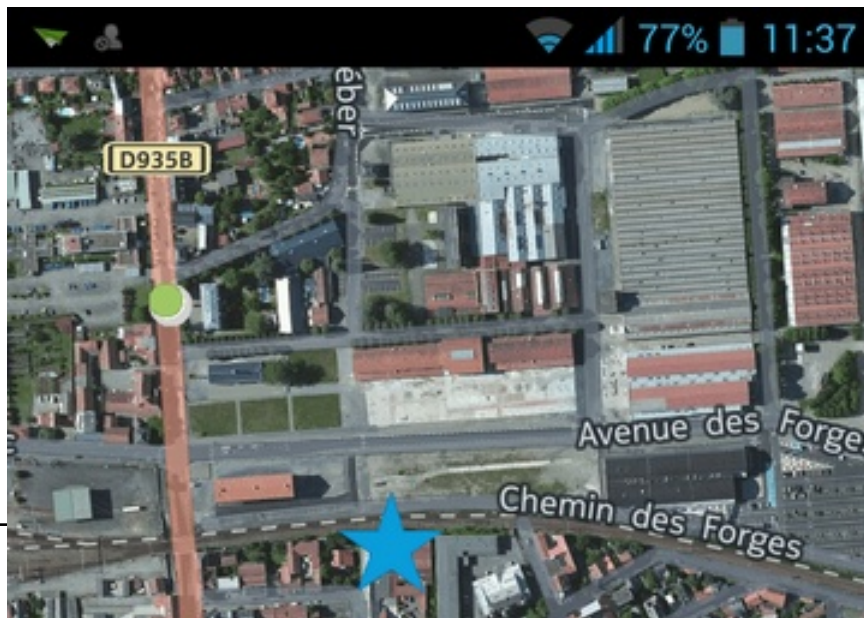
marker := map.AddMarker(latitude, longitude);
marker.Filename := 'image-64x64.png';
marker.Scale     := 0;
```

La version Firemonkey vous permet d'activer le mode "Haute résolution"

```
map.HiRes := true;
```

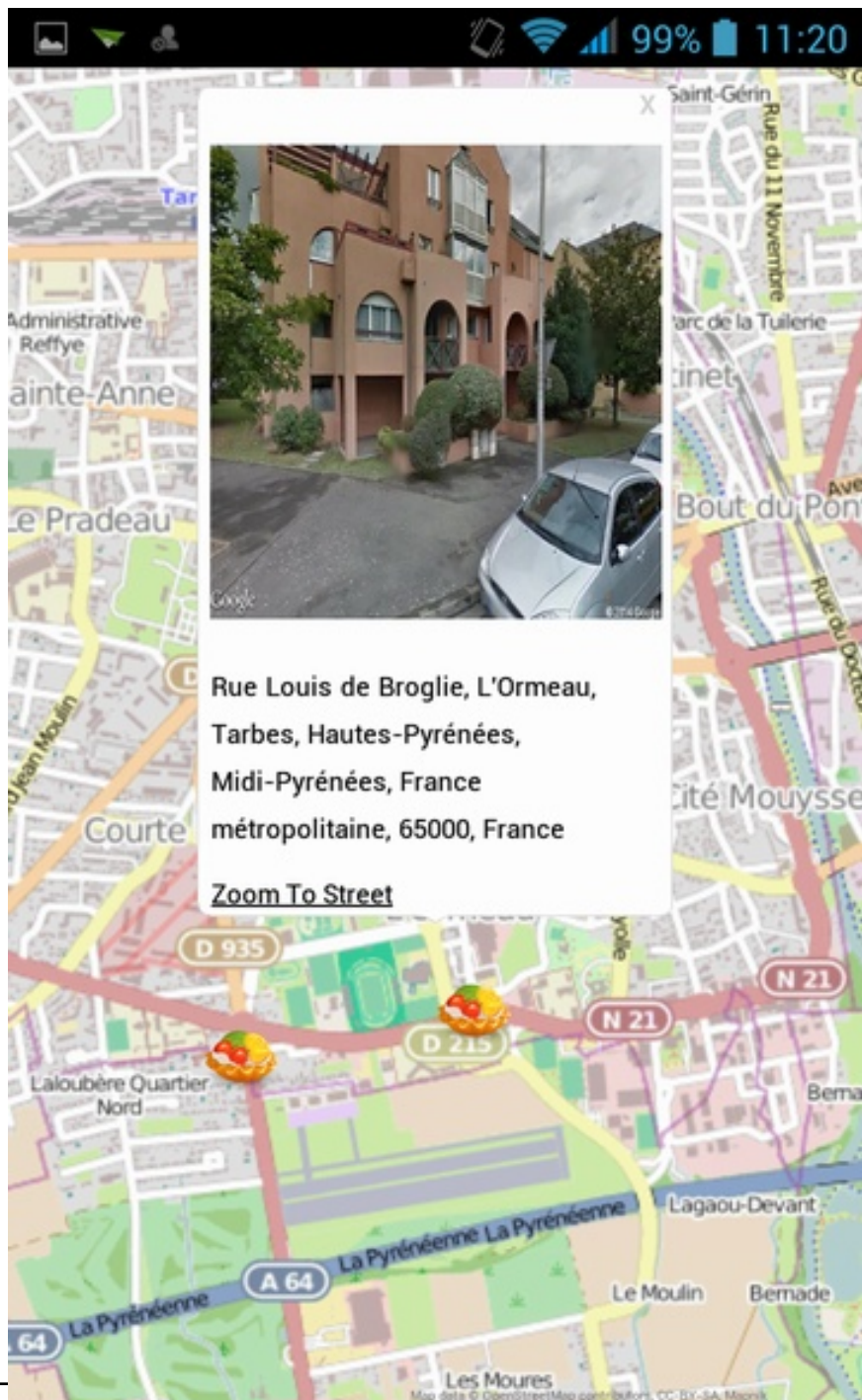
Pour une meilleure visibilité de vos cartes il est préférable d'utiliser des tuiles en 512x512

```
map.HiRes := true;
map.TileServer := tsHereHybrid;
map.TileSize := 512;
```



Si votre fournisseur n'en dispose pas vous pouvez tout de même forcer la taille des tuiles en 512 pixels, elles seront alors automatiquement doublées mais le résultat ne sera pas optimal.

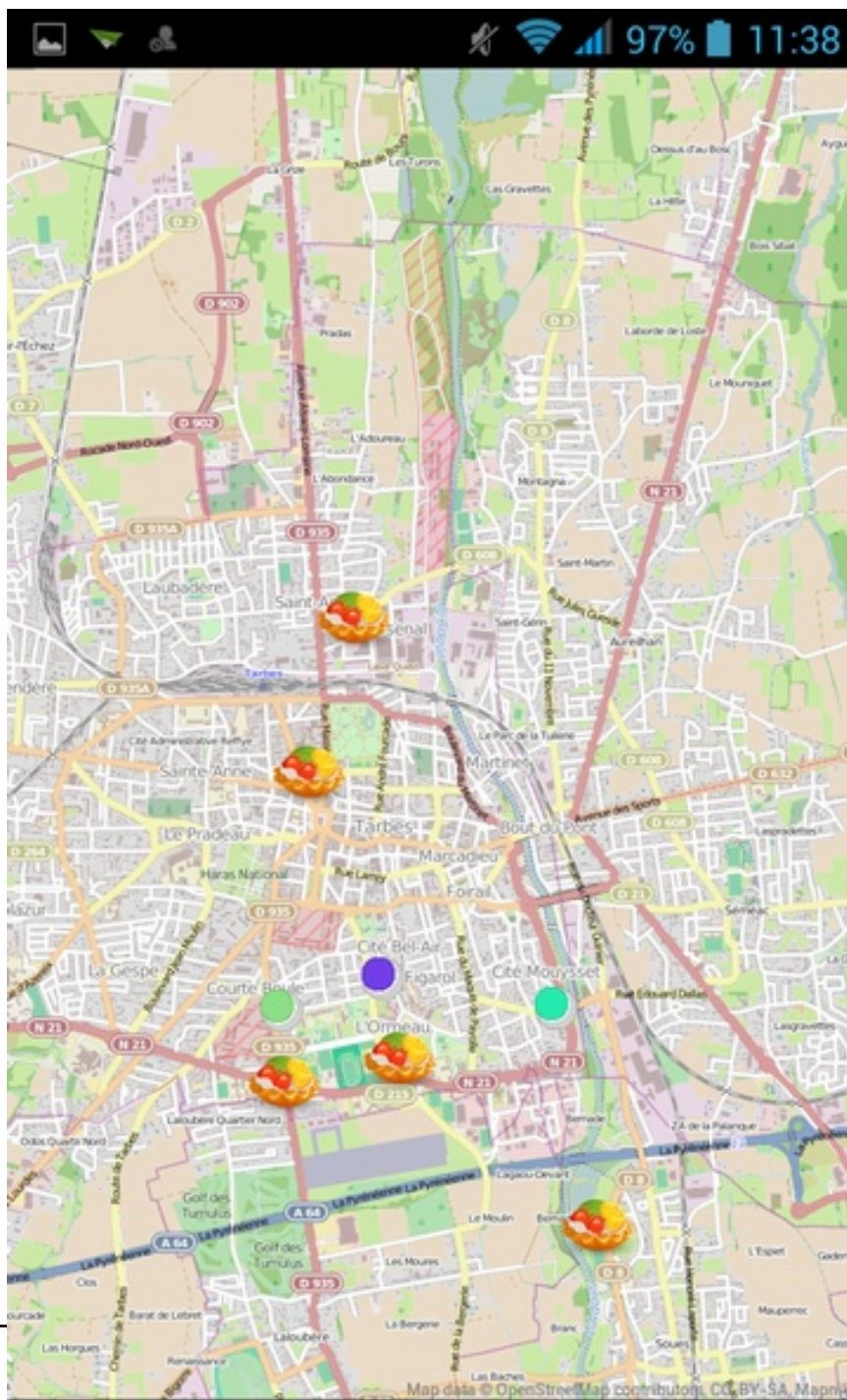
```
map.HiRes      := true;  
map.TileSize   := 512;  
map.TileServer := tsOSM;
```



Si vous gardez des tuiles de 256 pixels en mode haute résolution votre carte sera plus fine mais moins lisible.

L'avantage est qu'une plus grande superficie sera visible.

```
map.HiRes := true;  
map.TileSize := 256;  
map.TileServer := tsOSM;
```



Les [markers](#) sans images, les [TECShapePois](#) définis en pixels, les [InfoWindos](#) et le [layer Panoramio](#) sont automatiquement mis à l'échelle.

Le [layer TECNativePlaceLayer](#) dispose de propriétés pour adapter l'image à la résolution

Pour les Markers avec image il faudra adapter les propriétés **XAnchor**, **YAnchor** et **Filename** en fonction de la résolution (au minimum il faut doubler la taille de l'image en haute résolution)

Pour utiliser les formes vous devez rajouter l'unité
uecNativeShape

Tous les éléments positionnés sur la carte sont accessibles au travers de la propriété **Shapes** de type **TECShapes**, ils descendent tous de la classe **TECShape**

Lorsque vous ajoutez plusieurs éléments à la suite
encadrez-les par le couple **BeginUpdate / EndUpdate**

```
// for optimisation
map.shapes.BeginUpdate;

map.Shapes.Markers.add(lat1,lng1);
map.shapes.Markers.add(lat2,lng2);
...
map.shapes.EndUpdate;
```

Styles

Au lieu de décorer manuellement chaque élément vous pouvez définir [des règles](#) pour leur appliquer un style

Groupes (TECShapes)

Un groupe gère un ensemble d'éléments, en fait la propriété Shapes est le groupe par défaut, vous accédez à un groupe par la propriété Group['name']

*Vous pouvez aussi y accéder par un index au travers de la propriété **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**

vide le groupe

procedure **BeginUpdate**

Ne pas répercuter immédiatement les ajouts d'éléments dans le groupe, assure une meilleur performance

procedure **EndUpdate**

Répercuter les ajouts au groupe

```

  // 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**

Affiche la carte de manière a ce que tous les éléments du groupe soient visible dans le meilleur zoom

*Les propriétés **Pois**, **markers**, **lines** et **Polygones** disposent aussi d'une procédure **fitBounds***

procedure SaveToFile(const Filename: string)

Enregistre le groupe dans un fichier, le format est déterminé par l'extension .gpx, .kml ou format texte propre à TECNativeMap

function LoadFromFile(const Filename: string): boolean

Charge le groupe avec le contenu d'un fichier, le format est déterminé par l'extension .osm, .olt .gpx, .kml ou format texte propre à TECNativeMap

L'événement OnLoad est déclenché lorsque le contenu est totalement disponible

function LoadFromOSMStream(const Stream: TStream): boolean;

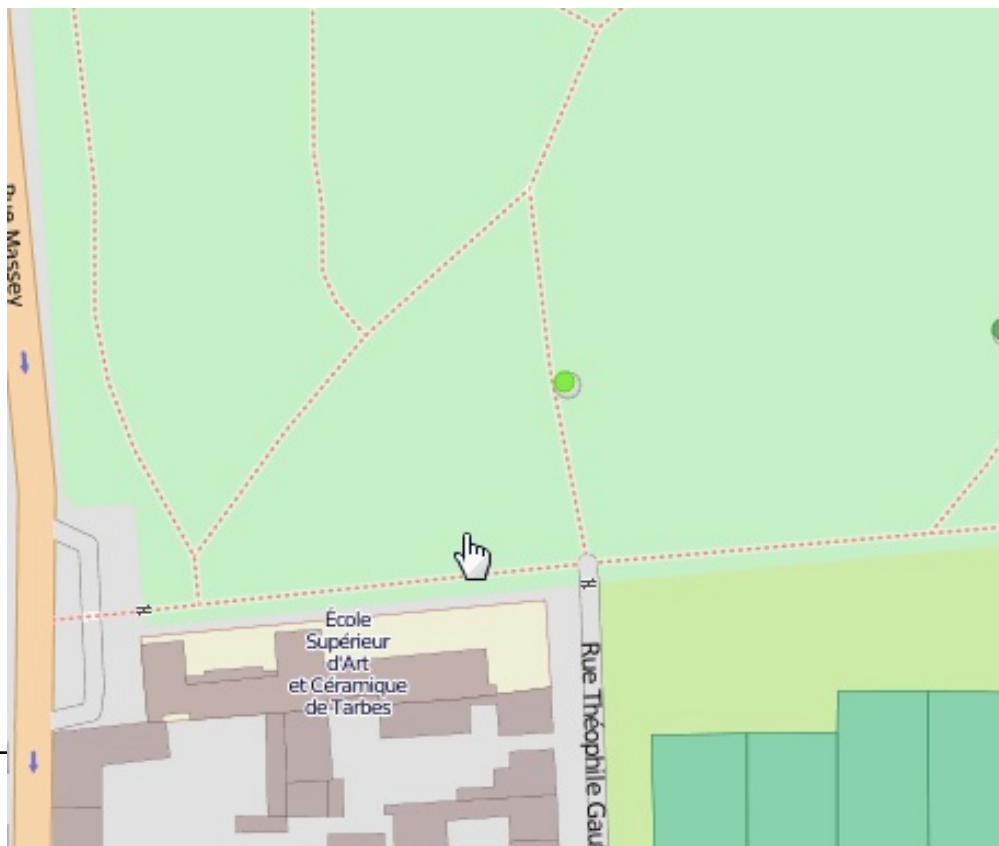
Charge le groupe avec un stream contenant des données au format OSM XML ou olt

function LoadFromOSMString(const data: string): boolean;

Charge le groupe avec une chaine contenant des données au format OSM XML ou olt

property Clusterable : boolean

Active les support des [Clusters](#), cela permet de regrouper des markers ou des pois proches pour une meilleure visibilité



property **ClusterManager** : TECClusterManager

Gestion des clusters

property **Pois**: TECShapePOIList

Liste des éléments de type [TECShapePOI](#)

property **Markers**: TECShapeMarkerList

Liste des éléments de type [TECShapeMarker](#)

property **Lines**: TECShapeLineList

Liste des éléments de type [TECShapeLine](#)

property **Polygones**: TECShapePolygoneList

Liste des éléments de type [TECShapePolygone](#)

property **InfoWindows**: TECShapeInfoWindowList

Liste des éléments de type [TECShapeInfoWindow](#)

Les listes d'éléments disposent d'un itérateur comme les groupes

```
// 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

Niveau maximal de zoom pour lequel les éléments sont affichés, au-dessus le groupe ne s'affiche pas

property **MinZoom**: byte

Niveau minimal de zoom pour lequel les éléments sont affichés, en dessous le groupe ne s'affiche pas

property **Zindex** : longint

Les groupes s'affichent dans l'ordre croissant de leur ZIndex

function **TopZindex** : longint;

Retourne le ZIndex le plus élevé de l'ensemble des éléments du groupe

TECNativeMap.TopGroupZindex *retourne le Zindex le plus élevé de tous les groupes*

property **Clickable** : boolean

Indique si l'ensemble des éléments sont clickables

Les éléments ont eux aussi une propriété clickable

property **Serialize** : boolean

Indique si le groupe doit être sauvegardé lorsque l'on enregistre l'ensemble de la carte

Ne s'applique pas si l'on sauvegarde directement le groupe

```
map.Group['group1'].Serialize := true;
map.Group['group2'].Serialize := false;

map.SaveToFile(filename); // group2 not save
```

property **Visible**: boolean

Affiche/Cache le groupe

property **Show**: boolean

Affiche/Cache le groupe mais contrairement à visible les éléments restent accessible à la souris

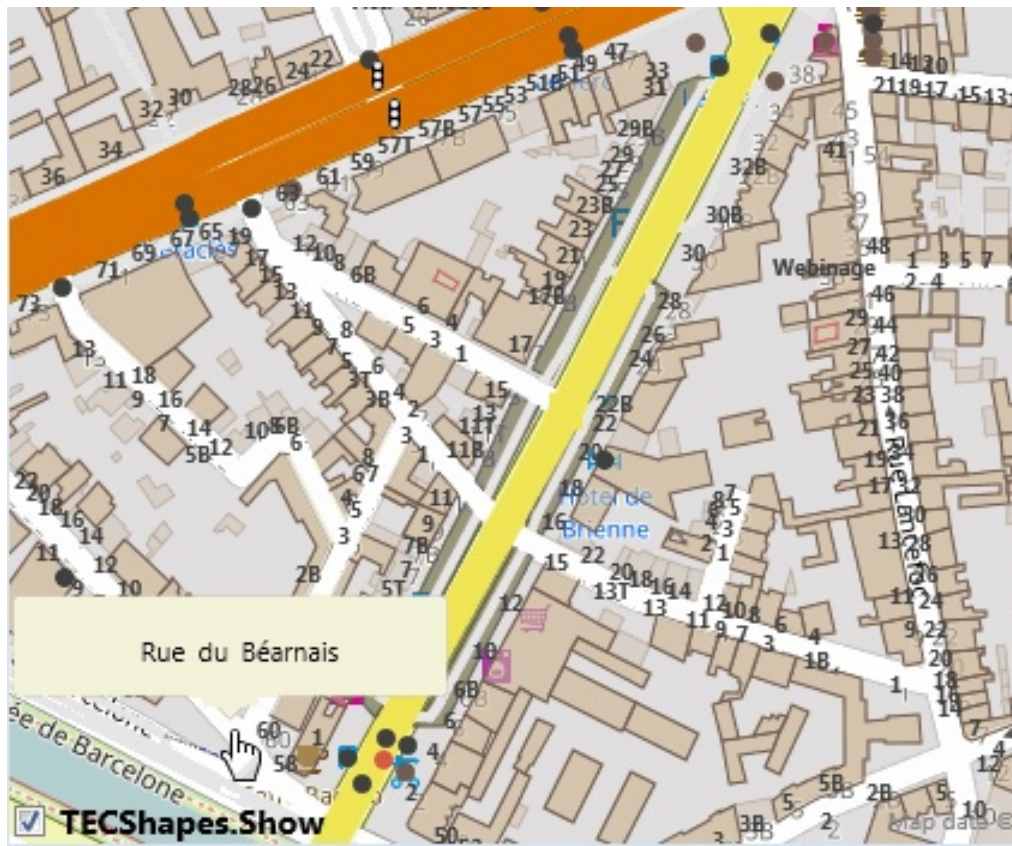


Fig. 94 TECShapes.Show

property **ShowHint**: boolean

Autorise l'affichage de la propriété **Hint** des éléments sous la forme d'une bulle lors du survol de la souris

property **Name**: string

nom du groupe en lecture seule

property **ToTxt**: string;

Import/Export au format natif

property **ToGpx**: string;

Import/Export au format Gpx

property **ToKml**: string;

Import/Export au format Kml

Ajouter des Shapes

Vous pouvez utiliser les raccourcis suivant pour ajouter vos éléments

```
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;

***Vous ne devez pas détruire manuellement les éléments
ainsi créé si besoin utilisez **remove**, ils sont maintenu
dans les listes de leur groupe respectif***

TECClusterManager

Les propriétés suivantes vous permettent de modifier l'affichage des 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

Les éléments qui sont à moins de MaxPixelDistance d'un cluster y sont regroupés, 60 pixels par défaut

property **DrawWhenMoving** : boolean

Permet d'afficher les clusters pendant que l'on déplace la carte, par défaut true

property **MaxZoom** : byte

Si le zoom est supérieur à **MaxZoom** on ne regroupe pas les éléments, défaut 18

property **OnAddShapeToCluster** : TOnAddShapeToCluster (sender : TECCluster; const Shape:TECShape;var cancel:boolean)

Déclenché lors de l'ajout d'un élément dans un cluster, vous pouvez refuser en basculant cancel à true, défaut false

property **OnColorSizeCluster** : TOnColorSizeCluster (const Cluster : TECCluster;

```
var Color:TColor;var BorderColor:TColor;var TextColor:TColor,
var WidthHeight,FontSize:integer)
```

Déclenché avant l'affichage pour vous permettre d'ajuster les propriétés

```
property OnDrawCluster : TOnDrawCluster (const Canvas :
TECCanvas; var rect : TRect; Cluster : TECCluster)
```

Si vous vous branchez sur cet événement vous prenez en charge intégralement le dessin du cluster

```
property OnMouseOverCluster : TOnNotifyEventCluster (const Cluster
: TECCluster)
```

Déclenché par l'entrée de la souris sur le cluster

```
property OnMouseOutCluster : TOnNotifyEventCluster (const Cluster
: TECCluster)
```

Déclenché par la sortie de la souris du cluster

exemple, changer la couleur en fonction du nombre d'éléments contenu par le 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;

```

Élément TECShape

Tous les éléments affichable descendent de **TECShape**, ils partagent les propriétés suivantes

property **Clusterable** : boolean

Permet d'exclure l'élément des clusters, par défaut true

```

map.shapes.markers[10].Clusterable := false;

```

property **Address**: string

Retourne l'adresse de l'élément, la recherche est bloquante

procedure **Location**

Geolocalise l'élément, équivalent **Address** mais n'est pas bloquante

Déclenche **OnShapeLocation**

(item:TECShape,GeoResult:TECGeoResult;const Valid:boolean);

Si le TECShape n'a pas d'événement OnShapeLocation c'est le OnShapeLocation de son [groupe](#) qui est déclenché.

Valid est a true si la position actuelle de l'élément est la même que celle lors du déclenchement de la recherche.

```
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)

Retourne la distance en kilomètres entre les deux éléments

procedure InfoWindow(**const** content:string);

procedure InfoWindow(window:TECShapeInfoWindow);

Associe une [infoWindow](#) à l'élément, elle s'affiche lors d'un click sur l'élément.

```
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**

Permet de supprimer directement l'élément

procedure **SetPosition**(const dLat, dLng: double)

Déplace l'élément en latitude et longitude

procedure **SetDirection**(const dLat, dLng: double)

Déplace l'élément en latitude et longitude, change son angle de rotation pour qu'il pointe dans le sens de la direction

procedure **CenterOnMap**

Déplace la carte de manière à ce que l'élément soit au centre

function **IndexOf**: integer

Index de l'élément dans sa liste (Pois, Markers, Lines ou Polygones)

function **ShowOnMap**: boolean

Indique si l'élément est dans la partie visible de la carte

property **Animation** : TECShapeAnimation

Permet d'animer la forme voir [Animation](#)

property **Latitude**: double

Latitude de l'élément

property **Longitude**: double

Longitude de l'élément

property **Altitude**: double

Altitude de l'élément, vous devez avoir connecté un composant TECGeolocalise à votre TECNativeMap

property **Color**: TColor

Couleur de l'élément

property **HoverColor**: TColor

Couleur au survol

property **Tag**: longint

Vous pouvez l'utiliser librement

property **Visible** : boolean

Affiche ou cache l'élément

property **TrackLine** : **TECShapeLine**

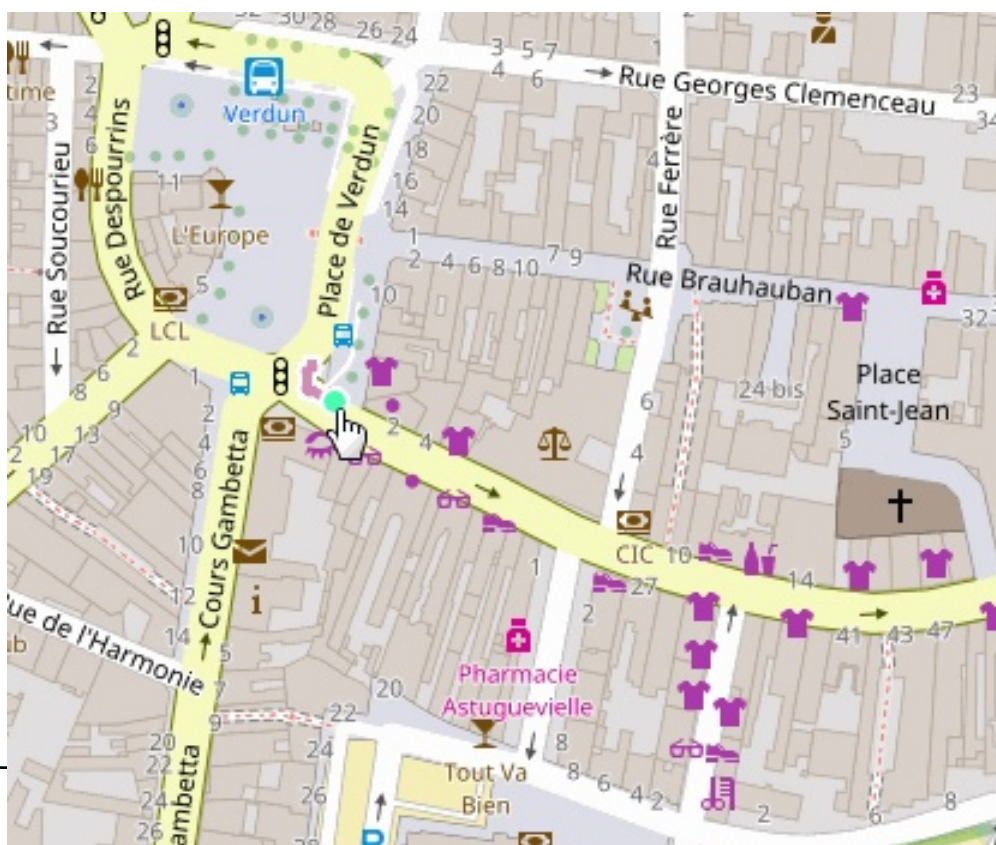
Si TrackLine est définie la trace du déplacement de l'élément sera automatique gérée

Pour l'activer il suffit simplement d'y accéder, vous pouvez aussi directement assigner une ligne

```
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

Permet de savoir si une TrackLine est activée.

Si vous testez directement TrackLine elle est automatiquement créée si elle n'existe pas !

property **Serialize** : boolean

Permet d'indiquer si l'élément peut être enregistré dans un fichier (oui par défaut)

property **Clickable**

Rend l'élément réactif à la souris

property **ZIndex**

Les éléments s'affichent dans l'ordre croissant de leur ZIndex

BringToFront

Force l'élément à s'afficher au premier plan sans tenir compte des zindex

Un seul élément à la fois peut-être au premier plan, BringToFront ne change pas l'ordre d'affichage qui reste dépendant du ZIndex

SendToBack

Retour à l'ordre normal d'affichage

property **Draggable**

Rend un élément déplaçable à la souris

property **Angle**

Angle de rotation en degré (0-360)

Disponible pour les `TECShapeMarker` et `TECShapePOI`

property **Hint: string**

Texte affiché dans la bulle lorsque la souris survole l'élément, la propriété ShowHint du groupe parent doit avoir été basculé à true

Le texte peut être enrichi avec certains balises html voir

property EnabledHint

Active/Désactive l'affichage de la bulle d'information de l'élément

property MaxShowHint

indique le nombre de fois que sera affichée la bulle d'information, vous pouvez la réactiver avec EnabledHint ou en rechargeant MaxShowHint

property PropertyValue[Name]

vous permet d'accéder à la valeur de la propriété **Name**

Vous pouvez utilisez PropertyValue pour stocker vos propres données, cette propriété est aussi utilisée par les tuiles vectorielles, les données OpenStreetMap y sont sauvegardées.

property Properties

Liste des Property/Value, au format "nom:valeur", chaque couple est séparé par un retour chariot.

property Group: [TECShapes](#)

Groupe auquel l'élément appartient

property Selected

Sélectionne ou non un élément, un élément sélectionné s'affiche comme s'il était survolé

La propriété `Selected` de la carte contient tous les éléments sélectionnés

```
// you can use Iterator
```

```
var shape:TECShape
```

```
for shape in map.Selected do
begin
  ..
end;
```

TECSelectedShapesList

Type de la liste des éléments sélectionnés, vous donne accès à

```
function ByArea(const
NEALat,NEALng,SWALat,SWALng :double):integer;
```

Sélectionne les éléments se trouvant dans la zone délimitée par NEALat,NEALng (coin haut-droit) et SWALat,SWALng (coin bas-gauche)

```
function ByKMDistance(const
FLat,FLng,FKMDistance :double):integer;
```

Sélectionne les éléments situés à moins de KMDistance (distance en km) du point FLat,FLng

```
procedure UnSelectedAll
```

Désélectionne tous les éléments sans les supprimer de la carte

```
procedure Clear
```

Efface de la carte tous les éléments sélectionnés

```
function count : integer;
```

```
procedure SaveToFile(const filename:string);
```

```
property Item[index: integer]: TECShape
```

Retourne l'élément dont on passe l'index

```
property ToTxt:string
```

Retourne les éléments sélectionnés dans le format texte interne de TECNativeMap (en lecture seulement)

```
property GroupFilter : TStringList
```

Permet de filtrer en fonction des [groupes d'éléments](#), laissez vide pour pouvoir sélectionner les éléments sans tenir compte de son groupe

property **OnChange**:TNotifyEvent ;

Déclenché par l'ajout/suppression d'un élément à la liste des sélectionnés

Recherche

Vous avez plusieurs fonctions pour effectuer des recherches dans vos éléments

```
function FindShapeByArea(const SWALat, SWALng, NEALat,
NEALng: double;
                        const ShapeList: TList<TECShape>;
                        const FilterShapes :
TNativeShapes = [];
                        Filter:TOnShapeFilter=nil): integer;
```

Rempli **ShapeList** avec les éléments se trouvant dans une zone rectangulaire dont on indique les coins Sud-Ouest et Nord-Est

FilterShapes permet de filtrer les éléments recherchés, par défaut tous, exemple **[nsMarker,nsPoi]** pour ne rechercher que des TECShapeMarker et TECShapePOI

Filter permet d'ajouter une procédure pour filtrer la recherche

```
// 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;
    Rempli ShapeList avec les éléments qui se situent à
    moins de FKMDistance km du point FLat,FLng

function FindShapeByFilter(const
ShapeList: TList<TECShape>;
                                const FilterShapes
: TNativeShapes;
                                Filter:TOnShapeFilter): integer;

```

Toutes ces fonctions retournent le nombre d'éléments trouvé.

Pour les versions de Delphi qui ne supportent pas les génériques ShapeList est une simple TList

Événements

Les formes déclenchent les événements suivant :

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 **OnShapeDbClick** : TOnShapeMouseEvent

property **OnShapePathChange** : TNotifyEvent

property **OnShapeLocation** : TOnShapeLocation

Les événements peuvent être attribués individuellement à chaque élément, dans ce cas ils ne sont pas répercutés au niveau de 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;
```

Aimer des markers sur une ligne ou un polygone

Vous pouvez définir des **TECShapeMarker** et des **TECShapePOI** qui seront automatiquement attirés vers une **TECShapeLine** ou un **TECShapePolygone** si vous les lâchez à une certaine distance de ceux-ci.



Fig. 96 SnapDrag

Vous gérez cela au travers de la propriété **SnapDrag** de type **TECSnapDrag** du composant cartographique **TECNativeMap**.

procédure **ClearMarker**;

Supprime tous les markers aimantés

fonction **AddMarker(const Marker:**

TECShape): boolean;

Ajoute un TECShapeMarker ou TECShapePOI dans la liste des markers aimantés.

retourne true si l'élément est bien ajouté.

procedure RemoveMarker(const Marker: TECShape);

supprime le marker de la liste des markers aimantés

procedure ClearGuide;

Supprime tous les TECShapeLine et TECShapePolygone de la liste des guides

function AddGuide(const Guide: TECShapeLine): boolean;

Ajoute un TECShapeLine ou un TECShapePolygone dans la liste des guides.

retourne true si l'élément est bien ajouté.

procedure RemoveGuide(const Guide: TECShapeLine);

Supprime l'élément de la liste des guides

property MeterDistance: boolean ;

Sélectionne l'unité pour la distance déclenchant l'aimantation.

true pour des metres , false (default) pour des pixels.

property SnapDistance: integer ;

Distance d'attrance des markers vers un guide

property SnapShape: TECShape ;

Retourne le marker qui vient d'être attiré.

property SnapGuide: TECShape;

Retourne le guide qui vient d'attirer un marker

property **OnSnap: TNotifyEvent;**

L'événement est déclenché lorsqu'un marker est attiré par un 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
```

*Si vous souhaitez que des markers soient attirés par des lignes particulières, vous pouvez créer votre propre variable de type **TECSnapDrag***

```
var MySnapDrag : TECSnapDrag;
...
MySnapDrag := TECSnapDrag.create;

MySnapDrag.addMarker(mrk);

MySnapDrag.AddGuide(poly);
MySnapDrag.AddGuide(red_line);
...
MySnapDrag.free
```

TECShapeMarker

Les TECShapeMarker sont des marqueurs représenté par une image, vous pouvez la leur attribuer de 3 façons

En indiquant un fichier .png au travers de la propriété **Filename**

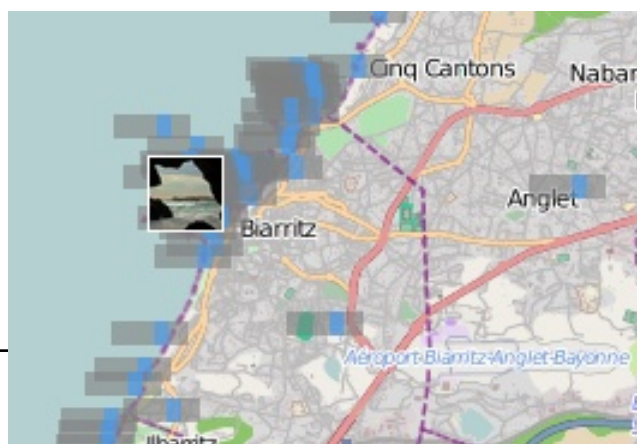
```
map.Shapes.Markers[0].filename :=  
;http://google-maps-icons.googlecode.com/files/restaurant.png'
```

Filename accepte aussi les fichiers locaux

Vous pouvez aussi indiquer directement une image encodée en Base64 (Data URI)

```
const Data_Uri =  
'data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAAAUA  
AAAFCAYAAACNbyblAAAAHE1EQVQI12P4//8/w38GIAXDIBKE0DHxgljNBAAO  
9TXL0Y4OHwAAAABJRU5ErkJggg==';  
  
map.Shapes.Markers[0].filename := Data_Uri;
```

Par défaut une animation indique que l'image est entrain d'être téléchargée, vous pouvez supprimer cet effet en basculant la propriété **WaitAnimation** à false



Dans la **version VCL** vous pouvez attribuer une **TImageList** contenant des png à la propriété **Icons** du composant **TECNativeMap**, puis vous indiquez l'index de l'image dans la propriété **Icon** du **TECShapeMarker**.

```
map.Icons := myImageList;
// use icon number 1
map.Shapes.Markers[0].Icon := 1;
```

vous pouvez aussi directement affecter un **TPngImage** à la propriété **Graphic** du **TECShapeMaker**

```
map.Shapes.Markers[0].Graphic := MyPngImage;
```

Vous pouvez spécifier la zone clickable avec la fonction **SetHitBox** (x,y,w,h)

X,Y indique le coin haut gauche de la zone clickable

W la largeur

H la hauteur

Le coin haut gauche du marker est 0,0

Utilisez les propriétés **XAnchor** et **YAnchor** pour déterminer le point précis de l'image correspondant à la latitude et longitude

Exemple pour centrer l'image sur les coordonnées géographique

```
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;
```

Image par défaut

Si vous laissez **Filename** et **Graphic** vide, si vous n'attribuez pas de **TImageList** ou définissez **Icon** en dehors des limites alors le marqueur va se dessiner en utilisant comme couleur principale sa propriété **Color**

StyleIcon

Il existe 6 styles pour les markers sans images



Fig. 98 StyleIcon si3d - siFlat - siFlatNoBorder - siDirection

Si vous avez plusieurs dizaines de milliers de points à afficher, utilisez le style siFlatNoBorder, c'est le plus rapide à afficher

```
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;

```

vous pouvez redéfinir sa couleur au survol de la souris
*au travers de la propriété **HoverColor***

SVG

Sous Firemonkey vous pouvez aussi utiliser des images au format SVG, uniquement les plus simples mais c'est suffisant pour afficher les icones du projet [MAKI](#) par exemple.



Fig. 99 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}'
```

Vous pouvez aussi utiliser `siOwnerDraw` pour prendre en charge totalement le dessin de votre 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;
```

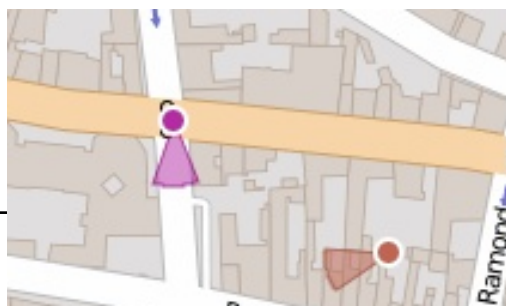
Champ de Vision

Si la propriété **Fov** (Field of view) est supérieur à 0, un cône s'affiche dans la direction définie par l'angle du marker.

Un angle de 0 correspond au nord.

Vous pouvez modifier la longueur du cône avec la propriété **FovRadius**.

```
mrk.fov := 40;
mrk.FovRadius := 30;
mrk.Angle := 180;
```



Scale

La propriété **Scale** permet de modifier la taille du marker.

```
mrk.scale := 1.5; // increases by 50%  
mrk.scale := 0.5; // decrease of 50%
```

Vous pouvez utiliser `map.styles.addRule('.marker:hover {scale:1.5;}'` pour augmenter la taille des markers lorsque la souris passe dessus

```
map.styles.addRule('.marker:hover {scale:1.5;}');
```



Fig. 101 automatic scale

Adapter la taille en fonction du Zoom

Utilisez la propriété **ScaleMarkerToZoom** pour que la taille du marker change en fonction du Zoom.



Fig. 102 ScaleMarkerToZoom

```
map.ScaleMarkerToZoom := true;
```

OnBeforeDraw

Cet événement est déclenché avant le dessin de votre marker, vous pouvez vous en servir pour ajouter un fond

```
// 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;
```

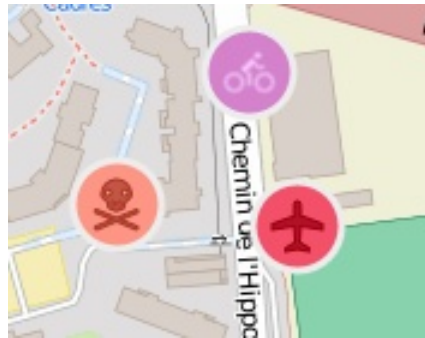


Fig. 103 OnBeforeDraw

GroundOverlay

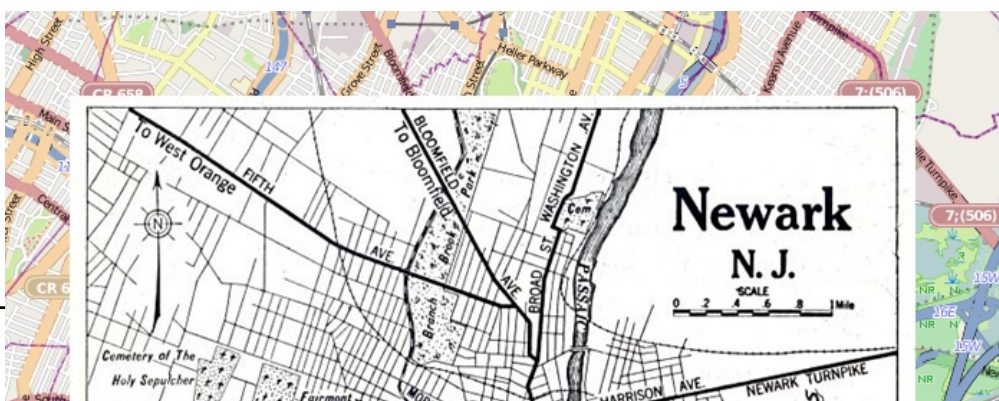
Vous pouvez contraindre une image à recouvrir une certaine zone

SetBounds(NorthEastLatitude,NorthEastLongitude,SouthWestLatitude,SouthWestLongitude)

Vous activez le recouvrement en basculant la propriété **fitBounds** à 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;
```



Les **Pois** (point of interest) sont équivalents aux [markers](#), la différence est que vous avez 9 formes prédéfinies (Texte, Ellipse, rectangle, triangle, flèche, point de flèche, losange, croix, croix en diagonale, étoile et hexagone) mais vous pouvez aussi prendre en charge vous même leur dessin.

Par défaut les tailles des éléments sont en pixels mais vous pouvez aussi utiliser des mètres en utilisant la propriété **POIUnit**.

```
// Delphi map component EMap

// add POI at center of map
id := map.shapes.Pois.add(map.latitude,map.longitude);

// random form

case random(11) 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. 105 PoiShape

Vous pouvez modifier la taille des formes en basculant leur propriété **Editable** à 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;

```

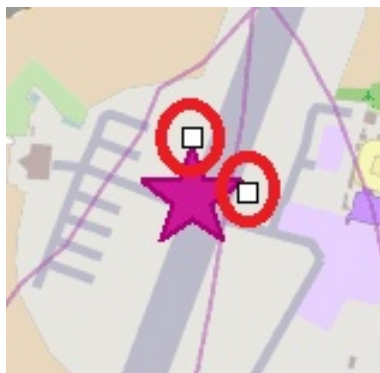


Fig. 106 TECShapePOI en mode édition - poignée de redimensionnement affichée

TECMapPois

Les POIs sont gérés par une liste de type **TECShapePois** accessible au travers de la propriété **Pois** des groupes **TECShapes**

la propriété **OnOwnerDrawPOI** : TOnOwnerDrawPOI permet d'indiquer une procédure pour prendre en charge le dessin des Pois de 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;

```

Chaque élément TECShapePOI dispose de sa propre propriété OnOwnerDrawPOI

Vous avez aussi une propriété OnAfterDraw qui permet de dessiner par dessus une figure, exemple pour y inscrire son numéro.

```

// 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;
```

Comme pour les [TECShapeMarker](#) la propriété [Scale](#) permet d'ajuster la taille et [ScaleMarkerToZoom](#) est aussi fonctionnelle.

TECShapeLine vous permet de tracer une ligne constituée d'un ensemble de points sur votre carte.

Les lignes sont gérés par une liste de type **TECShapeLines** accessible au travers de la propriété **Lines** des groupes **TECShapes**

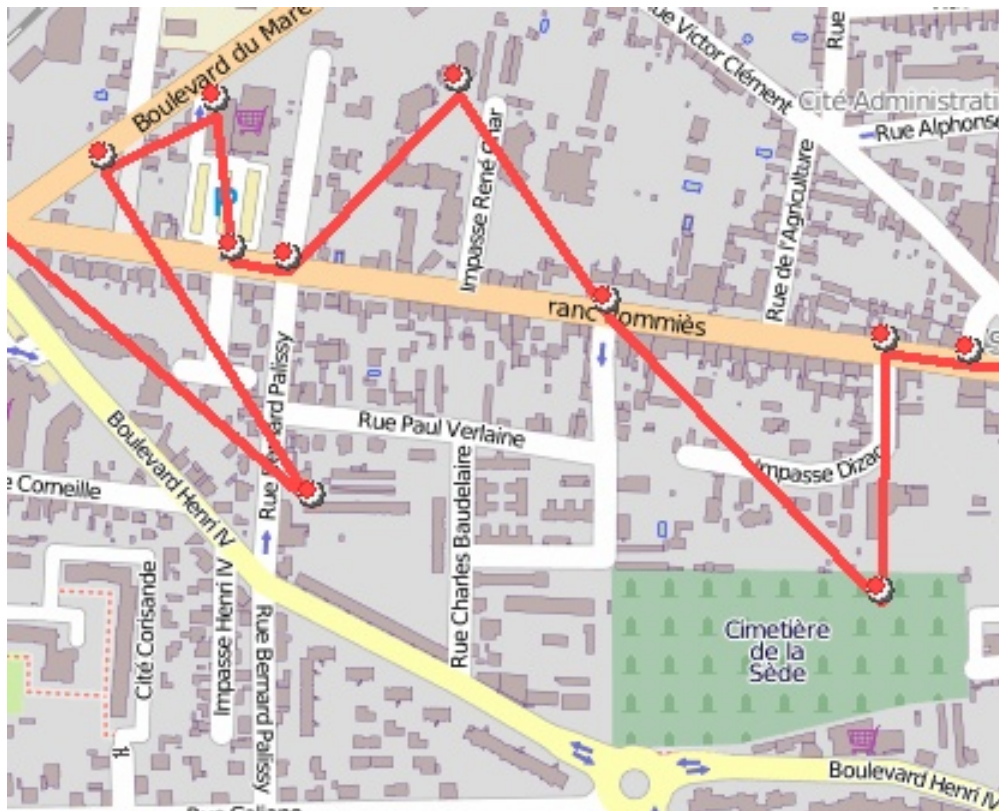


Fig. 107 Une ligne en mode édition

TECShapeline

Cette classe descend de **TECShape**

Elle possède en plus les propriétés suivantes

function **Add**(const Lat, Lng: double; const Alt: double =
_ErrorAltitude): integer;

Ajoute un point avec éventuellement son altitude

function **Add**(const dLatLngs: array of double): integer;

Ajoute un tableau de points, constitué d'une suite de latitudes et de longitudes

```
line.add([lat1,lng1,lat2,lng2,...,latx,lngx]);
```

Contrairement à l'ajout d'un simple point, vous n'avez pas besoin d'encadrer l'ajout de votre tableau par BeginUpdate / EndUpdate, cela est fait automatiquement.

procedure **Insert**(const index: integer; const Lat, Lng : double; const Alt: double = _ErrorAltitude);

Insère un point avec éventuellement son altitude

procedure **Reverse**

Inverse le sens de la ligne

property **ShowDirection** : boolean

Ajoute des flèches à chaque segment pour montrer la direction



Fig. 108 ShowDirection

```
procedure Slice(const StartIndex,EndIndex : integer;Line :
TECShapeLine); overload;
```

Copier dans Line la partie délimité par les segments StartIndex et EndIndex

```
function Slice(const
```

```
StartIndex,EndIndex:integer;GroupName:string=""):TECShapeLine; overload;
```

Retourne une ligne composée de la portion délimité par les segments StartIndex et EndIndex

```
procedure Slice(const StartKm,EndKm : double;Line :
TECShapeLine); overload;
```

Copier dans Line la partie débutant au kilomètre StartKM et finissant au kilomètre EndKM

```
function Slice(const StartKm,EndKm :
double;GroupName:string=""):TECShapeLine; overload;
```

Retourne une ligne composée de la portion débutant au kilomètre StartKM et finissant au kilomètre EndKM



```
// 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**;

Efface les points

property **Encoded**: string ;

Encode / Decode le polyline en utilisant l'algorithme de google (
developers.google.com/maps/documentation/utilities/polylinealgorithm)

```
line.add([lat1,lng1,lat2,lng2,...,latx,lngx]);
```

property **EncodePrecision**: byte;

Précision d'encodage, par défaut 5 décimales (180.00000 to -180.00000)

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

Supprime un segment de la ligne

function **Count**: integer;

Indique le nombre de segment

property **Weight**: byte

Épaisseur de la ligne

property **BorderSize** : integer

Épaisseur de la bordure

property **BorderColor** : TColor

Couleur de la bordure

property **HoverBorderColor** : TColor

Couleur de la bordure au survol de la souris



Fig. 110 Bordure noire de 2 pixels

property **PenStyle** : TPenStyle

type de trait **psSolid**, **psDash**, **psDot** et **psDashDot** sont disponibles

property **LineType** : TECLineType

Type de ligne : **ltStraight** ligne droite entre chaque point, **ltBezier** pour tracer des courbes de bézier



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

Calcule les altitudes des points, vous pouvez lui passer une procédure de type TOnGetAltitude pour pouvoir afficher une barre de progression (voir DemoNativeRoute)

```
// 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;
```

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

Calcule la latitude et la longitude d'un point sur la ligne/polygone en fonction de sa distance en mètres, retourne **True** si on a trouvé un point

SensStartEnd sens du parcours, true pour départ -> arrivée

IMeter la distance en mètre

dLatitude,dLongitude des variables de type **double** qui recevront la latitude et la longitude

idPoint une variable qui contiendra l'indice dans le tableau **Path** où se situe le point, le calcul retourne une approximation car votre polyline/polygone ne comporte pas l'ensemble des points réels

Heading une variable qui contiendra l'angle du point par rapport au nord (de 0 à 360°)

bEnd indique si l'on a atteint ou dépassé la fin du polyline/polygone(ou le début suivant le sens)

property **HoverPoint**: integer read FHOverSeg;

Index du segment pointé par la souris

property **Path**[index: integer]: TECPointLine

Tableau contenant l'ensemble des points de la ligne

function **Distance** : double;

Indique la distance totale de la ligne en km

property **Duration** : integer;

Si la ligne représente un itinéraire (route) indique la durée du trajet.

property **NorthEastLatitude**: double

Latitude du coin supérieur droit de la boîte englobant la ligne

property **NorthEastLongitude**: double

Longitude du coin supérieur droit de la boîte englobant la ligne

property **SouthWestLatitude**: double

Latitude du coin inférieur gauche de la boîte englobant la ligne

property **SouthWestLongitude**: double

Longitude Latitude du coin inférieur gauche de la boîte englobant la ligne

```
// Delphi map component EMap  
  
line := map.Shapes.Lines[0];  
  
// show the entire line  
map.fitBounds(line.NorthEastLatitude,line.NorthEastLongitude,line.SouthWestL
```

property **ShowText** : boolean

Si la ligne représente une route permet d'afficher les diverses étapes

property **Shapes** : [TECShapes](#)

Si la ligne représente une route donne accès aux formes qui signalent les étapes

Les points de départ et d'arrivée sont représentés par des [TECShapePOI](#), les autres points intermédiaires sont des [TECShapeMarker](#)

L'exemple ci-dessous permet de modifier l'affichage des points de départ et d'arrivée.

```
// Delphi map component EMap
// change default poi start and finish of route
var line : TECShapeLine;
...
line := map.shapes.lines[0];

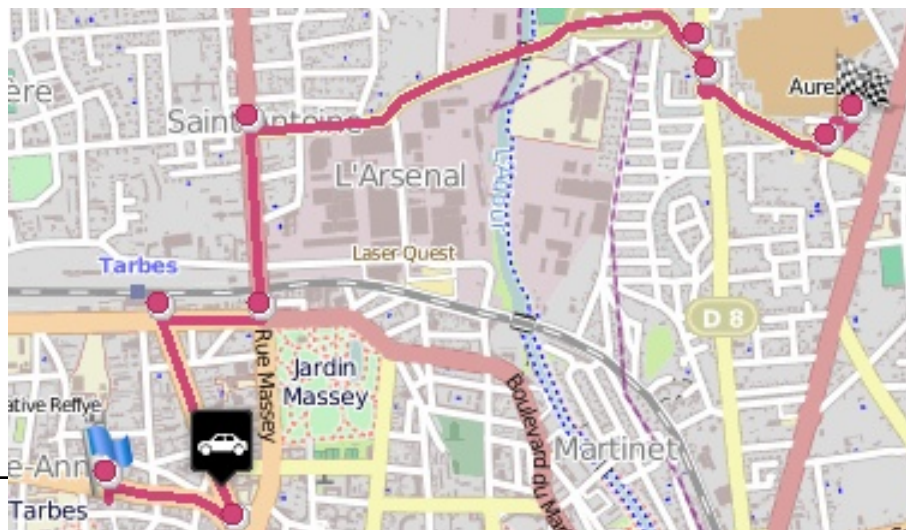
if line.shapes.Pois.count > 1 then
begin
  i := line.shapes.markers.add(line.shapes.pois[0]
].latitude, line.shapes.pois[0].longitude);
  line.shapes.markers[i].filename :=
'http://www.helpandweb.com/flag_blue.png';
  line.shapes.markers[i].hint := line.shapes.pois[0]
].hint;

  i

1. line.shapes.markers.add(i).latitude, line.shapes.pois[line.shapes.Pois.count-
].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

Rend éditable la ligne, un double click sur un point le supprime, un double clic sur un segment ajoute un point, les points sont déplaçable à la souris

Cela ne permet que des tracés rectilignes, pour modifier une route cela demande juste 2-3 lignes de codes supplémentaires.

Par défaut les points de sélection sont représentés par un carré noir pour le point de départ, un carré blanc pour le point final et un rond blanc pour les points intermédiaires.

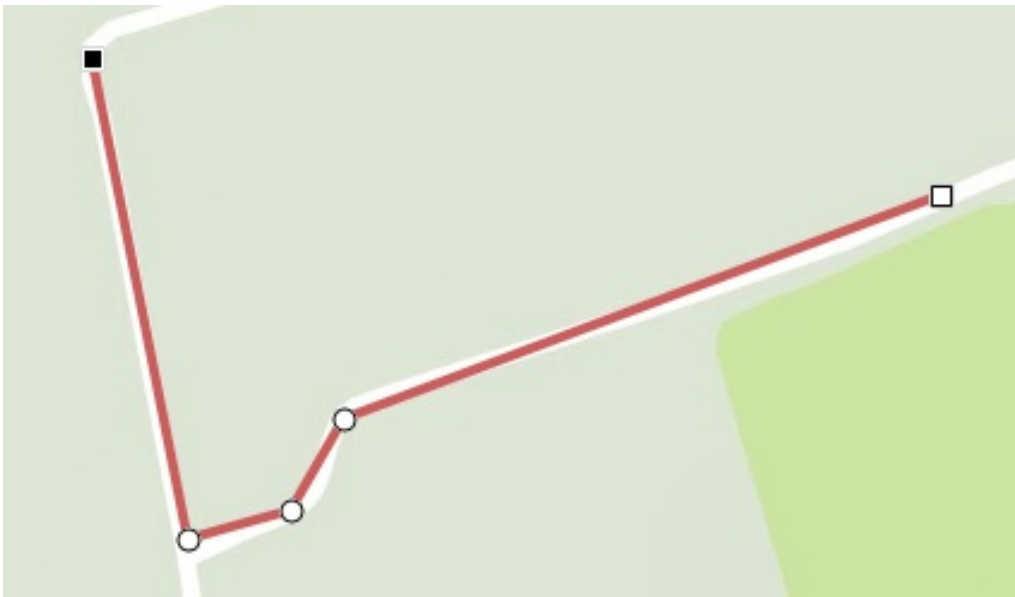


Fig. 113 Edit line

Utilisez les propriétés **FilenameStartEditLine**, **FilenameEndEditLine**, **FilenamePointEditLine** de votre carte pour changer les bitmaps

```
// Reverse the point of departure and arrival
```

```

s := map.FilenameStartEditLine;
map.FilenameStartEditLine := map.FilenameEndEditLine;
map.FilenameEndEditLine   := s;

```

Vous pouvez prendre en charge la création des points de sélection en utilisant la propriété **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;

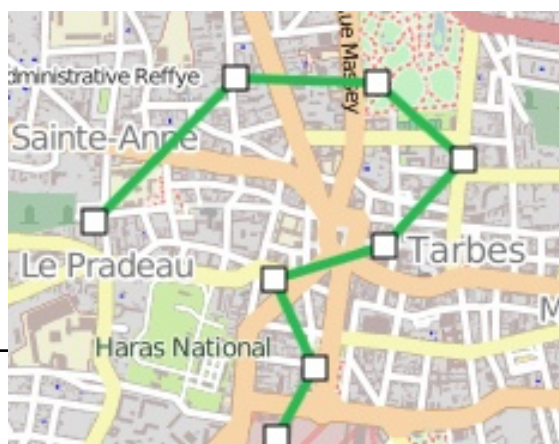
    poi.Width := 10;
    poi.Height:= 10;

    poi.POIShape := poiRect;

    poi.Color      := claWhite;
    poi HoverColor  := claWhite;
    poi.BorderColor := claBlack;
    poi HoverBorderColor:= claBlack;

end;

```



property **OnShapePathChange** : TNotifyEvent

Permet d'être informé lorsque le tracé de la ligne change.

Sélectionner une portion de la ligne

Pour pouvoir sélectionner une portion de votre ligne à la souris ou par code, utilisez la classe TecNativeLineSelect (unité uecEditNativeLine)

```
var SelectLine : TecNativeLineSelect;
...
SelectLine := TecNativeLineSelect.create;
// triggered by selection
SelectLine.OnSelect := doSelect;
// triggered by deselection
SelectLine.OnDeselect := doDeselect;

SelectLine.Line := your_Line;
```

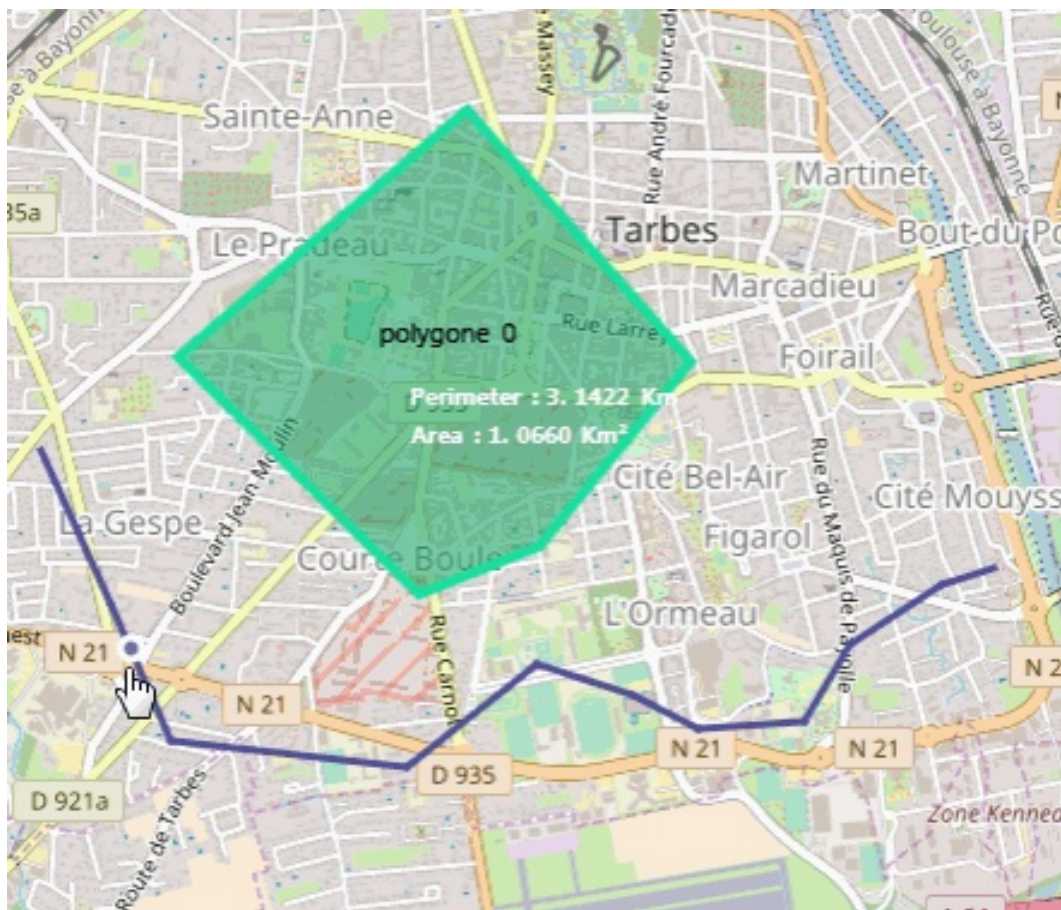


Fig. 115 Sélection d'une partie d'une ligne ou d'un polygone

Survolez la ligne et cliquez sur le point de sélection pour le valider, recliquez dessus pour le supprimer, l'événement OnSelect est déclenché dès la pose du second point

Vous pourrez alors créer une ligne ou un polygone avec SelectionTo, ou accéder directement aux points sélectionnés avec 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;
```

Utilisez SelectionFrom(ALat,ALng,BLat,BLng) pour sélection le segment par code.

```
// 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;
```

Ligne géodésique

Une géodésique est le chemin le plus court entre deux point, utilisez la fonction **AddGeodesicLine** pour en créer une.

```
function TNativeMapControl.AddGeodesicLine(const SLat,SLng,ELat,ELng:double;
const GroupName: string = "";
const maxSegmentLength:integer=5000): TECShapeLine;
```

Il vous faut au minimum indiquer les coordonnées de deux points.

maxSegmentLength défini la taille maximale, en mètres, des segments intermédiaires.

```
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)
```

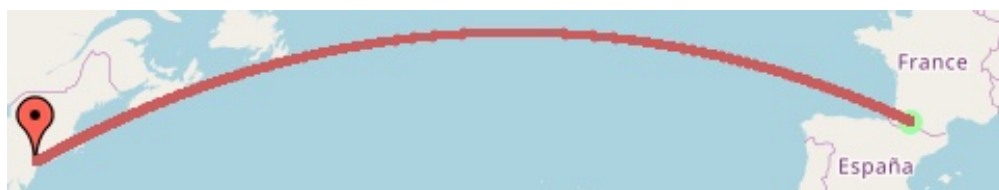


Fig. 116 geodesic line

Enregistrer son déplacement

Le but de cet exemple est d'enregistrer son déplacement dans une

TECShapeLine, un triangle pointant dans la direction du déplacement indiquera la position actuelle.

```

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;

```

Météo

En utilisant les services d'[OpenWeatherMap.org](https://openweathermap.org) vous pouvez afficher la météo le long de votre TECShapeLine.

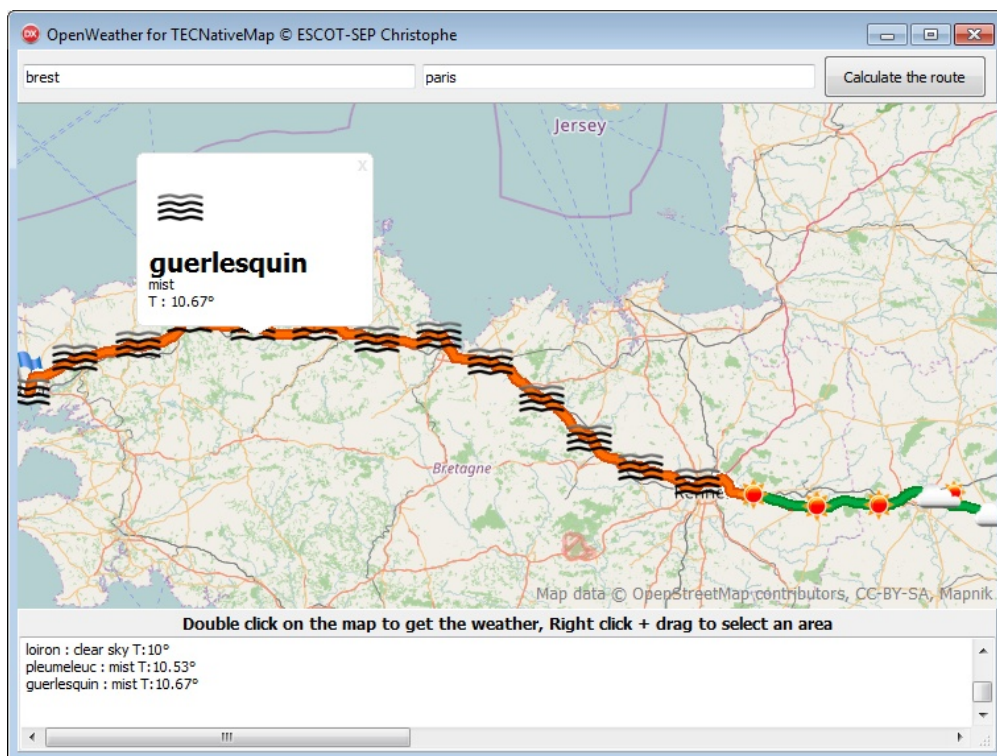


Fig. 117 Weather on the road

Vous devez tout d'abord obtenir une [clef pour utiliser l'api d'OpenWeatherMap](#)


```
map.OpenWeather.Key := your_key;

// Get the weather along a TECShapeLine
Line.ShowWeather := true;
```

Consulter la demo RouteWeather pour plus d'informations sur la façon d'utiliser l'api d'OpenWeatherMap.

TECShapePolygone descend de **TECShapeLine** et permet d'afficher des polygones sur votre carte.

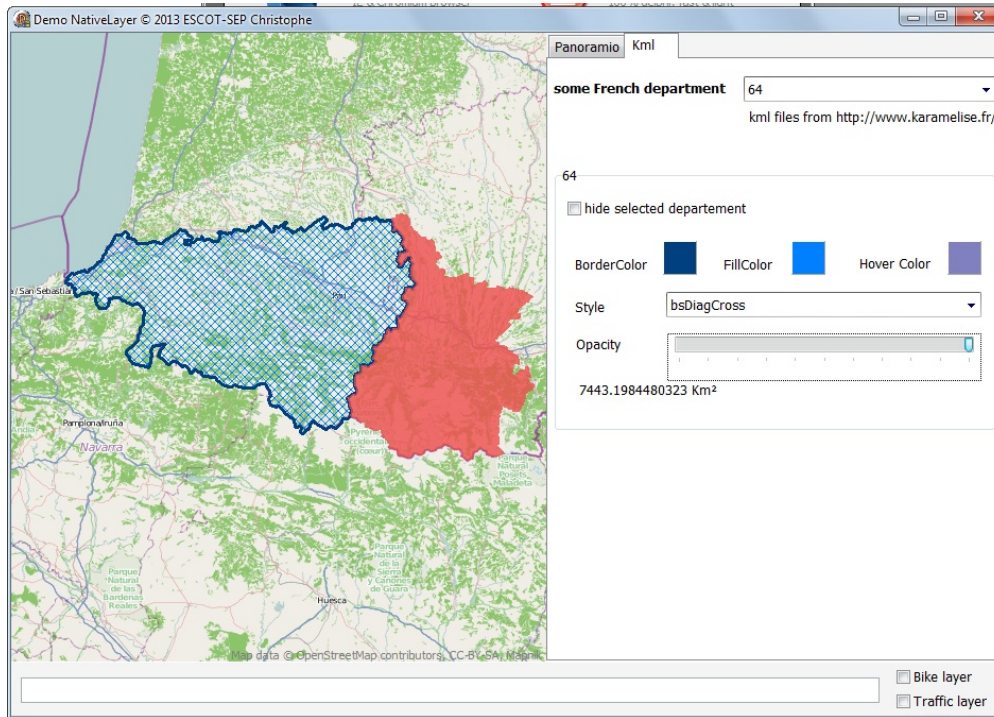


Fig. 118 Polygones

En plus des propriétés de **TECShapeLine** vous avez accès à

Property **Area** : double

Surface en Km²

property **FillColor** : TColor

Couleur de remplissage

Property **Color** : TColor

Couleur de la bordure

property **Style** : TBrushStyle

Style de remplissage (bsSolid, bsClear, bsHorizontal, bsVertical, bsFDiagonal, bsBDiagonal,

bsCross, bsDiagCross)

*Style ne fonctionne pas sous Firemonkey, utilisez à la place **BrushFilename** pour indiquer une image qui remplira votre polygone, BrushFilename peut être soit un chemin local soit une url ou une image encodée en base64 comme pour Marker.Filename*

property **Opacity** : byte

Opacité du polygone (0 = totalement transparent, 100 = opaque)

property **Level** : integer

Hauteur en mètres du polygone pour un affichage en pseudo 3D



Fig. 119 Level = 16 - Level = 0

OnAfterDraw

Vous avez aussi un événement OnAfterDraw qui permet de dessiner par dessus votre polygone, exemple pour y inscrire sa 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 vous permet d'afficher un panneau contenant du texte et des images.

Les fenêtres sont gérés par une liste de type **TECShapeList** accessible au travers de la propriété **InfoWindows** des groupes [TECShapes](#)



Fig. 120 InfoWindow

TECShapeInfoWindow

Les propriétés suivantes sont disponibles

property **Color** : TColor

Couleur de la fenêtre

property **Style** : TECInfoWindowStyle

Style de la fenêtre : **iwsTransparent**, **iwsRectangle** ou **iwsRoundRect** (par défaut)

property **Visible** : boolean

Affiche / Cache la fenêtre

property **CloseButton** : boolean

Affiche / Cache le bouton de fermeture

property **ContentCenter** : boolean

Centrer le texte horizontalement

property **Width** : integer;

Largeur en pixel

property **Height** : integer;

Hauteur maximale, si le contenu prend moins de place alors la hauteur est automatiquement ajustée

property **Content** : string

Texte à afficher

Vous pouvez utiliser un sous ensemble de HTML pour enrichir l'affichage

Les balises ****, **<a>**, **<h>**, **<tab>**, ****, **<i>**, **<u>**, **<s>**, ****, **
** et **<PlainText>** sont supportées

```
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> <img
  src="http://maps.google.com/mapfiles/ms/icons/orange-dot.png" width=32 height
```

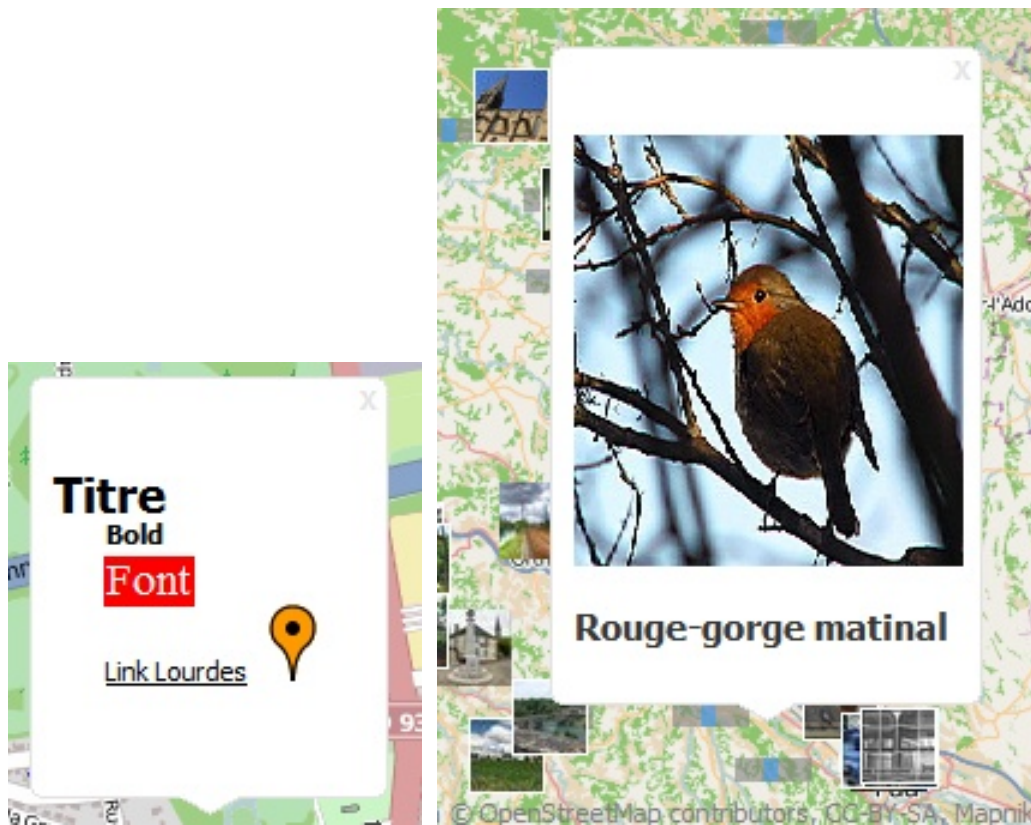


Fig. 121 html Content InfoWindow

property **OnOpen** : TOnInfoWindowOpen;

Cet événement est déclenché juste avant que le fenêtre ne soit visible, vous pouvez changez le contenu ou annuler l'ouverture

```
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;
```

Lorsqu'une fenêtre est fermée en cliquant sur sa croix l'événement **OnCloseInfoWindow** de **TECNativeMap** est déclenché et une **TECShapeInfoWindow** lui est passé en paramètre.

OnCloseInfoWindow *ne se déclenche pas si on ferme en utilisant la propriété* **visible**

Fermeture automatique

En utilisant une **animation** de type **TECAAnimationAutoHide** vous pouvez fermer automatiquement la fenêtre après quelques secondes.

```
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;
```

Lien

Vous pouvez définir un lien à l'aide de la balise `<a>`, pour intercepter le click branchez-vous sur l'événement **OnBeforeUrl** de votre carte.

```
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;
```

Vous avez la possibilité de définir un lien spécial qui vous permet de vous déplacer sur la carte, il est de la forme #zoom/latitude/longitude

```
'<a href="#16/43.094089/-0.046520"> goto here </a>'
```


TECShape dispose d'une propriété Animation de type TECShapeAnimation permettant de l'animer.

Les classes suivantes sont disponibles en standard, **TECAnimationShapeColor**, **TECAnimationAutoHide**, **TECAnimationMarkerFile**, **TECAnimationMarkerZoomFilename**, **TECAnimationFadePoi**, **TECAnimationDrawPath**, **TECAnimationMoveOnPath** et **TECAnimationMoveToDirection**

Vous pouvez créer les votre en héritant de la classe TECShapeAnimation et en redéfinissant la procédure Animation

TECAnimationShapeColor

Cette animation fait alterner les propriété **Color** et **HoverColor** du 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

Cette classe permet de modifier la propriété **Filename** des **TECShapeMarker** et donc d'avoir une animation d'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

Cette classe permet de modifier la propriété **Filename** des **TECShapeMarker** en fonction du niveau de zoom de la carte

```
...
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

La taille (width et Height) d'un [TECShapePOI](#) varie d'un minimum vers un maximum et dans le même temps le marker devient de plus en plus transparent jusqu'à disparaître.

La transparence pour les poiEllipse n'est gérée que dans la version Firemonkey



Fig. 123 TECAnimationFadePoi

property **MaxSize** : integer

Taille maximale du marker (défaut 20)

property **StartSize** : integer

Taille minimale du marker (défaut 10)

property **StartOpacity**: byte

Opacité de départ (défaut 50), elle décroît jusqu'à 0 (transparence totale)

property **Step** : integer

Nombre d'étape pour passer de StartSize à MaxSize (défaut 10)

```
anim := TECAAnimationFadePoi.Create;  
anim.MaxSize := 32;  
anim.StartSize := 8 ;  
  
anim.StartOpacity := 90;  
  
ShapePOI.Animation := anim;
```

TECAAnimationDrawPath

Affiche un [Polyline](#) pas à pas

Constructor **Create**(const ValueDuration: cardinal = 5000);

Création de l'animation, ValueDuration est le temps en milliseconde pour afficher la totalité du tracé

property **Latitude** : Double

Latitude du dernier point affiché

property **Longitude**: Double

Longitude du dernier point affiché

property **OnDraw** : TNotifyEvent

Événement déclenché après chaque affichage

*L'événement **OnEndAnimation** est déclenché lorsque le tracé a été entièrement affiché*

```
// 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. 124 TECAAnimationDrawPath

TECAAnimationMoveOnPath

Cette classe permet de déplacer un **TECShape** le long d'un **TECShapeLine**

property **Distance** : integer ;

Distance en mètres depuis le début du TECShapeLine

property **Heading** : boolean;

Ajuster l'angle de l'élément en fonction du déplacement

property **Speed** : integer;

Vitesse de déplacement en Km/H

property **Stop** : boolean;

Autoriser ou non le déplacement

property **ComeBack** : boolean;

Indique le sens de déplacement, si **ComeBack = true** alors l'élément se déplace de la fin vers le début du TECShapeLine

property **OnDriveUp** : TNotifyEvent

Événement déclenché lorsque l'élément arrive à la fin ou au début, suivant le sens, du chemin

```

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;

```



Vous n'avez pas besoin de détruire TECShapeAnimation le TECShape le fait automatiquement quand vous lui assignez une nouvelle animation ou quand il est lui-même libéré

```

shape := map.shapes.markers[0];
  // stop animation after 10 cycles
shape.Animation := TECAnimationShapeColor.create(10);
  // change color every 250ms
shape.Animation.Timing := 250;

...

  // free TECAnimationShapeColor
shape.Animation := nil;

```

Les animations disposent des événements **OnAnimation**, déclenché après chaque cycle et **OnEndAnimation** déclenché en fin d'animation si vous avez indiqué un nombre maximal de cycles lors de la création de l'animation.

```

shape := map.shapes.markers[0];
  // stop animation after 10 cycles
shape.Animation := TECAnimationShapeColor.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;

```

TECAnimationMoveToDirection

Cette animation permet de déplacer un élément dans une direction
fonction de sa vitesse

property **Direction**: double

de 0 à 360°, 0 indiquant le haut de la carte (nord)

property **Heading** : boolean

Orienté ou non l'élément en fonction de sa direction

property **Speed**: integer

Vitesse en km/h

procedure **Start**

Lance l'animation

property **Stop**: boolean

Arrête l'animation

property **TimeOut** : cardinal

indique un temps en milliseconde après lequel l'animation est arrêtée, 0 = illimité , 1000 = 1s

```
var Direction : integer;
    animD      : TECAAnimationMoveToDirection;
    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;

```



Fig. 126 TECAnimationMoveToDirection

TECAnimationAutoHide

Cette animation cache un élément en basculant sa propriété visible à false, MaxTiming indique le temps en milliseconde après lequel l'élément est caché.

```
var win: TECShapeInfowindow;
...
win: = map. addInfoWindow (lat, lng);

// automatically close the window after 15 seconds
win.Animation := TECAutoHide.Create;
TECAutoHide (win.Animation).MaxTiming := 1000 *
15;
```

Clefs pour les API

Par défaut TECNativeMap utilise les services d'OpenStreetMap.

Si vous souhaitez utiliser MapQuest vous devez [obtenir une clef auprès de MapQuest](#)

Faites de même pour utiliser les services de [MapZen](#) et [MapBox](#).

Mode hors-ligne

Si vous indiquez un répertoire dans la propriété [LocalCache](#), toutes les données des routes seront mise en cache et pourront être réutilisées en mode hors-ligne.

Disponible dans la future version 2.7

TECRouting

Cette classe gère les routes ainsi que la navigation tour par tour.

```
procedure Request(const StartAdress, EndAdress: string;const params:  
string = ""); overload;
```

```
procedure Request(const dLatLngs: array of double;const params:  
string = ""); overload;
```

Calculer une route entre des adresses ou des points

```
// 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 vous permet de passer des paramètres supplémentaires au moteur de routage

property **routeType**: TMQRouteType

Type de route (rtFastest, rtShortest , rtPedestrian, rtBicycle)

property **GroupName** : string

Nom du groupe auquel appartiendra la nouvelle route ([TECShapeLine](#)), par défaut vide

property **Color** : TColor

Couleur de la route

property **Weight**: integer

Épaisseur de la route

property **StartMarkerFilename**: string

Image utilisée pour marquer le départ, par défaut



property **EndMarkerFilename**: string

Image utilisée pour marquer l'arrivée, par défaut



property **RouteDrawingTime**: integer

Temps (en milliseconde) pour afficher la route, permet de réaliser une animation pour un affichage progressif.

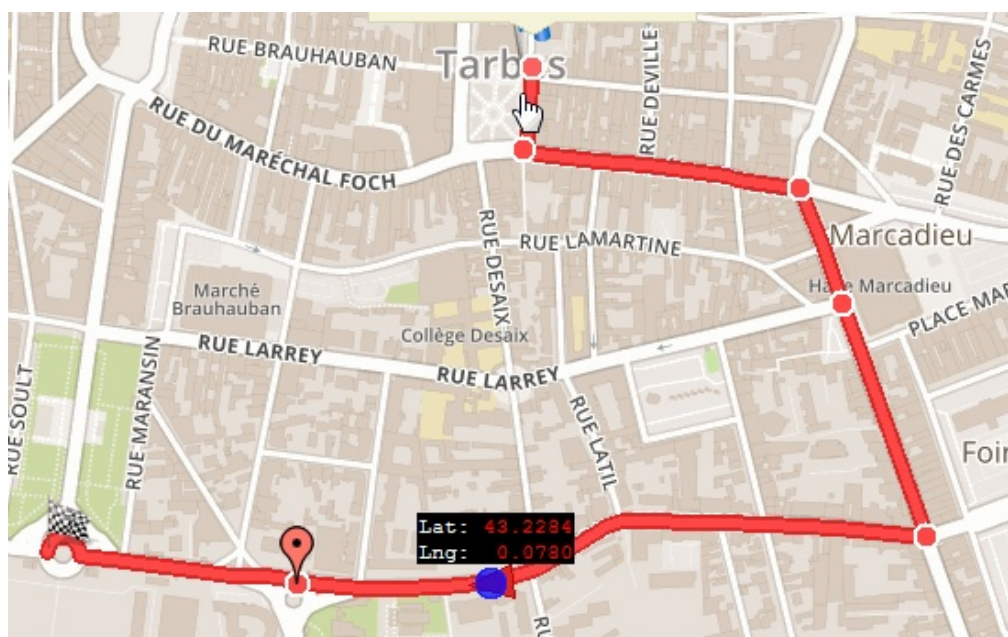
Par défaut 0, la route s'affiche en une seule fois.

property **EditRoute**: TECShapeLine

Permet de modifier la route avec la souris

property **EditOnClick** : boolean

Par défaut un clic sur une route la fait basculer en mode édition, il suffit de cliquer sur les points A ou B pour quitter ce mode



Moteur de routage

procedure **Engine**(value:TRoutingEngine;const ApiKey:string="");

Permet de choisir un moteur de routage, au choix [MapQuest](#) (seul moteur disponible avant), [MapZen](#), [OSRM](#) ou [MapBox](#)

```
// 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]);
```

Utilisez **EngineValidUrl** pour modifier l'url avant quelle ne soit envoyée au moteur

```
// 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]);
```

Définir son propre moteur

Utilisez **EngineUrl**, **EngineExcute** pour utiliser votre propre moteur.

property **EngineUrl** : TECThreadRoutingUrl

property **EngineExecute** : TECThreadRoutingExecute

```
// 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,
```

```
0.09652261) ;
```

EngineUrl et EngineExecute sont exécutés dans un autre thread !

property **EngineName** : string

Donne un nom au moteur, utilisé pour le stockage local

property **EngineKey** : string

Clef pour utiliser le moteur, vous devez obtenir les vôtres auprès de MapQuest, MapZen ou MapBox

property **TurnByTurn**: TECTurnByTurn

Gestion de la navigation tour par tour

4 événements sont disponibles

property **OnAddRoute** : TOnAddRoute

Déclenché lorsque la route a été créée

property **OnChangeRoute** : TOnChangeRoute

Déclenché lorsque la route a été modifiée

property **OnErrorRoute** : TOnErrorRoute

Déclenché si la route n'a pas été créée

property **OnDrawRoute** : TOnDrawRoute

Déclenché lorsque la route a été affichée intégralement (utile si RouteDrawingTime > 0)

```
// 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]);
```

TECTurnByTurn

property **line**: TECShapeLine

Route à suivre

function **Position**(const Lat, Lng: double):boolean

Indiquez votre position GPS

property **AlertDistance**: integer

Par défaut 300 mètres

property **ExecutionDistance**: integer

Par défaut 100 mètres

property **ErrorDistance**: integer

Par défaut 30 mètres

événements disponibles

property **OnAlert**: TOnTurnByTurnAlert

Déclenché lorsque vous arrivez à moins de **AlertDistance** mètres de la cible

property **OnInstruction**: TOnTurnByTurnInstruction

Déclenché lorsque vous arrivez à moins de **ExecutionDistance** mètres de la cible

property **OnArrival**: TNotifyEvent

property **OnError**: TOnTurnByTurnError

property **OnAfterInstruction**: TOnTurnByTurnInstruction

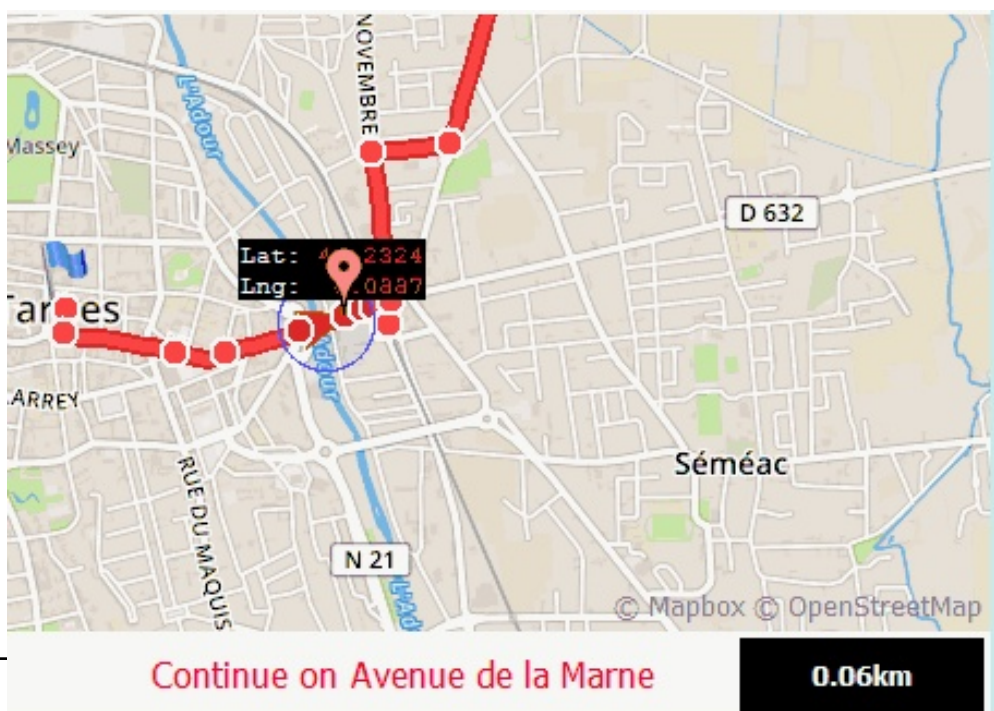
property **OnConnectRoute**: TNotifyEvent

Déclenché par la connexion d'une route

property **OnDisconnectRoute**: TNotifyEvent

Déclenché par la déconnection d'une route

```
// 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]);
```



La demo DemoNativeRoute a été réécrite pour utiliser ces nouvelles fonctionnalités.

Anciennes méthodes, privilégiez les nouvelles fonctionnalités disponible depuis la version 2.7

Recherche de routes

Vous avez à votre disposition 4 routines (2 synchrones et 2 asynchrones) pour calculer un itinéraire.

```
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 = " )
```

Ces fonctions permettent d'obtenir les informations d'une route sans effectuer de tracé, les procédures asynchrones vont s'exécuter en arrière-plan et déclencher l'événement **OnRoutePath** lorsque les données sont disponibles.

Voir open.mapquestapi.com/directions/ pour le paramètre Params

*Si vous ne libérez pas le **TECMapRoutePath** obtenu, il le sera automatiquement lors de la destruction de **TECNativeMap***

Afficher la route

Vous pouvez créer une **TECShapeLine** à partir d'un **TECMapRoutePath** avec la fonction **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;
```

Vous pouvez aussi directement modifier une `TECShapeLine` avec la 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;
```

Modifier la route à la souris

Pour cela vous allez devoir rajouter l'unité **uecEditNativeLine** (`FMX.uecEditNativeLine` sous FireMonkey)

Vous pourrez alors utiliser la classe **TecNativeLineToRoute** pour pouvoir modifier dynamiquement le trajet.

property **Line** : `TECShapeLine`

`TECShapeLine` à modifier

property **RouteType** : `TMQRRouteType`

type de route (`rtFastest`, `rtShortest`, `rtPedestrian`, `rtBicycle`)

property **Modified** : boolean

Indique si la route à été modifié

property **OnClick** : TOnShapeMouseEvent

Permet de réagir au clic sur la route

property **OnMouseOver** : TOnShapeMouseEvent

Permet de réagir au survol de la route

property **OnChange** : TNotifyEvent

Événement déclenché à chaque modification de la route

Exemple d'utilisation

```
// Delphi map component EMap

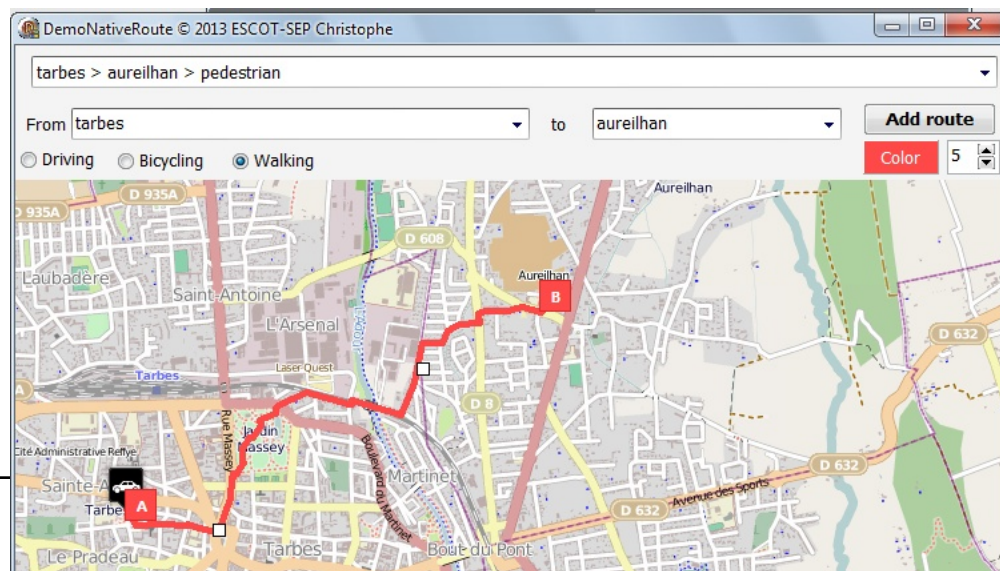
var ecNativeLineToRoute : TecNativeLineToRoute;
...
ecNativeLineToRoute := TecNativeLineToRoute.create;

// edit route
ecNativeLineToRoute.Line := map.shapes.Lines[0];

// end edit route
ecNativeLineToRoute.Line := nil;
...
ecNativeLineToRoute.free;
```

DemoNativeRoute

Une démonstration est disponible pour bien comprendre la gestion des routes.



TECNativeLayer est la classe de base des layers qui vous permettent de réagir en fonction de la zone affichée et des actions de la souris.

```
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;
```

Vous pouvez soit vous brancher sur `OnMapHiResChange` soit surcharger la procédure `doOnMapHiResChange(sender : TObject)` pour que votre layer s'adapte au de résolution

Panoramio

Google a fermé Panoramio le 4 novembre 2016, le service n'est plus fonctionnel

Un layer **Panoramio** est intégré dans **TECNativeMap** au travers de la propriété **PanoramioLayer**


```
// show panoramio layer  
map.PanoramioLayer := true;
```

L'événement **OnPanoramioClick**(sender: TObject; const Item: TECShape; const ownerId, ownerName, PhotoId, PhotoDate, PhotoTitle, PhotoUrl, copyright: string) est déclenché lors d'un clic sur un élément Panoramio.



Fig. 129 Click sur un élément Panoramio

TECNativePlaceLayer

Vous permet d'afficher automatiquement le résultat d'une recherche

Pour utiliser ce layer vous devez incorporer l'unité `uecNativePlaceLayer` ou `FMX.uecNativePlaceLayer` selon que vous utilisiez la version `VCL` ou `FireMonkey`

Exemple : un layer qui affiche les 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;

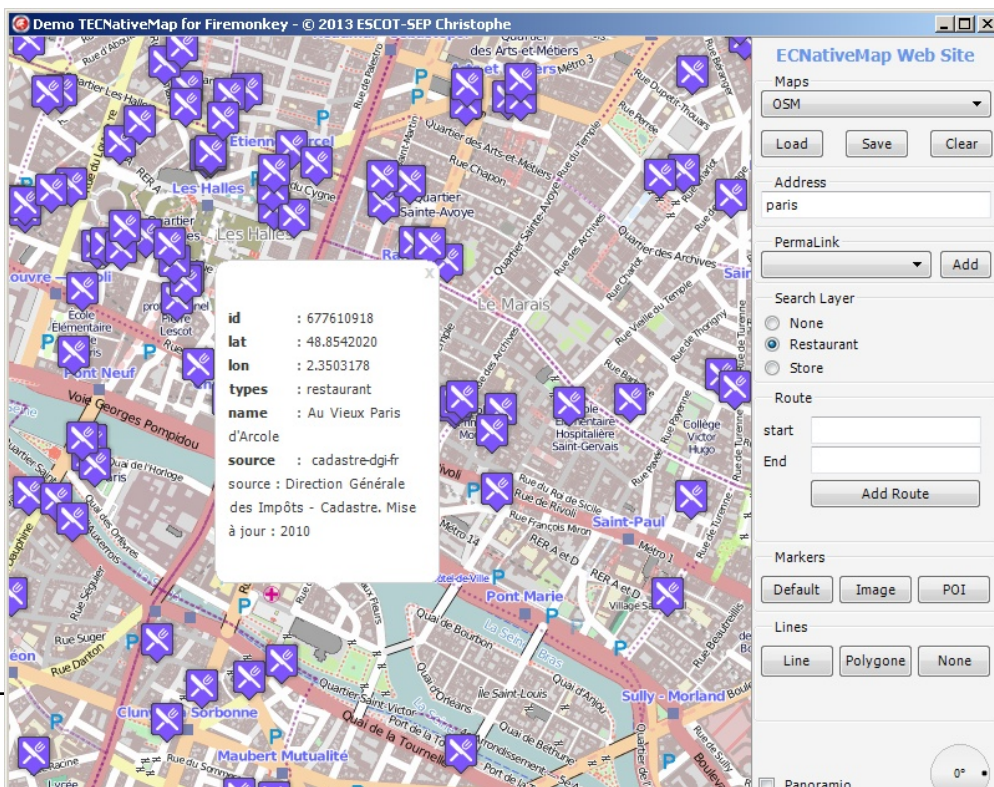
if FPlacesLayer.Shapes.InfoWindows.Count=0 then
begin
    FPlacesLayer.Shapes.InfoWindows.add(0,0,'');
    FPlacesLayer.Shapes.InfoWindows[0].zindex := 100;
end;

FPlacesLayer.Shapes.InfoWindows[0].content := Content;
FPlacesLayer.Shapes.InfoWindows[0]
].SetPosition(Shape.Latitude,shape.Longitude);
FPlacesLayer.Shapes.InfoWindows[0].Visible := true;

end;

end;

```



Par défaut les éléments du layer sont des [TECShapeMarker](#) mais vous pouvez changer le type de TECShape en redéfinissant **TECNativePlaceLayer.doCreateShape(SearchResult : TECPlaceResult):TECShape**;

XapiLayer

XapiLayer utilise TECNativePlaceLayer et est directement intégré dans TECNativeMap pour une utilisation simplifiée.

La recherche s'effectue dans la zone affichée par votre carte et est mise à jour à chaque déplacement

property **Shapes**:TECShapes

Le groupe qui contient les éléments affichés, il est nommé 'XAPI'

property **Search** : string

La recherche s'effectue en faisant une requête sur un serveur [XAPI](#)

Vous pouvez rechercher tous les [tags OSM](#) en utilisant une syntaxe du type 'node[key=value]'

```
// search bus stop
map.XapiLayer.Search := 'node[highway=bus_stop]';
```

Pour les nodes vous pouvez utiliser une syntaxe simplifiée.

```
// search bus stop
map.XapiLayer.Search := 'highway=bus_stop';
```

Pour les clef [amenity](#) vous pouvez simplifier encore plus.

```
// search all restaurant, bar and café
map.XapiLayer.Search := 'restaurant|bar|cafe';
```

property **Visible** : boolean

Affiche / cache le layer

property **MaxItem** : integer

Nombre d'éléments maximum

property **OnClick** : TOnShape

Cet événement est déclenché lors d'un click sur un item du layer

property **OnChange** : TNotifyEvent

Déclenché après chaque requête

```
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;
```

Utilisez les [styles](#) pour habiller vos éléments

```
Selector := '#' + map.XapiLayer.Shapes.Name +
  '.marker.amenity';

map.styles.addRule(Selector + ':restaurant' +
  '{graphic:base64,iVBORw0KGgoAAAANSUhEUgAAABIAAAABAMAAAABI '
  +
  'sxEJAAAAGXRFWHRTb2Z0d2FyZQBBZG9iZSBJbWFnZVJlYWR5ccllPAAAABJQTFRFAAAA//AAAA '
  +
```



```

'AAAAAAAA/h6U3wAAAAV0Uk5TAAAQgL++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+D02eUosKQAZAjIM5
+
' JUBsgE2eEotQDIHyHXDJU2JRPyn8wQmgcfCfTPyelHghGw8/i4D4PBrbgVYWYcMGtLAoAakUmQ8V
+
' TOALyFXDGHZUtEqCXRbCy8DzNLAKlNqTMn0ATi0AVI5LYf1o17/1I7Pk4a18q5Zv3BKsPKpUMCvQ
+
)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;}');

```

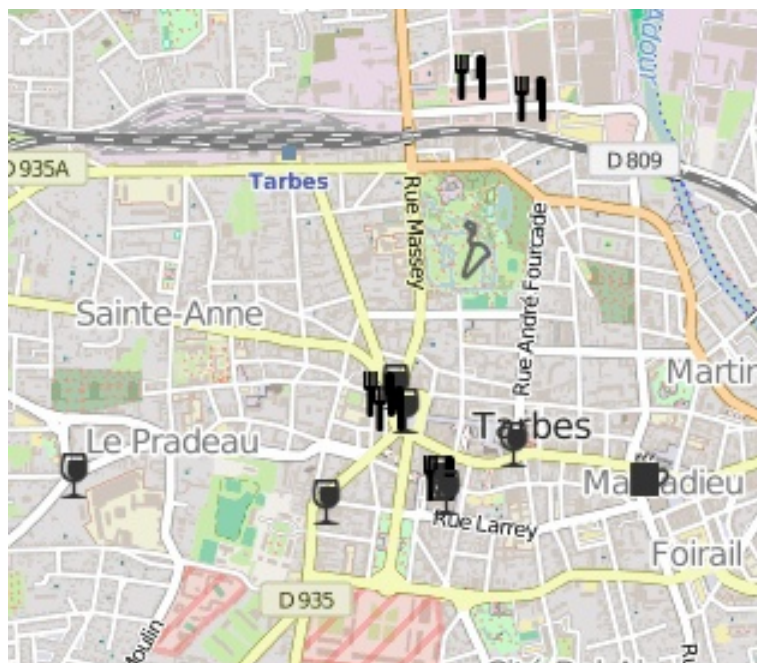


Fig. 131 'restaurant|bar|cafe'

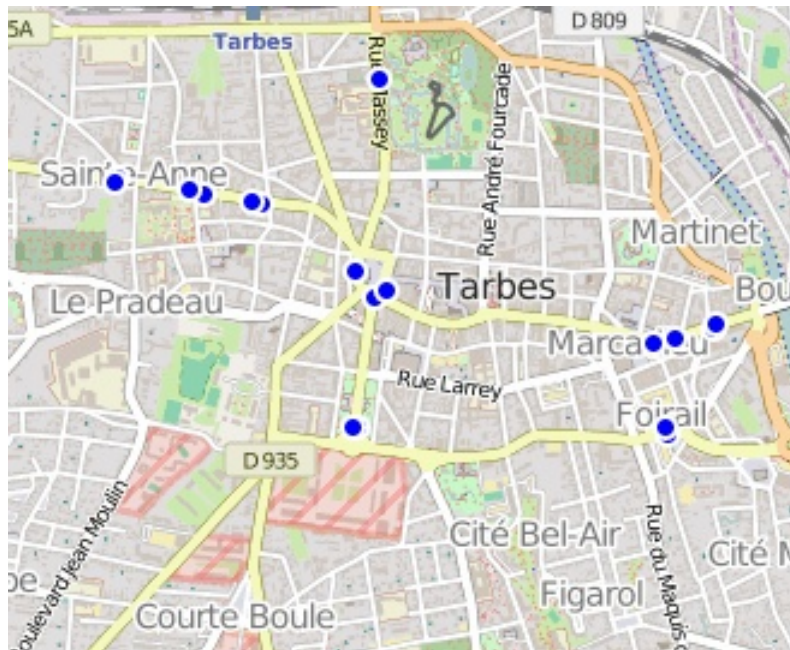
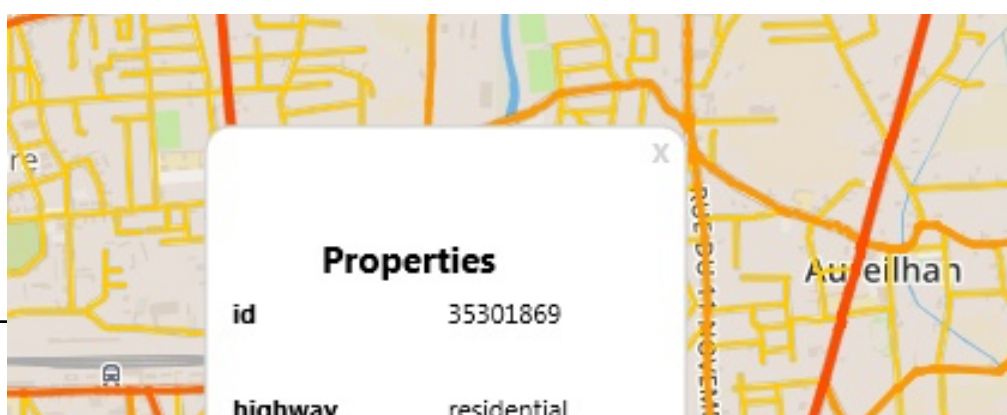


Fig. 132 'highway=bus_stop'

Way

Avec une syntaxe du type `way[key=value]` pour pouvez extraire des chemins.

```
// 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
```



Vous pouvez aussi trouver les jonctions entre les routes.

```
map.XapiLayer.Junction := true;
map.XapiLayer.Search :=
;way[highway=unclassified|road|motorway|trunk|primary|secondary|tertiary|resi
```

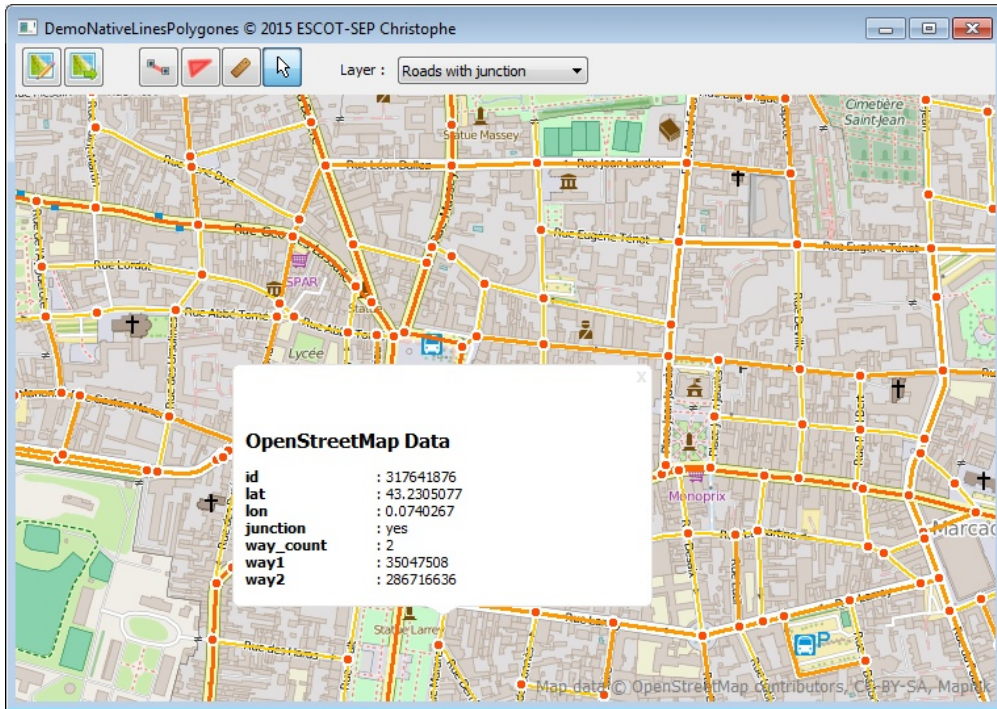


Fig. 134 Xapi Roads & Junction

Recherche manuelle

Lorsque le layer est visible la recherche est synchronisée avec la zone visible de votre carte.

Pour pouvoir effectuer une recherche sur n'importe quelle zone vous devez cacher le layer et utiliser **Bound** pour délimiter la zone de recherche.

```
// 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

Ce layer vous apporte le support du format [UTFGrid](#)

Exemple : traduction en Delphi de [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>'

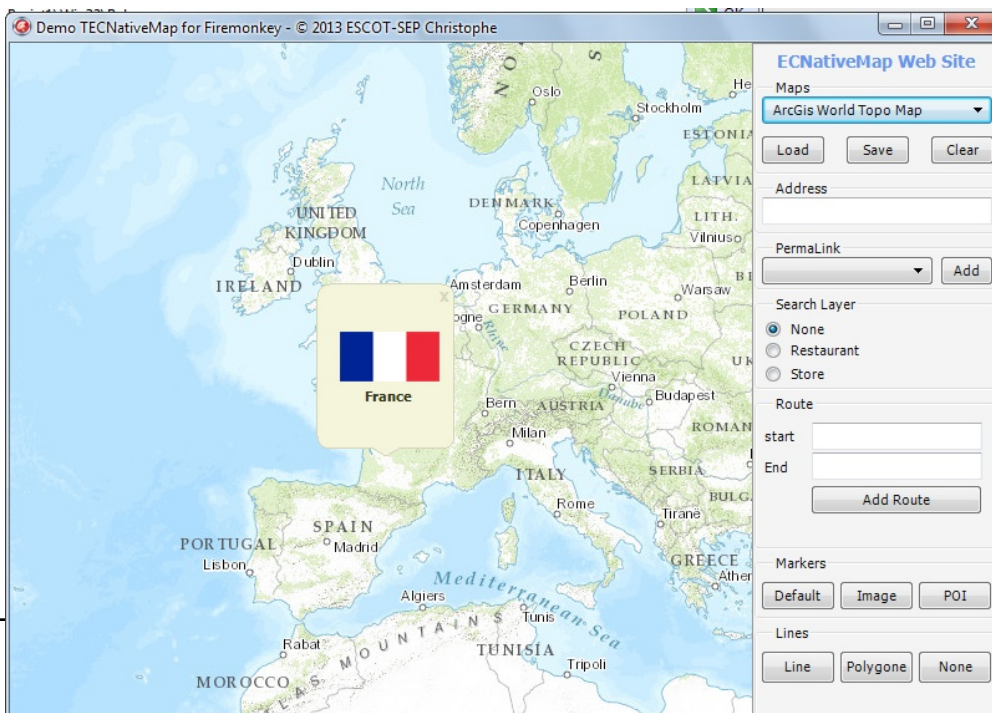
    +FUTFLayer.data['admin']+ '</h3>';

    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

Ce layer affiche une infoWindow au-dessus des TECShape, elle est mise à jour et déplacée automatiquement en fonction du Hint et de la position des TECShape

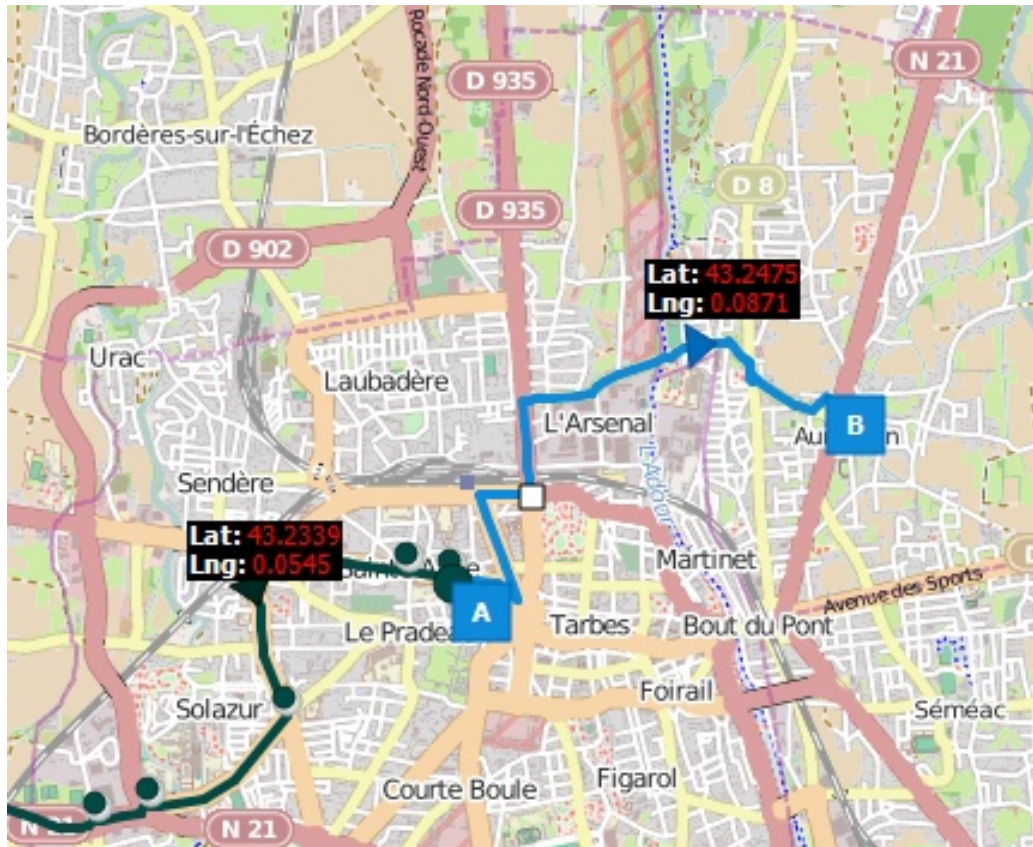


Fig. 136 Using labels to display positions

Pour utiliser ce layer vous devez incorporer l'unité uecNativeLabelLayer ou FMX.uecNativeLabelLayer selon que vous utilisiez la version VCL ou 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]);
```

Consultez les sources des démos Firemonkey et DemoNativeRoute pour un exemple d'utilisation

Mappilary

Mappilary est un service du même type que Google Street Map, il permet d'afficher des photos au niveau des rues, l'avantage est que vous pouvez vous-même l'enrichir en enregistrant vos parcours.

TECNativeMap se contente d'utiliser les informations en libre accès au près de Mappilary, vous devez obtenir une clef pour utiliser les service de Mappilary.

Pour cela allez sur le site www.mapillary.com, connectez-vous, dans la section **Integrations** ajoutez vos applications, vous obtiendrez alors un **Client ID** qui vous servira de clef.

*Pour utiliser mappilary vous devez incorporer l'unité **uecMappilary** ou **FMX.uecMappilary** selon que vous utilisiez la version **VCL** ou **FireMonkey***

Utilisez **TECMappilaryLayer** pour incorporer un layer Mappilary.

```
FMappilaryLayer := TECMappilaryLayer.Create(map);
```

```
FMappillaryLayer.ClientId := 'HERE-YOUR-CLIENT-ID' ;

FMappillaryLayer.Visible := true;
```

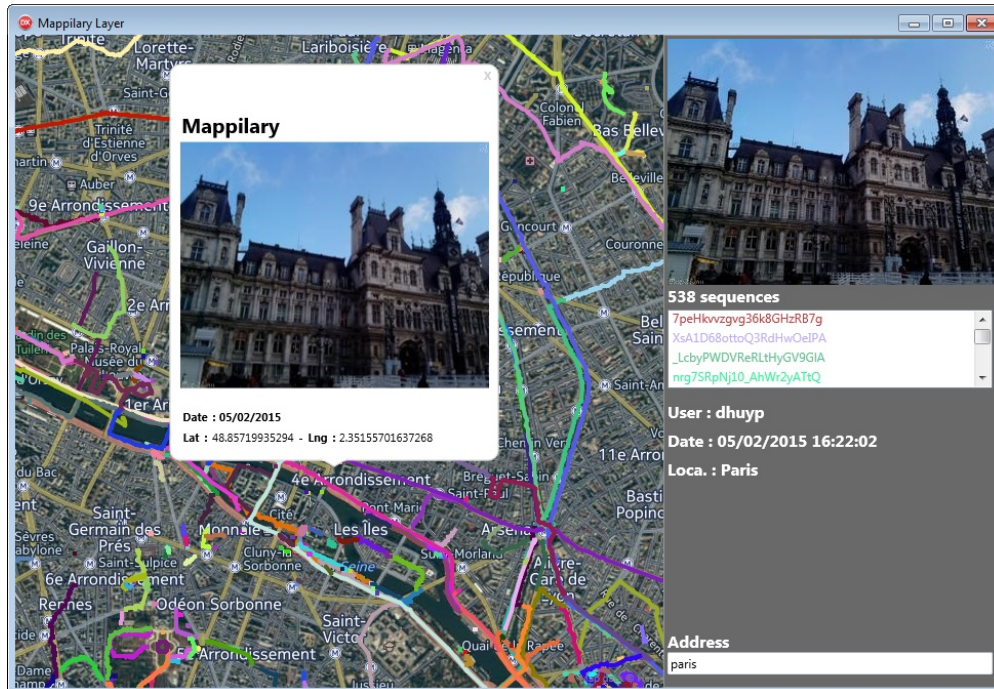


Fig. 137 Demo Mappillary

TECMappillaryLayer

procedure **CancelSearchSequence**;

Annule une recherche de sequences.

procedure **CancelSearchImageClose**;

Annule une recherche d'images

procedure **SearchImageClose**(const Lat, Lng: double; Distance: integer);

Lance une recherche d'images situées à une Distance maximale, exprimée en mètres, du

point Lat,Lng

Vous pouvez vous connecter à l'évènement **OnImages** pour être averti d'une réponse.

La propriété **Images** contient la liste des images.

function **FindImageClose**(const Lat, Lng: double; Distance: integer; var seq : TMapillarySequence; var PhotoIndex : integer) : integer;

Trouver la photo la plus proche parmi celles affichées dans le layer, la recherche est en local.

Lat,Lng indique le point de départ de la recherche

Distance indique la distance maximale en mètres de la zone de recherche

Seq contiendra la séquence trouvée ou nil

PhotoIndex contiendra l'index de la photo dans seq ou -1

Retourne la distance en mètres entre le point de recherche et la photo, 0 si pas de photo

function **LoadImage**(const Key: string): TBitmap; overload;

Obtient un bitmap contenant l'image dont on a passé la clef.

Attention vous ne devez pas libérer le bitmap obtenu, pour le conserver vous devez en faire une copie !

function **LoadImage**(const Key: string; Bitmap:TBitmap):boolean;

rempli un bitmap avec l'image dont on a passé la clef.

Attention vous êtes responsable de la création et de la libération du 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;

Obtient un bitmap contenant l'image dont on a passé la clef, **cette fonction n'est pas bloquante**, **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;

Obtient les informations d'une 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;

Obtient l'url d'une image dont on passe la clef.

function **LocalPathImage**(const Key:string):string;

Obtient le chemin d'une image dans le cache local

property **CanceledSearchSequence** : boolean ;

Permet de savoir si une recherche de séquences a été annulée.

property **CanceledSearchImageClose** : boolean ;

Permet de savoir si une recherche d'images a été annulée.

property **ClientId**: string ;

OBLIGATOIRE la clef pour utiliser les services de Mappilary

property **Connexion** : TMappilaryConnexion ;

Type de connexion

mcOnlyMappilaryServer : uniquement par internet

mcOnlyLocal : uniquement en local

mcLocalOrServer : local et si besoin internet (valeur par défaut)

property **LocalCache**: string ;

Répertoire du cache local

property **Thumb**: TMappilaryThumbSize ;

Taille des images (Thumb320, Thumb640, Thumb1024, Thumb2048)

par défaut **Thumb320**

property **Sequences**: TMappilarySequenceList ;

Liste des séquences trouvée dans la zone visible de la carte

La propriété est mise à jour automatiquement à chaque déplacement de la carte

property **Images** : TMapillaryImageList;

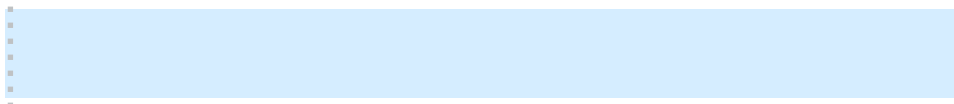
Liste des images obtenue avec un appel à **SearchImageClose**

property **MaxDayInCache** : integer;

Durée de vie des données stockées dans le cache local, par défaut 30 jours

property **OnClick**: TOnMAppillaryClick ;

Déclenché lors d'un click sur un élément mappillary



property **OnSequenceColor**:TOnMappilaySequenceColor;

Permet d'attribuer une couleur à une séquence.

Par défaut une couleur unique est attribuée en fonction de la clef de la séquence, une séquence aura donc toujours la même couleur.

```

Layer.OnSequenceColor := doSequenceColor;

procedure TForm2.doSequenceColor(Layer
: TECMappillaryLayer;

                                MappillarySequence: TMapillarySequence;
                                var SequenceColor:TColor);

begin
    SequenceColor := your_color;
end;
```

property **OnSequences**: TOnMappilartySequences ;

Déclenché lorsque des séquences on été trouvées.

La recherche de séquence est déclenchée lors du déplacement de la carte

```

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;

Déclenché lors de la réception des images après un appel à SearchImagesClose

```


```

TMapillaryAPI

TMapillaryLayer utilise en interne la classe TMapillaryAPI, vous pouvez l'utiliser directement si vous ne souhaitez pas afficher le layer.

Le fonctionnement est identique, il vous suffira simplement d'indiquer la zone de recherche avec la fonction **Bound**(NELat, NELng, SWLat, SWLng: double) puis **SearchSequence** pour lancer la recherche.

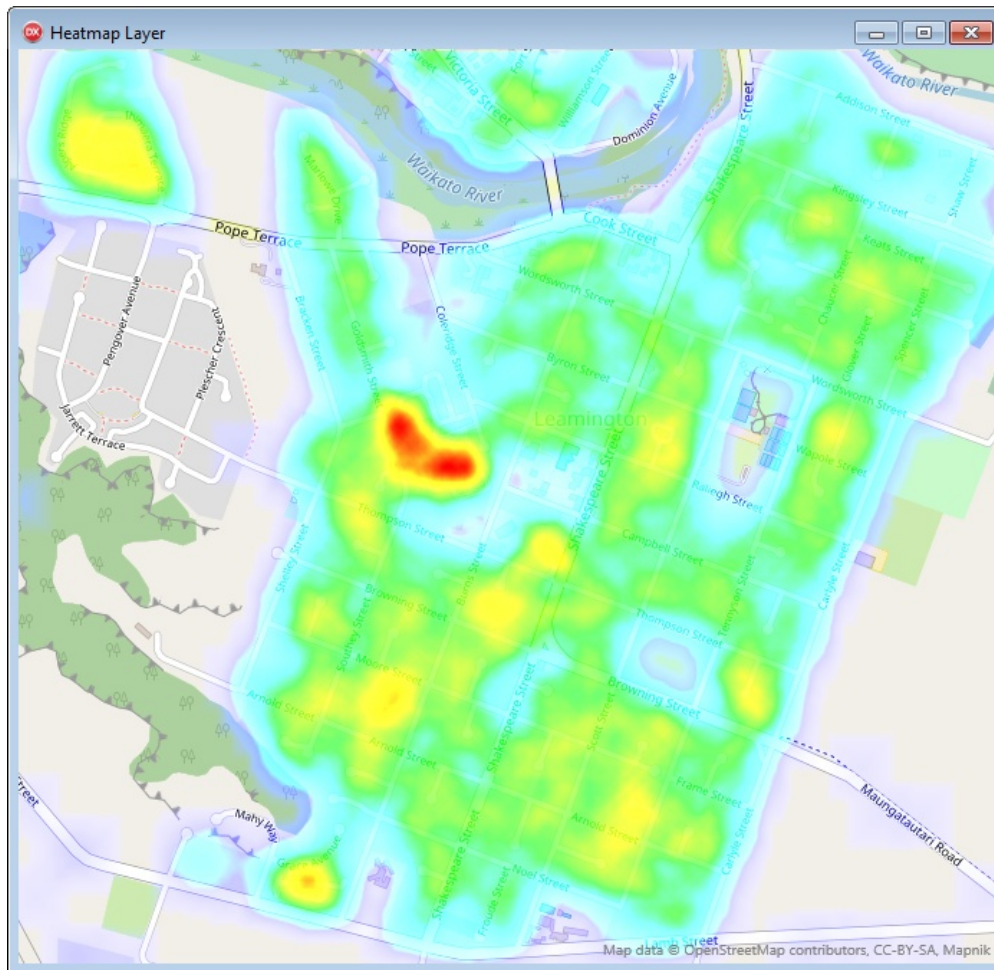


Fig. 139 Heatmap layer

TECHeatmapLayer est la classe qui permet de gérer ce type de layer.

constructor **Create(_FMap: TECNativeMap);**

```
// create heatmap layer
HeatmapLayer := TECHeatmapLayer.Create(map);
```

procedure **Clear;**

Effacer l'ensemble des données

procedure **Add(const Latitude, Longitude: double; const value: double = 1);**

Ajouter un point géographique et lui assigner une valeur, vous pouvez faire plusieurs ajouts sur un même point.

procedure **Update;**

Après l'ajout des points, appelez Update pour actualiser le layer

property **Palette: THeatPalette**

Palette permet de gérer les couleurs qui sont utilisées pour créer le dégradé

```
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;

Détermine si le point est affiché sous la forme d'un disque, sinon c'est un carré

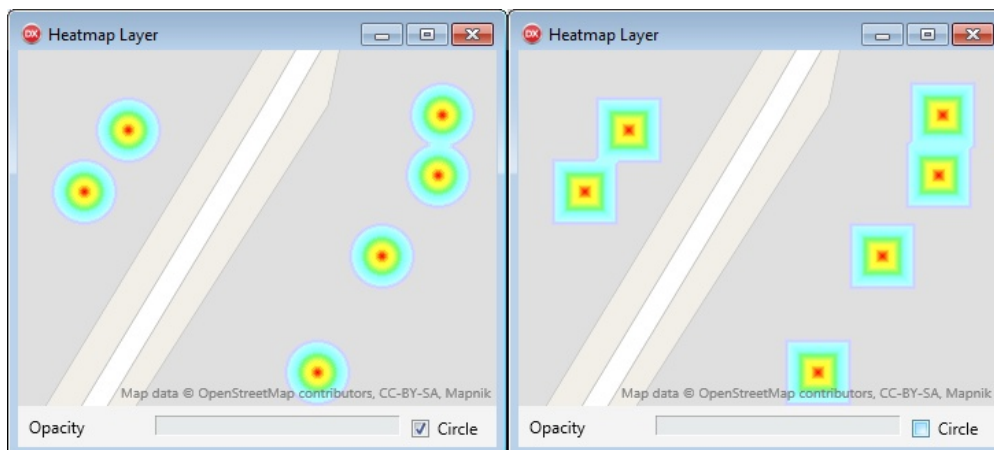


Fig. 140 Style d'affiche des points

Property **Radius**: integer;

Taille du point

property **Opacity** : byte;

Change l'opacité du layer (de 0 à 100)

property **MinZoom**: byte ;

Zoom minimal pour que le layer s'affiche

property **MaxZoom**: byte ;

Zoom maximal pour que le layer s'affiche

property **GroupZIndex:** integer ;

Zindex du groupe contenant l'ensemble des heatmaps, l'affichage s'effectue dans l'ordre croissant des ZIndex

property **ZIndex:** integer ;

ZIndex du layer par rapport aux autres heatmap

property **OnUpdate:** TNotifyEvent;

Événement déclenché lorsque la carte thermique a été généré

Weather Layer

En utilisant les services d'[OpenWeathermap.org](https://openweathermap.org) vous pourrez superposer des layers owPrecipitation, owSnow, owClouds, owPressure, owTemp, owWind

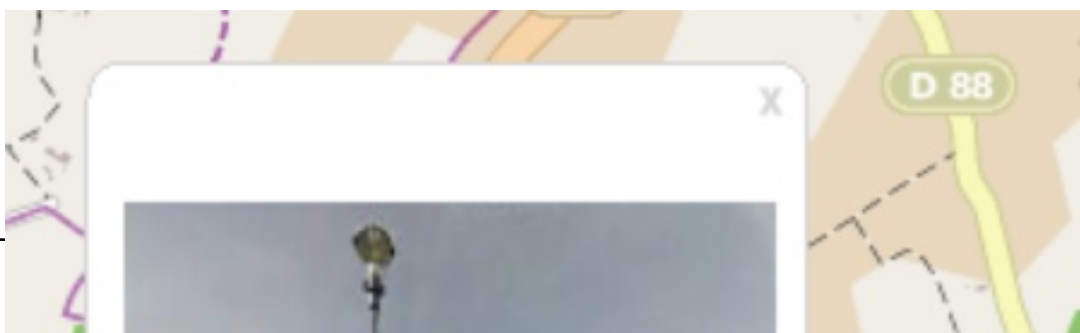
```
map.OpenWeatherTilesLayer.Key := 'your api key';  
    // see pressure  
map.OpenWeatherTilesLayer.Add(owPressure) ;  
    // see precipitation  
map.OpenWeatherTilesLayer.Add(owPrecipitation) ;  
    // remove pressure  
map.OpenWeatherTilesLayer.remove(owPressure) ;
```

ECNativeMap fonctionne sous Android, vous pouvez [télécharger et installer l'apk de démonstration](#).



Fig. 141 Demo ECNativeMap for Android

Un appui long sur la carte pose un élément, son apparence est aléatoire, vous pouvez le déplacer et cliquer dessus pour obtenir son adresse.





Zoomer et dézoomer



Par un effet de rotation vous pouvez changer l'orientation de votre carte

Le switch "**Find me**" permet d'afficher votre localisation



Vous permet d'aller vers une destination, ou de créer une route en remplissant les 2 champs

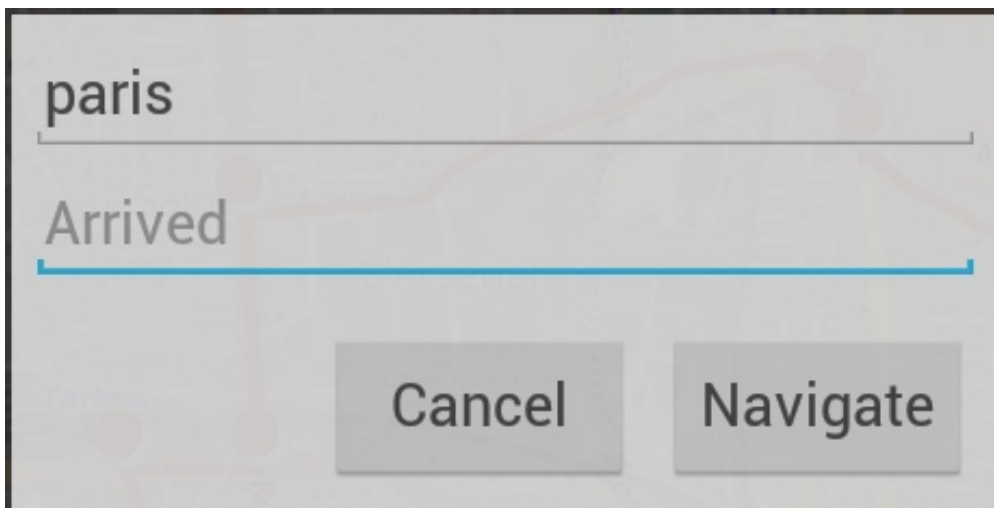


Fig. 143 Goto Paris

Un suivi automatique va alors être mis en place, cliquez sur le triangle pour le désactiver ou le réactiver

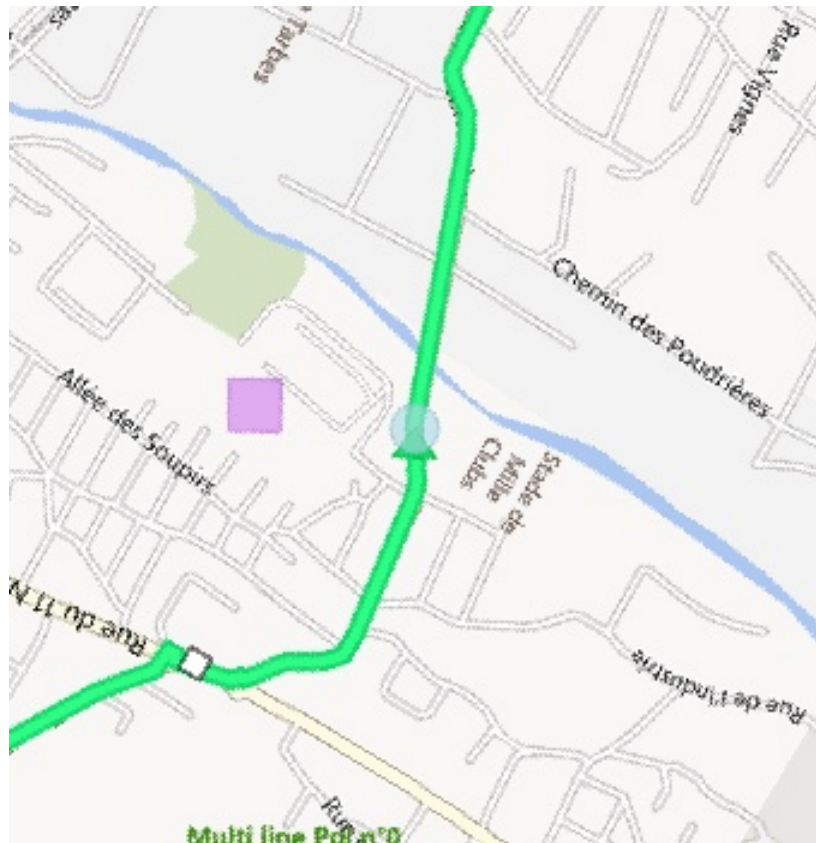
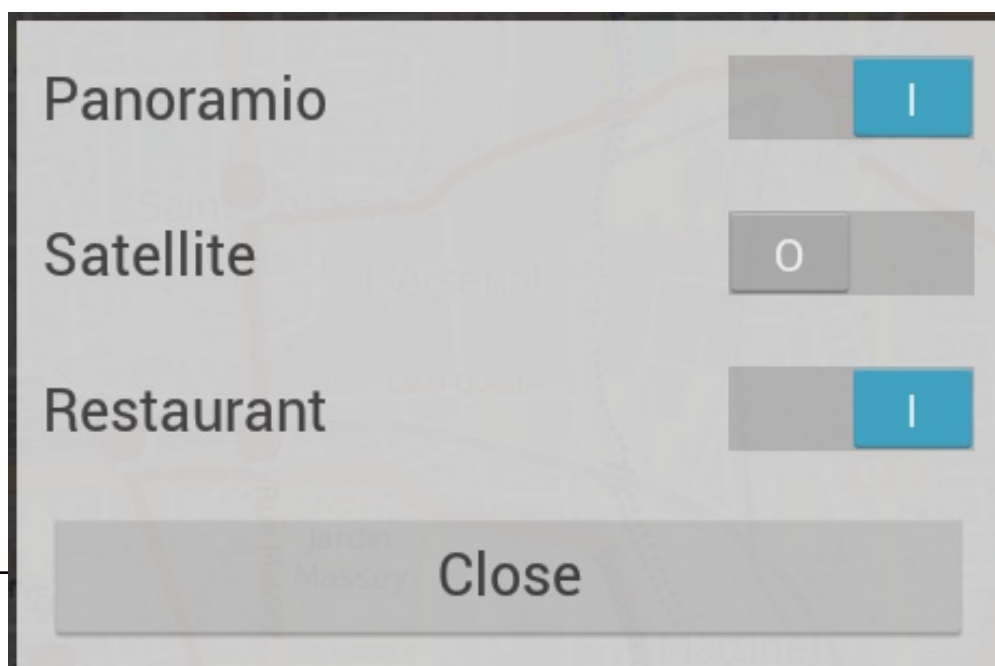


Fig. 144 Tracking



Affiche les images panoramio, satellite et les restaurants.





Pour gérer le cache, effacer la carte ou la sauvegarder, gérer les effets du zoom, l'inertie et du mode [haute résolution](#).

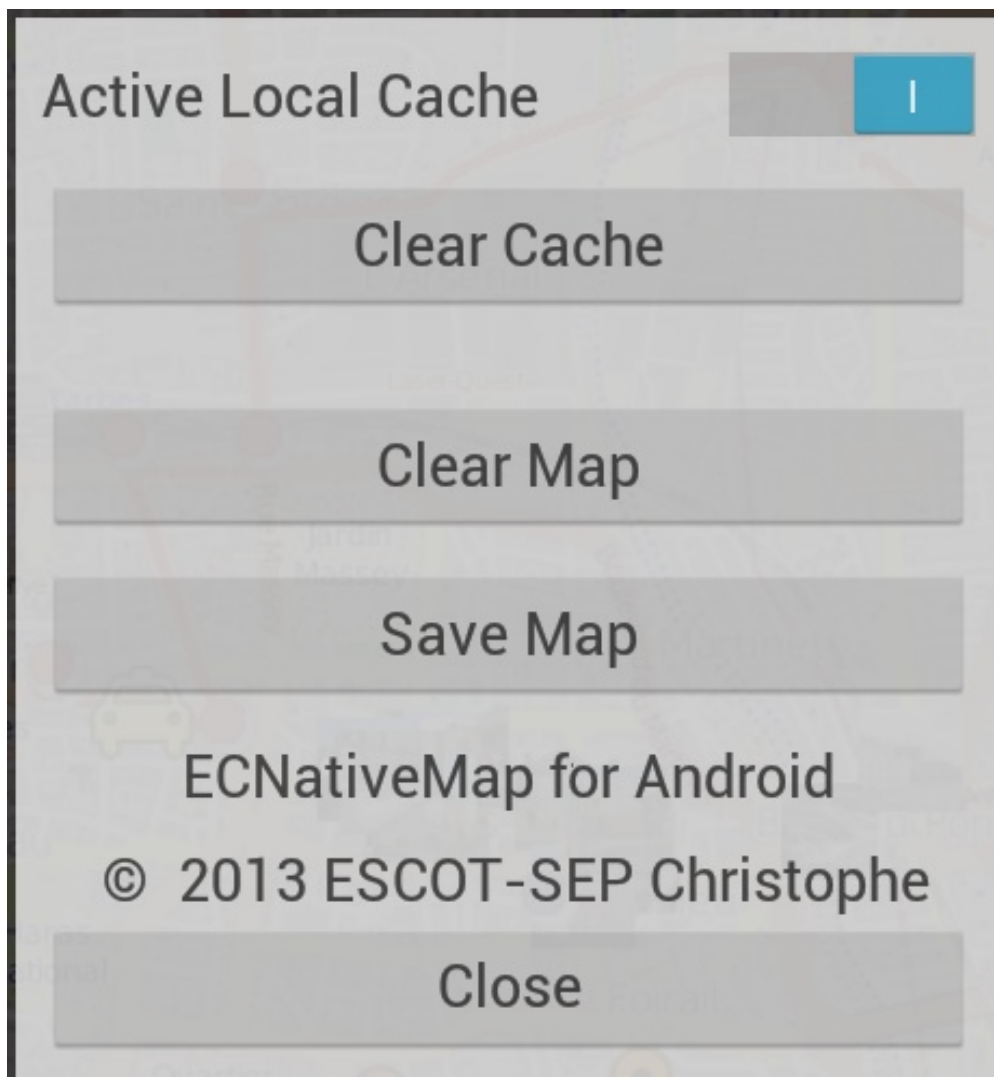


Fig. 146 Save your Map

Donnée OSM

Vous pouvez télécharger un apk pour android d'une mini demo qui affiche les données OSM d'une petite zone.



Fig. 147 Android Test OLT

TECNativeMap est aussi disponible pour **iOS** !

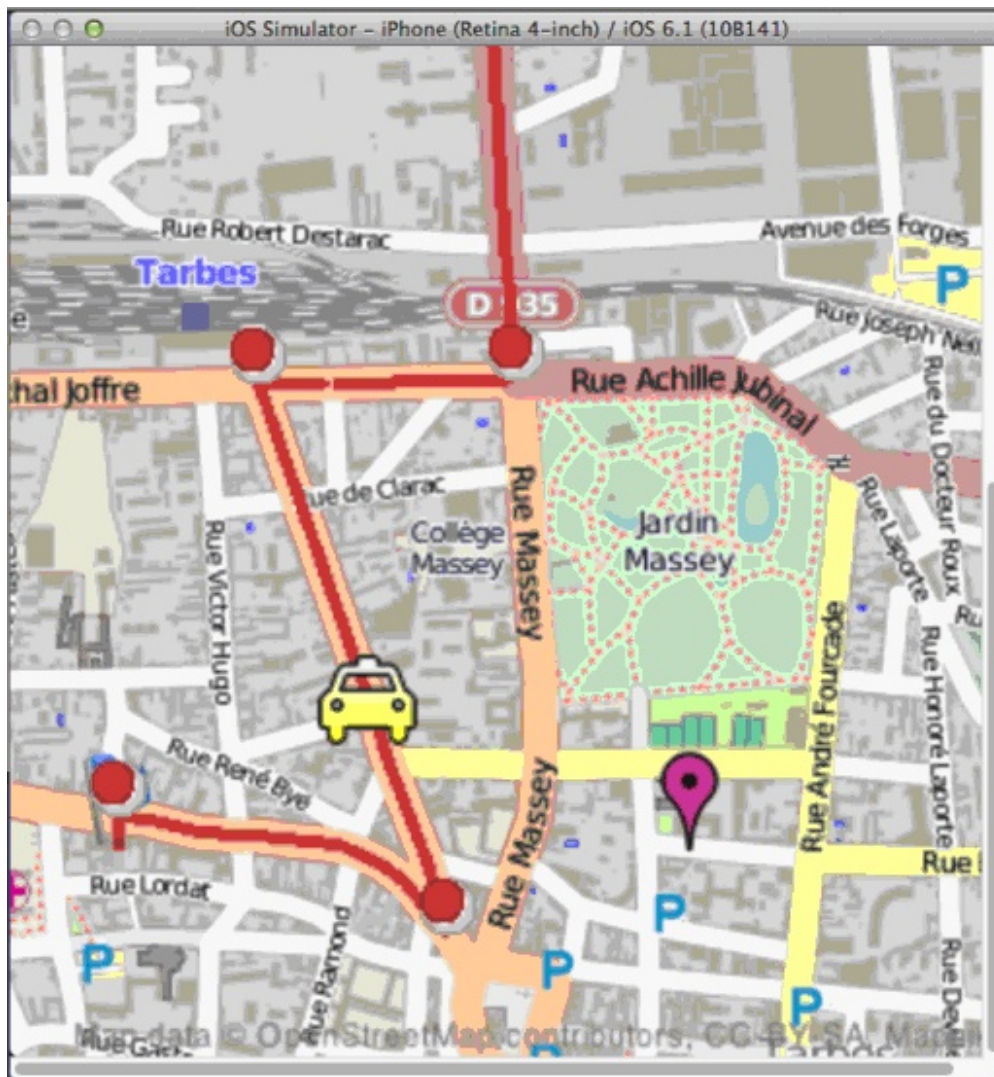


Fig. 148 ECNativeMap on iOS simulator

Autres composants disponibles dans la suite complète TECMap

[TECCesiumMap](#)

Cesium est une bibliothèque Javascript pour gérer des globes 3D et des cartes 2D, plus d'informations sur le site www.cesiumjs.org[Lire](#)

[TECMapAdressEdit](#)

TECMapAdressEdit est un composant descendant de TEdit qui dispose d'une assistance automatique pour la saisie des adresses

[Lire](#)

[TECStaticMap](#)

TEStaticMap est un descendant de TImage qui va vous permettre d'afficher une carte statique en utilisant les services de [Google Map](#), [Google StreetView](#) et [OpenMapQuest](#)

[Lire](#)

[TECVelib](#)

TECVelib est une classe qui vous donne accès à l'api des [Vélos en livre-service JCDecaux \(Vélib\)](#)

[Lire](#)

Cesium est une bibliothèque Javascript pour gérer des globes 3D et des cartes 2D, plus d'informations sur le site www.cesiumjs.org

TECCesiumMap est un composant qui vous permet d'utiliser **Cesium** dans vos développements Delphi, il n'est pour l'instant disponible que pour la VCL (Windows)

Internet Explorer 11 est le minimum requis pour utiliser Cesium, TECCesiumMap ne fonctionne pas sous XP !



Fig. 149 TECCesiumMap

Ce composant est intégré dans la suite TECMap

Installation

Par défaut TECCesiumMap utilise les serveurs de cesiumjs.org, mais pour la mise en production de vos développements je vous conseille fortement d'installer cesium sur votre propre serveur Web.

Téléchargez la dernière archive de [Cesium](#) puis dézippez la simplement sur votre serveur.

TECesiumMap dispose de la propriété [WebServer](#) qu'il vous suffit de renseigner avec le répertoire racine de votre installation.

Utilisez **CreateRuntimeCesiumMap** pour une création dynamique

```
// function CreateRuntimeCesiumMap(AOwner: TComponent=nil;AParent:TWinControl=nil)
cesium := CreateRuntimeCesiumMap(self,panel1);
```

***Vous devez** obtenir une clef pour utiliser l'api de Cesium ,
branchez-vous sur l'événement OnInitialize pour
connecter votre Token.*

```
map.OnInitialize := doInitialize;
...

procedure TForm.doInitialize(Sender: TObject);
begin
  map.IonAccessToken := 'your cesium ion token';
end;
```


TCesiumControl

La structure est la même que pour [TECNativeMap](#), les éléments fonctionnent de la même manière et vous retrouvez la notions de [groupes](#), la [géolocalisation](#) et le [calcul des routes](#) sont identiques

Ce composant non visuel est celui qui manipule vraiment Cesium, TECCesiumMap est le composant visuel que vous installez sur vos formes et qui utilise TCesiumControl

```
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;
```

*Contrairement à TECNativeMap dans TECCesiumControl
vous indiquez des coordonnées écran pour
positionner votre fenêtre d'information*

```
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: TMQRouteTyp = rtFastest; const params: string = ""): TECroutePath;

function **GetRoutePathFrom**(const dLatLngs: array of double; const routeType: TMQRouteTyp = 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: TMQRouteTyp = rtFastest; const params: string = ""): TECroutePath;

property **Address**: string

property **MouseLatLng**: TLatLng

property **MouseAlt**: double

property **ScreenShot**: TBitmap

property **ClickX** : integer read FClickX;

Position en X (coordonnée écran) du click

property **ClickY** : integer read FClickY;

Position en Y (coordonnée écran) du click

property **MouseX** : integer read FMouseX;

Position en X (coordonnée écran) de la souris

property **MouseY** : integer read FMouseY;

Position en Y (coordonnée écran) de la souris

property **Shapes**: TECCesiumShapes read FShapes;

property **HintInfoWindow** : TECCesiumShapeInfoWindow
read getHintInfoWindow;

Donne accès au TECCesiumShapeInfoWindow qui gère les info-bulles des éléments, utile pour modifier le 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;

[Url de votre serveur Cesium](#)

property **BingKey**: string

property **Latitude**: double ;

property **Longitude**: double ;

property **Altitude**: double;

property **Zoom**: double;

property **Draggable**;

property **TileServer**: TTileServer

Les serveurs suivant sont disponibles :

- tsOpenMapQuest
- tsOSM
- tsOpenCycleMap
- tsArcGisWorldTopoMap
- tsArcGisWorldStreetMap
- tsArcGisWorldImagery
- tsBingRoad
- tsBingAerial
- tsBingAerialLabels

Il est conseillé de remplir la propriété **BingKey** avec [votre clef Bing](#) pour utiliser les tuiles de

Bing Maps (tsBingRoad, tsBingAerial, tsBingAerialLabels)

```
map.BingKey      := YOUR_BING_KEY
map.TileServer   := tsBingRoad;
```

property **Terrain**: boolean;

Active le relief du terrain

property **SceneMode**: TCesiumMode;

Sélectionne le type d'affichage, par défaut 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>';
```



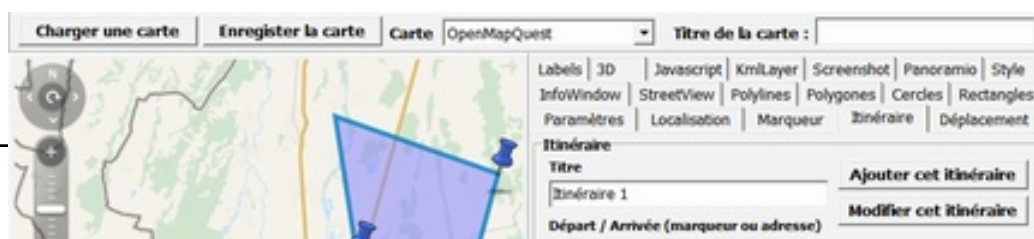
Fig. 150 Marker by default

```
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;
```




```
// 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. 152 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';;
```

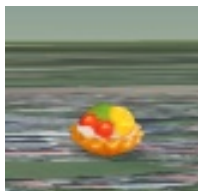


Fig. 153 Marker with Png

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. 154 TECCesiumShapeLine

TECCesiumShapePolygone

```
var polygon : TECCesiumShapePolygone;

    // create polygon and add first point Latitude, Longitude
polygon := cmap.AddPolygone(48.8594069779731,
2.34283447265625) ;
    // better for optimisation
polygon.BeginUpdate;
    // change color
polygon.fillColor := clYellow;
    // change opacity
polygon.fillOpacity := 30;
    // add other points
polygon.Add(48.8638113489879,2.35519409179688);
polygon.Add(48.8608751447095,2.36532211303711);
polygon.Add(48.8543245298006,2.36360549926758);
    // update
polygon.EndUpdate;
    // zoom to polygon
polygon.fitBounds;
```



Fig. 155 TECCesiumPolygone

TECCesiumShapePOI

Contrairement à TECNativeMap seul les types poiEllipse et poiText sont supportés pour l'instant

```
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. 156 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. 157 poiText

Material

Vous pouvez utiliser la propriété material pour par exemple incruster une image dans un polygone.

```
Cesium.Shapes.polygones[index].Material :=  
' "https://cesiumjs.org/images/2015/02-02/cats.jpg" ; '
```

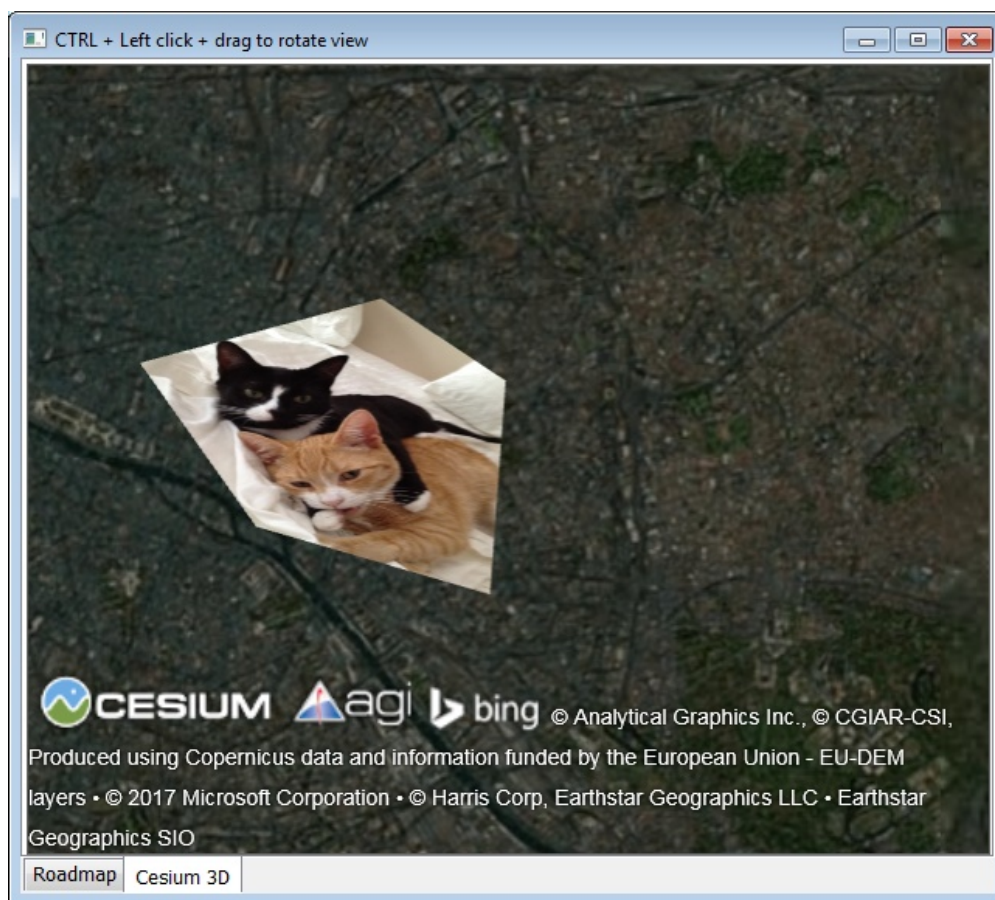


Fig. 158 material property

Vous pouvez directement utiliser du javascript pour définir un material plus complexe.

```
Cesium.Shapes.polygones[index].Material :=
```



```
'new Cesium.ImageMaterialProperty({'+
    'image : "'+UrlImageToDataUri(
    'd:\alert-weather.png')+'",'+
    'transparent:true }')';
```



Fig. 159 material local image

TECCesiumShapeInfoWindow

Contrairement à TECNativeMap la position des InfoWindows est indiquée en coordonnées écran, vous pouvez utiliser du html pour votre contenu et utiliser CSS pour styler vos 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;

```

L'événement **OnCloseInfoWindow** du composant TECCesiumMap est déclenché lorsque l'on ferme par la croix et l'**InfoWindow** qui vient d'être fermée est récupérée.

Si votre fenêtre contient un lien, le click déclenche l'événement **OnUrl**

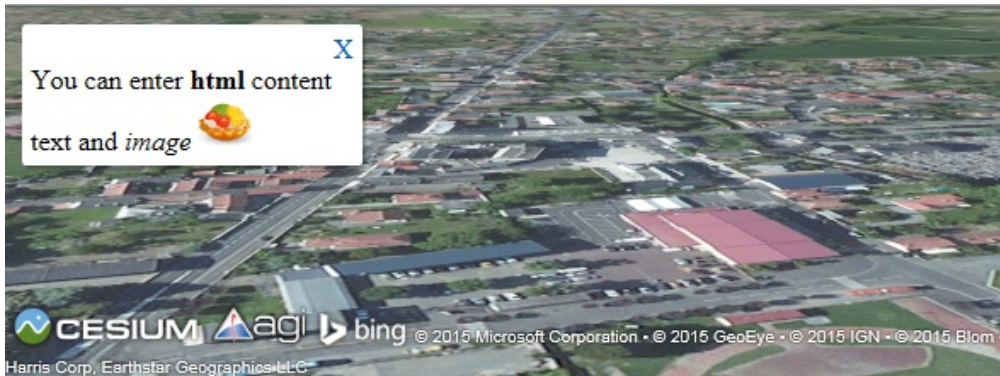


Fig. 160 InfoWindow

TECCesiumShapeModel

Cette classe gère les modèles 3D, Cesium utilise un convertisseur COLLADA -> glTF est proposé

```
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);
```



[Téléchargez les démos !](#)

CZML

CZML est l'équivalent en plus puissant du langage KML utilisé par google earth


```
procedure LoadCZML(const CZML:string;const idCZML:string="";const
Options:string="";const Promise:string="");
```

Pour les options voir

[//cesiumjs.org/Cesium/Build/Documentation/CzmlDataSource.html](http://cesiumjs.org/Cesium/Build/Documentation/CzmlDataSource.html) (Cesium.CzmlDa
options))

Promise permet d'exécuter du javascript quand le script czml est
terminé, LoadAndZoomToCZML l'utilise.

```
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;
```

Lorsque le script est terminé l'évènement OnCZML de votre composant
est déclenché, il retourne 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. 162 create point with czml

TECMapAdressEdit est un composant descendant de TEdit qui dispose d'une assistance automatique pour la saisie des adresses

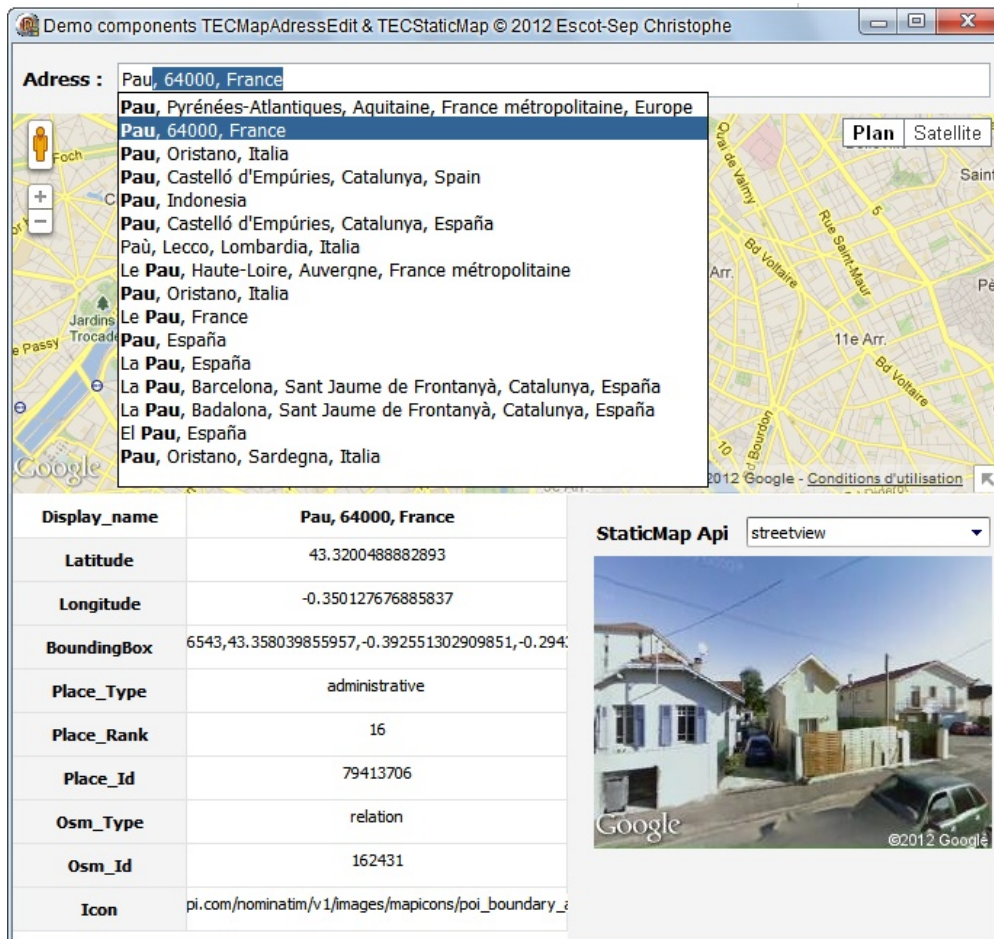


Fig. 163 TECMapAdressEdit en action

Lorsque vous présélectionnez dans la liste une proposition d'adresse l'événement **OnSelect** est déclenché

Lorsque vous validez une adresse, soit en pressant la touche **retour** soit en cliquant sur une proposition avec le bouton gauche de la souris, l'événement **OnAdress** est déclenché

Vous avez accès à la propriété **Nominatim** qui contient des

informations suivante sur l'adresse sélectionnée

- 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**

Les adresses sont obtenues avec le service [Nominatim de MapQuest](#)

TEStaticMap est un descendant de TImage qui va vous permettre d'afficher une carte statique en utilisant les services de [Google Map](#), [Google StreetView](#) et [OpenMapQuest](#)

Api vous permet de sélectionner le service à utiliser (**saGoogle**, **saStreetView** ou **saOpenMapQuest**)

Vous choisissez la région à afficher en renseignant soit la propriété **Adress** soit les propriétés **Latitude** et **Longitude**

Adress *est prioritaire sur* **Latitude** *et* **Longitude**, *assurez vous qu' Adress soit vide si vous souhaitez utiliser des coordonnées.*

MapType vous permet de choisir le type de cartes (**stMap**, **stTerrain**, **stSatellite**, **stHybrid**) (aucun effet si vous choisissez StreetView)

Zoom vous offre le choix du zoom

Params vous permet d'utiliser des paramètres complémentaires (ex **Params:='key=API_console_key&scale=2'**) voir les différentes documentations pour plus d'infos

Pour afficher votre image utilisez la fonction **LoadMap** qui retourne True ou False suivant la réussite ou l'échec

```
// Delphi Static Map component

// api google map
StaticMap.api := saGoogle;
// satellite view
StaticMap.MapType := stSatellite;

StaticMap.Zoom := 14;

StaticMap.Adress := 'Tarbes';
// show map
StaticMap.LoadMap;
```

TECVelib est une classe qui vous donne accès à l'api des [Vélos en livre-service JCDecaux \(Vélib\)](#)

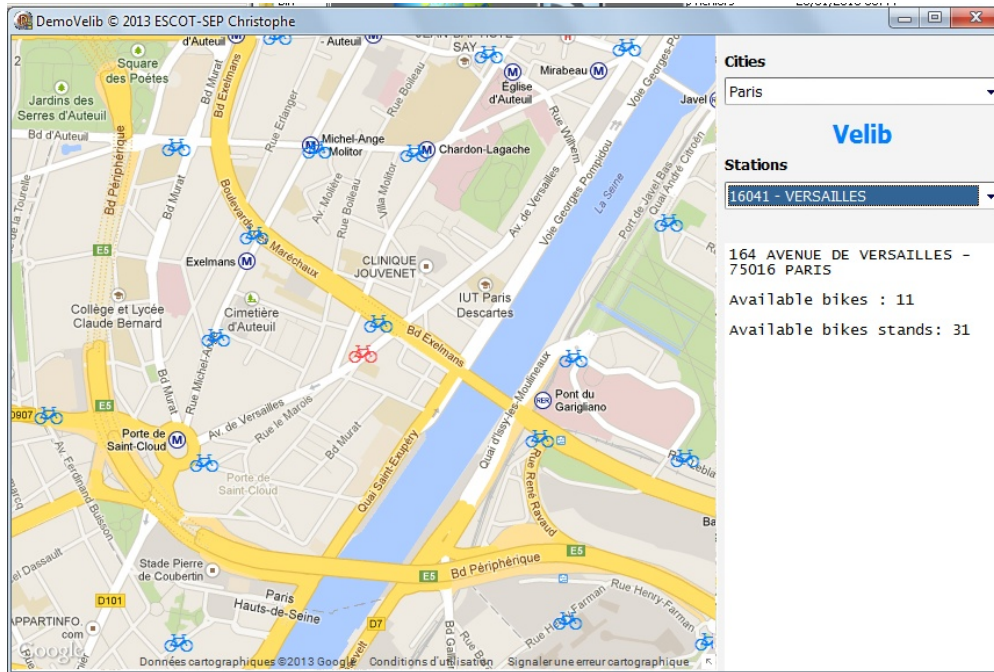


Fig. 164 DemoVelib

TECVelib

function **getContracts** : integer;

Récupère l'ensemble des contrats, ils sont stockés dans le tableau **Contracts**

Retourne le nombre de contrats, accessible aussi au travers de la propriété **ContractsCount**

function **getAllStations**:integer;

Récupère l'ensemble des stations de tous les contrats, accessible dans le tableau **Stations**

Retourne le nombre de stations

function **getContractStations**(const Contract:string):integer;

Récupère l'ensemble des stations pour un contrat, accessible dans le tableau **Stations**
Retourne le nombre de stations

```
function InfoStation(const Contract: string; const Station:  
TECVelibStation): boolean;
```

Obtient les informations d'une station liée à un contrat

```
property ApiKey: string;
```

Une clé d'api est nécessaire, elle est gratuite il vous suffit de [vous enregistrer sur le site](#)
TECVelib possède sa propre clef mais il est préférable que chacun enregistre la sienne.

```
property Stations : TECVelibStations;
```

Liste des stations (voir **getAllStations** et **getContractStations**)

```
property Contracts[index:integer] : TECVelibContractInfo;
```

Liste des contrats (voir **getContracts**)

```
property ContractsCount:integer;
```

Nombre de contrats

Exemple d'utilisation

```
// 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;
```

Dans ce chapitre nous traiterons de cas précis et nous verrons comment utiliser TECMap et ses divers composants avec Delphi

[Création interactive d'une route](#)

Nous allons réaliser une classe qui manipulera un composant TECMap dans le but de créer une route en cliquant sur la carte le point de départ et d'arrivée, une fois la route créé nous devons pouvoir la modifier à la souris

[Lire](#)

[Afficher StreetView dans une InfoWindow](#)

Nous allons voir comment afficher [StreetView](#) dans une [InfoWindow](#)

[Lire](#)

[Afficher un Layer Panoramio dans toutes les cartes](#)

Google Maps dispose déjà en natif d'un [layer Panoramio](#) nous allons voir comment porter cette fonctionnalité vers toutes les apis disponible dans notre **composant Delphi TECMap**.

[Lire](#)

[Afficher une MiniMap](#)

Nous allons voir comment incruster une mini carte synchronisée sur la carte principale

[Lire](#)

Nous allons réaliser une classe qui manipulera un composant TECMap dans le but de créer une route en cliquant sur la carte le point de départ et d'arrivée, une fois la route créé nous devrons pouvoir la modifier à la souris

Un événement sera déclenché à chaque modification de la route et nous pourrions alors consulter les diverses informations de celle-ci (distance, durée, points de passages)

```
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;
```

Lors de la connexion du composant TECMap nous sauvegardons les événements sur lesquels nous nous branchons.

```
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;

```

Le principal événement est **OnMapClick** qui nous permet de réagir lors que l'utilisateur clique sur la carte.

Si la route n'est pas créé on teste si l'on doit créer le marker de début ou de fin, pour chaqu'un d'eux nous ajustons ses coordonnées avec **AlignLatLngToRoute** pour se positionner sur la route la plus proche du point cliqué.

```

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;
```

On se branche sur l'événement **OnMarkerMove** des markers pour pouvoir modifier manuellement le début et la fin de notre route dans le cas des apis qui ne permettent pas de gérer les routes modifiables.

```

procedure TInfoRoute.doOnMoveMarker(sender: TObject; const
  Index: integer;
                                var dLatitude,
  dLongitude: double);
begin
    FECMap.AlignLatLngToRoute(dLatitude, dLongitude);
    setRoute;
end;

```

Lors de la création du point d'arrivée on appelle **setRoute** pour créer notre route.

Seules les apis **Google Maps** et **OpenMapQuest** permettent de modifier la route à la souris, pour les autres nous allons émuler cette fonction en utilisant une **Polyline** et nos markers de début et de fin.

```

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;

```

Avec **Google Maps** ou **OpenMapQuest** lorsque la route est modifiée l'événement **OnRouteChange** est déclenché, si la route vient d'être créé nous la rendons modifiable et nous ajustons sa couleur.

```

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
      end
    end
  end;

```

Lorsque la route est créé/modifiée l'événement **OnRoute** de notre classe est déclenché, en vous y branchant vous pourrez récupérer les informations sur celle-ci au travers de la propriété **RoutePath** de type

TECMapRoutePath

```
RoutePath.Distance;    // distance in meters
RoutePath.Duration;    // duration in seconds
```

Ci-dessous l'intégralité de l'unité, vous pouvez remarquer la gestion des événements **OnBeforeReLoad** et **OnAfterReLoad** pour tenir compte d'un éventuel rechargement de la carte par exemple lors d'un changement d'api.

```
unit UInfoRoute;

interface
uses Windows, SysUtils, Classes, Graphics, ECMaps;

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;
```

```
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;

```

```
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.
```

Nous allons voir comment afficher **StreetView** dans une **InfoWindow**

StreetView n'est disponible qu'avec **l'api Google Maps !**

La première étape est de créer notre **InfoWindow**, nous stockons son index dans `idStreetWindow`

Son contenu sera un élément **DIV** qui devra avoir un **nom unique** ici "streetwindow"

Il faut aussi définir sa taille, 300 pixels en hauteur et largeur.

```
idStreetWindow := map.InfoWindows.Add( '<div  
id="streetwindow" style="width:300px;height:300px"></div>'  
> );
```

Nous nous branchons sur les événements **OnInfoWindowOpen** et **OnInfoWindowClose** pour afficher/cacher **StreetView** lorsque l'**InfoWindow** sera ouverte ou fermée.

```
map.OnInfoWindowOpen := mapInfoWindowOpen;  
map.OnInfoWindowClose := mapInfoWindowClose;
```

Dans ces événements nous utilisons javascript pour faire le travail désiré.

```
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;

```

Pour ouvrir l'InfoWindow vous pouvez soit réagir directement lors d'un clic sur la carte en vous branchant sur OnMapClick

```

procedure TForm.mapMapClick(sender: TObject;const dLatitude,
dLongitude: Double);
begin

    map.InfoWindows[idStreetWindow].SetPosition(dlatitude,dlongitude);
    map.InfoWindows[idStreetWindow].Open := true;
end;

```

Mais vous pouvez aussi l'associer à un marker et alors elle s'ouvrira lors d'un clic sur celui-ci

```

    // anchor infowindow to marker number 0

    map.InfoWindows[idStreetView].Anchor := 0;

```



Fig. 165 StreetView in InfoWindow

Google Maps dispose déjà en natif d'un [layer Panoramio](#) nous allons voir comment porter cette fonctionnalité vers toutes les apis disponible dans notre **composant Delphi TECMap**.

Nous allons créer une classe sur le même modèle que celle qui gère la [création interactive d'une route](#) c'est pourquoi nous ne détaillerons pas le code commun.

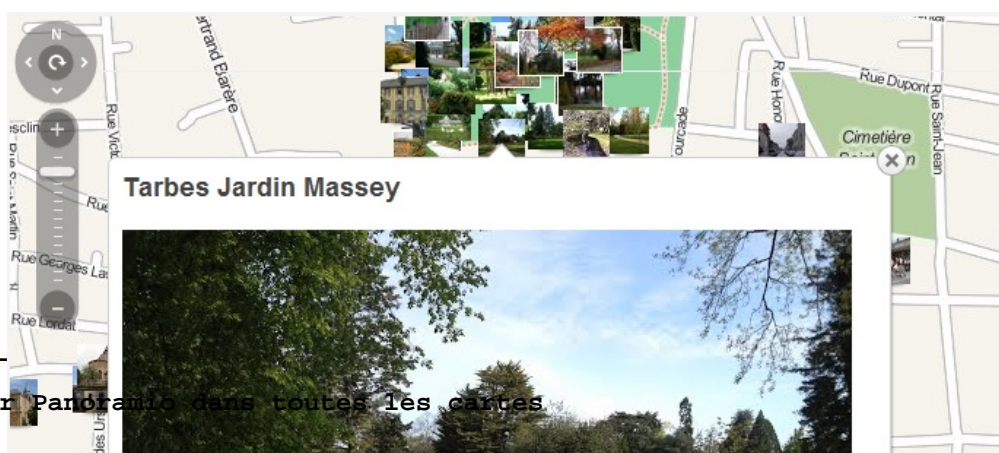
Fonctionnement

Le principe est simple, chaque fois que la vue de la carte change nous récupérons les images panoramio disponible dans la zone visible, avant de les afficher on les place dans un [groupe](#) pour pouvoir les manipuler plus facilement.

On se branche sur l'évènement **OnChangeMapBounds** pour détecter le changement de vue.

Pour la recherche d'images on utilise [PanoramioView.Search](#), elle est lancée dans un Thread séparé, nous devons nous brancher sur l'évènement **OnPanoramioSearch** pour récupérer notre liste d'images.

On va transformer chaque image en [marker](#) et gérer le Click pour pouvoir afficher une [InfoWindow](#) contenant le titre et l'image en taille moyenne.



Utilisation de la classe TPanoramioLayer

```
FPanoramioLayer := TPanoramioLayer.create;
...
// connect TECMap component for show Panoramio layer
FPanoramioLayer.Map := map;
// disconnect layer
FPanoramioLayer.Map := nil
```

Source de 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.

Nous allons voir comment incruster une mini carte synchronisée sur la carte principale

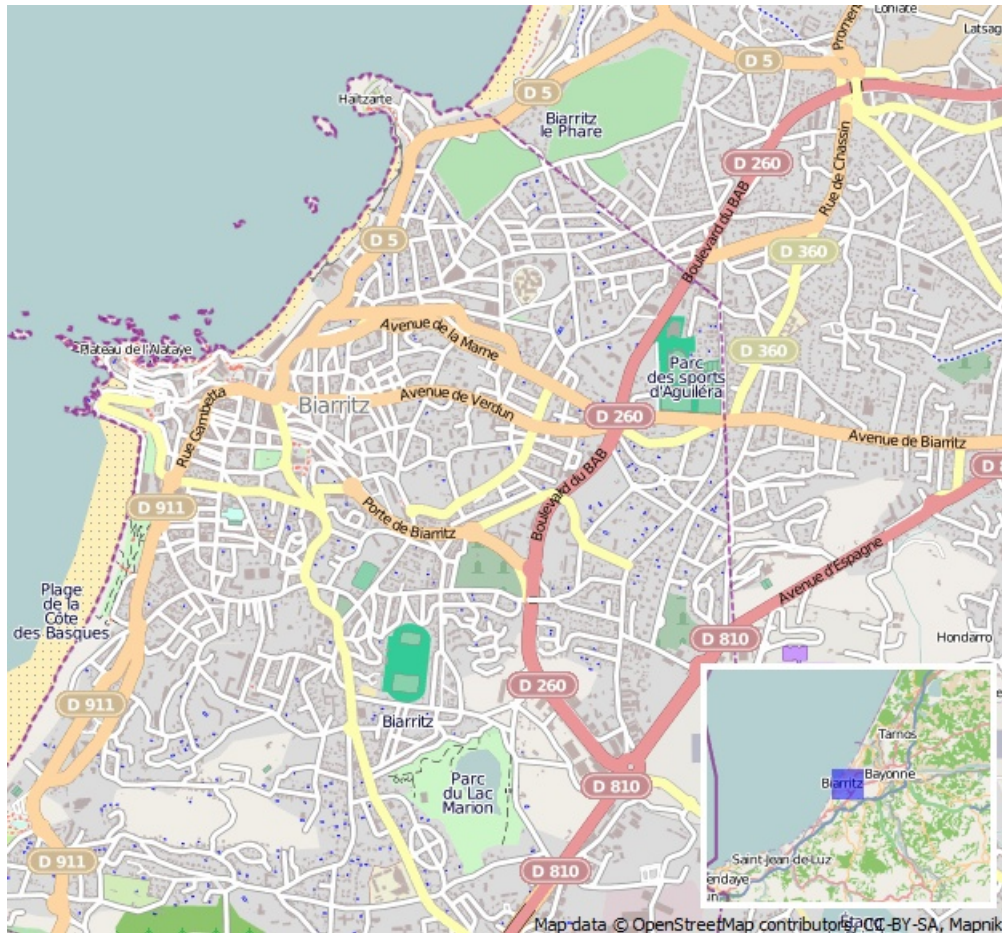


Fig. 167 Affichage d'une minimap sur un composant TECNativeMap

Design

Placez un **TPanel** sur votre composant **TECNativeMap**, fixez les propriétés **color** à `clWhite`, **BevelInner** et **BevelOuter** à `false` et **BorderWidth** à `4`

Déposez sur le TPanel un composant **TECNativeMap** avec

ses propriétés **Align** à **alClient**, **dblClickZoom** et **Draggable** à **false**

Code

La mini carte devra être centrée sur la carte principale avec un zoom 5 crans en dessous, et nous y dessinerons une zone rectangulaire correspondante à la vue de la carte principale.

```

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.Polygones.add(0,0) ;
        FMinipolygone := minimap.shapes.Polygones[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;

```

Vous devrez appeler **UpdateMiniMap** dans les événements **OnMapMove** et **OnChangeMapBounds** de la carte principale.

Vous devrez aussi répondre aux événements **OnMapMove** et **OnChangeMapBounds** pour ajuster la vue de la carte principale lorsque l'on actionne directement la mini carte, sans oublier d'ajuster la position de la minimap dans l'événement **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);
    end;

```

```

    private
        { Déclarations privées }
    }
    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.
```

Fig. 1 Deux composants, 5 Apis, 3 Moteurs d'affichage !

Fig. 2 Cesium

EF(<http://www.helpandweb.com/demos-ecmap.zip>):GRATUIT -

Téléchargez les démos !

Fig. 4 Livraison d'un projet utilisant chromium

Fig. 5 DemoLocalise

Fig. 6 DemoRoute

Fig. 7 DemoMobile

Fig. 8 Demo3D

Fig. 9 DemoOverlays

Fig. 10 DemoLayer

Fig. 11 Démonstration de l'utilisation de DistanceMatrix

Fig. 12 DemoAdressEdit

Fig. 13 Firemonkey Demo

Fig. 14 Url Link

Fig. 15 OpenMapQuest

Fig. 16 Carte CloudMade

Fig. 17 Carte OpenMapStreet Mapnik

Fig. 18 Carte OpenMapStreet A render

Fig. 19 OpenMapStreet CycleMap

Fig. 20 OpenMapStreet MapQuest

Fig. 21 Google : Type Roadmap

Fig. 22 Google : Type Hybrid

Fig. 23 Google : Type TERRAIN

Fig. 24 Obtenir les styles au format JSON

Fig. 25 Styles au format JSON

Fig. 26 POI

Fig. 27 Only poi-park

Fig. 28 Editeur de styles CloudMade

Fig. 29 Contrôles des cartes Google

Fig. 30 Utilisation d'un TComboBox pour gérer la geolocalisation

- Fig. 31 Place AutoComplete
- Fig. 32 DemoLocalise
- Fig. 33 Polygone en mode édition
- Fig. 34 DemoOverlay
- Fig. 35 Activation du regroupement de markers
- Fig. 36 Marker mobile le long d'une route
- Fig. 37 Une infowindow associée à un cercle
- Fig. 38 DemoRoute - Instructions
- Fig. 39 DemoRoute - Altitudes
- Fig. 40 BicyclingLayer
- Fig. 41 OpenMapStreet CycleMap
- Fig. 42 TrafficLayer
- Fig. 43 WeatherLayer
- Fig. 44 DemoLayer
- Fig. 45 DemoPOI
- Fig. 46 Vue StreetView
- Fig. 47 Une vue StreetView connectée à une vue carte
- Fig. 48 EarthView dans Demo3D
- Fig. 49 DemoOverlay
- Fig. 50 Importation des overlays au format KML
- Fig. 51 Activation de la vue Panoramio
- Fig. 52 Affichage d'une InfoWindows avec les informations d'une image Panoramio
- Fig. 53 DemoLayer
- Fig. 54 DemoMatrix démonstration de l'utilisation de DistanceMatrix
- Fig. 55 ECNativeMap on Mac OS X
- Fig. 56 Bing Maps
- Fig. 57 Here Traffic
- Fig. 58 MapBoxStreets
- Fig. 59 Incrustation des nuages
- Fig. 60 ZoomScaleFactor

- Fig. 61 ScaleMarkerToZoom
- Fig. 62 sélection stylisée
- Fig. 63 3 selected items
- Fig. 64 Rotation map
- Fig. 65 Url Link
- Fig. 66 MiniMap
- Fig. 67 Scale bar
- Fig. 68 Outil de mesure
- Fig. 69 My local tiles
- Fig. 70 Archive ile de ré
- Fig. 71 MapArchiveCreator
- Fig. 72 show vector tiles
- Fig. 73 green < 5000 - blue > 6000
- Fig. 74 Demo OSMViewer
- Fig. 75 Vectors tiles
- Fig. 76 A gauche la carte principale - à droite une vue secondaire
- Fig. 77 Server
- Fig. 78 Client
- Fig. 79 DemoLocalise
- Fig. 80 Demo Geofences
- Fig. 81 DemoNativeBoundaryArea
- Fig. 82 OverPassApi query highway
- Fig. 83 OverPassApi.Data
- Fig. 84 Demo OSMViewer
- Fig. 85 Select nodes
- Fig. 86 Sélection d'une zone
- Fig. 87 search highway=residential
- Fig. 88 Android Test OLT
- Fig. 89 Photographer
- Fig. 90 HERE Hybrid Hi-Res 512
- Fig. 91 OSM Hi-Res 512

- Fig. 92 OSM Hi-Res 256
- Fig. 93 Clustering
- Fig. 94 TECShapes.Show
- Fig. 95 Trackline
- Fig. 96 SnapDrag
- Fig. 97 Loading images
- Fig. 98 StyleIcon si3d - siFlat - siFlatNoBorder - siDirection
- Fig. 99 Maki icons
- Fig. 100 Champ de vision
- Fig. 101 automatic scale
- Fig. 102 ScaleMarkerToZoom
- Fig. 103 OnBeforeDraw
- Fig. 104 GroundOverlay
- Fig. 105 PoiShape
- Fig. 106 TECShapePOI en mode édition - poignée de redimensionnement affichée
- Fig. 107 Une ligne en mode édition
- Fig. 108 ShowDirection
- Fig. 109 slice - red color between 1.2 km and 3.8 kilometer
- Fig. 110 Bordure noire de 2 pixels
- Fig. 111 Penstyle := psDashDot - LineType := ltBezier
- Fig. 112 Ajout de drapeaux pour le départ et l'arrivée
- Fig. 113 Edit line
- Fig. 114 Modification de l'apparence des points d'une TECShapeLine
- Fig. 115 Sélection d'une partie d'une ligne ou d'un polygone
- Fig. 116 geodesic line
- Fig. 117 Weather on the road
- Fig. 118 Polygones
- Fig. 119 Level = 16 - Level = 0
- Fig. 120 InfoWindow
- Fig. 121 html Content InfoWindow

- Fig. 122 TECAAnimationShapeColor
- Fig. 123 TECAAnimationFadePoi
- Fig. 124 TECAAnimationDrawPath
- Fig. 125 TECAAnimationMovOnPath
- Fig. 126 TECAAnimationMoveToDirection
- Fig. 127 DemoNativeRoute
- Fig. 128 DemoNativeRoute
- Fig. 129 Click sur un élément Panoramio
- Fig. 130 Click sur un élément TECNativePlaceLayer
- Fig. 131 'restaurant|bar|cafe'
- Fig. 132 'highway=bus_stop'
- Fig. 133 Xapi Way
- Fig. 134 Xapi Roads & Junction
- Fig. 135 UTFGrid geography-class
- Fig. 136 Using labels to display positions
- Fig. 137 Demo Mappilary
- Fig. 138 Demo Firemonkey MappilarySearchImages
- Fig. 139 Heatmap layer
- Fig. 140 Style d'affiche des points
- Fig. 141 Demo ECNativeMap for Android
- Fig. 142 Markers
- Fig. 143 Goto Paris
- Fig. 144 Tracking
- Fig. 145 Select Layers
- Fig. 146 Save your Map
- Fig. 147 Android Test OLT
- Fig. 148 ECNativeMap on iOS simulator
- Fig. 149 TECCesiumMap
- Fig. 150 Marker by default
- Fig. 151 Marker with Text
- Fig. 152 Marker with Maki icon

Fig. 153 Marker with Png

Fig. 154 TECCesiumShapeLine

Fig. 155 TECCesiumPolygone

Fig. 156 TECCesiumShapePOI

Fig. 157 poiText

Fig. 158 material property

Fig. 159 material local image

Fig. 160 InfoWindow

EF(<http://www.helpandweb.com/demos-ecmap.zip>):Téléchargez les démos !

Fig. 162 create point with czml

Fig. 163 TECMapAdressEdit en action

Fig. 164 DemoVelib

Fig. 165 StreetView in InfoWindow

Fig. 166 Layer Panoramio sous MapQuest

Fig. 167 Affichage d'une minimap sur un composant TECNativeMap

INDEX

*

45° 34

A

Aborted 66
 Abort 66
 addBounds 142
 AddByAdr 111
 AddGeodesicLine 299
 add 142
 AddInfoWindow 262
 Add 125
 AddLine 262
 AddMarker 87
 Add 87
 AddMarker 262
 add 97
 Add 72
 Add 58
 AddPOI 262
 AddPolygone 262
 AddPolylineFromRoute 111
 AddRouteByAddress 111
 Add 111
 AddToGroup 72
 AddView 217
 AdminLevelFromZoom 50
 AdrControl 147
 AdrCss 147
 Adresses 50

Adress	50
Adress	87
Adress	58
Adress	147
Adress	399
AdrPosition	147
Affichage d'une MiniMap sur un composant TECNativeMap	431
Afficher un Layer Panoramio dans Bing maps	423
aimant	262
AlertDistance	111
AlignLatLngToRoute	405
Altitude	1
Altitude	87
AltitudeMode	1
Altitude	27
Altitude	111
amAbsolute	1
amClampToGround	1
amClampToSea	1
amRelativeToGround	1
amRelativeToSea	1
Anchor	103
ancragePosition	178
Android	370
android	178
angle	262
Animation	323
Animation	87
apiCloudMade	34
apiGoogle	34
ApiKey	401
apiLeaflet	34
apiOpenMapQuest	34
ApiVersion	1
ApiVersion	34
ArcGis	50
archive	201
area	201
Area	72
Atmosphere	1
Author	168
AutoComplete	58
Available	147

AvoidHighWays	174
avoidHighways	111
AvoidTolls	174
avoidTolls	111

B

beginupdate	142
BeginUpdate	72
bézier	299
bicyclinglayer	125
bicycling	125
Bing Maps	423
bing maps	404
bing	1
BingKey	7
BingKey	178
bing	7
BorderColor	178
BorderColor	299
BorderSize	178
BorderSize	299
BOUNCE	87
Boundary	50
Bounds	66
bounds	27
BringToFront	262
BrushFilename	315
ByLatLng	87
ByName	87

C

cache	201
cache	219
Camera	1
CanceledSearchImageClose	125
CanceledSearchSequence	125
CancelSearchImageClose	125
CancelSearchSequence	125

cartes	1
cartographie	1
CEF1	10
CEF3	10
Cells	174
CenterOnMap	262
Center	72
cercle	72
chromium	1
Chromium	10
Circles	72
Clause	125
Clear	1
clear	142
ClearGuide	262
Clear	66
Clear	125
Clear	87
ClearMarker	262
clear	97
Clear	72
Clear	58
Clear	111
Clickable	87
Clickable	168
ClientId	125
CloseBtnVisible	147
CloseButton	318
CloudMAde et Google Maps	423
cloudmade	4
CloudMade	423
cloudmade	1
CloudMadeKey	7
CloudMade	7
CloudMade	178
Clusterable	262
ClusterManager	262
Cluster	87
Cluster	262
Color	111
color	206
Column	125
composant	4

Connexion	125
connexion	201
Contains	142
ContainsLatLng	27
ContentCenter	318
Content	103
Content	108
ContractsCount	401
Contracts	401
CopyImage	87
Copyright	111
CountDetail	58
Count	174
CountKey	66
Count	125
Count	87
count	97
Count	72
Count	58
CountResult	58
Count	111
cpBottomCenter	47
cpBottomLeft	47
cpBottomRight	47
cpLeftBottom	47
cpLeftCenter	47
cpLeftTop	47
cpRightBottom	47
cpRightCenter	47
cpRightTop	47
cpTopCenter	47
cpTopLeft	47
cpTopRight	47
css	108
CurrentTimeTour	1
Cursor	87
Cursors	87
cyclemap	125
CZML	377

D

dbChromium	10
dbIE	10
Delete	1
delete	142
Delete	125
Delete	87
delete	97
Delete	72
Delete	58
Delete	111
delphi	4
Delphi	423
Delphi	1
delphi	58
Delphi	404
Demo3D	10
DemoLayer	125
DemoLocalise	10
DemoMatrix	10
DemoMobile	10
DemoOverlays	10
DemoRoute	10
Destination	174
Destinations	174
Direction	97
DisplayBrowser	10
Distance	174
DistanceFrom	27
DistanceFrom	111
DistanceMatrix	174
distancematrix	10
Distance	97
Distance	72
DistanceRouteByAdress	111
DistanceRouteFrom	111
Distance	111
DistanceText	174
DistanceTo	262
download	201
Draggable	108
Draggable	87
Draggable	136
Draggable	111

Dragging	87
DragRect	178
drag	262
DragZoom	178
DROP	87
drZoom	178
Duration	174
Duration	1
Duration	111
DurationText	174

E

échelle	47
ecmap	4
editable	294
EditMode	72
EditOnClick	111
EditOverlay	72
EditRoute	111
EnableLayer	1
EnableTouchRotation	178
EnableUIControl	47
Encoded	299
EncodePrecision	299
EndAdress	111
EndLatitude	111
endLongitude	111
EndMarkerFilename	111
endupdate	142
EndUpdate	72
EngineExecute	111
EngineKey	111
EngineName	111
Engine	111
EngineUrl	111
EngineValidUrl	111
ErrorDistance	111
ExcludelfKeyExist	66
ExcludeKeyValue	66
ExecutionDistance	111

ExitTour	1
----------------	---

F

FileFormat	66
FilenameEndEditLine	299
FilenamePointEditLine	299
FilenameStartEditLine	299
fillColor	72
FillOpacity	72
FilterBounds	66
FilterNode	66
FilterPrimitive	66
FilterRelation	66
FilterWay	66
FindImageClose	125
Find	66
FindShapeByArea	262
FindShapeByFilter	262
FindShapeByKMDistance	262
FindToShapes	66
firemonkey	1
fitBounds	142
fitBounds	87
fitBounds	27
fitBounds	111
fitBounds	262
Flat	87
FlyToSpeed	1
FontSize	206
FovRadius	285
Fov	285
FromXYToLatLngAlt	1
FusionTablesLayer	125
fusiontables	125

G

geofence	50
geohash	50

GeoJSON	142
geoJSON	66
geolocalisation	10
GeoLocationFromLatLng	50
GeoLocation	50
GeoXml	125
GeoXml	72
GetAdressFromLatLng	50
getAdress	72
getAllStations	401
GetAltitudeAtLatLng	27
getAltitudes	72
GetAsyncRoutePathByAdress	111
GetASyncRoutePathFrom	111
getContractStations	401
getContracts	401
getDetails	58
getGroundAltitude	1
getKey	66
getLatLngFromMeter	111
GetPolygoneFromID	50
GetRoutePathByAdress	111
GetRoutePathFrom	111
google maps	4
google maps	420
Google Maps	423
google maps	1
google maps	404
Google	7
GoogleMapsKey	7
gpx	142
GPX	66
gradient	206
Grid	1
GroundOverlays	72
GroundOverlay	285
Groupe	423
Groupes	142
Group	72
Groups	142

H

HasMore	168
Haute résolution	258
haversine	111
Heading	1
HeadingFrom	27
Heading	87
Heading	97
Heading	111
Heading	147
heatmap	125
heatmap	125
Height	72
Here	178
hide	142
hint	262
hors-ligne	201
HoverBorderColor	299
HtmlCopyright	168
htmlInfoWindow	125
html	108
HtmlPhoto	168

I

IconAnchor	87
Icon	87
IconOrigin	87
IconSize	87
Id	108
Id	125
Images	125
Import/Export	66
Index	103
Index	87
Index	97
IndexOf	1
IndexOfLatLng	111
IndexOf	87
IndexOf	97

IndexOf	58
InfoStation	401
infowindow	420
infowindow	423
infowindow	103
InfoWindow	10
InfoWindow	87
InfoWindow	72
infowindow	262
InfoWindows	103
InfoWindow	318
Insert	72
instructions	111
iOS	375
ios	178
IsSearch	168
isTrackLineActive	262
ItemDetail	58
Item	174

J

junction	125
----------------	-----

K

Keys	66
kml	142
KML	66
kml	125
kml	72
Kmz	125
kmz	72

L

Label	108
LabelLayer	125

Label	125
Labels	72
Latitude	1
Latitude	103
Latitude	87
Latitude	58
latitude	27
Latitude	147
Latitude	399
layer	125
leaflet	4
Leaflet	423
leaflet	1
leaflet	7
leaflet	404
Leaflet	34
licence	4
lien	318
Lieu	58
ligne	72
LinkVisible	147
LoadAndZoomToCZML	377
LoadCZML	377
Loaded	1
LoadFromFile	66
LoadFromOSMStream	66
LoadFromOSMStream	262
LoadFromOSMString	66
LoadFromOSMString	262
LoadImage	125
LoadKml	1
localarchive	201
LocalCache	125
LocalCache	50
localcache	201
LocalCache	178
Localisation	50
LocalPathImage	125
LocationType	50
Longitude	1
Longitude	103
Longitude	87
Longitude	58

longitude	27
Longitude	147
Longitude	399
LookAt	1
ltBezier	299

M

mac osx	1
mac osx	178
maki	285
map	4
MapAPI	34
maparchive	201
MapBoxToken	178
map	1
mappilay	125
MapQuest	423
MapQuest	7
mapquest	404
MapQuest	178
maps	404
MapTypeControl	47
MapTypeControlPosition	47
MapTypeID	34
MapType	399
marker	423
Marker	97
MarkerOptions	111
Markers	87
Markers	72
MaxDayInCache	125
MaxDayInCache	178
MaxWidth	103
MaxZoom	142
MaxZoom	27
mcLocalOrServer	125
mcOnlyLocal	125
mcOnlyMappilaryServer	125
Menu	47
meteo	299

MeterDistance	262
MinDistanceMeterForNotifyMove	262
MinZoom	142
MinZoom	27
Mobile	97
MobilesEnabled	97
Mobiles	97
MobilesTiming	97
mode hors-ligne	178
Model	1
Models	1
moteur de routage	111
MouseAltitude	1
MouseLatitude	27
MouseLatLng	178
MouseLongitude	27
MouseNavigationEnabled	1
MoveMarkerOnRouteToMeter	87
Move	97
mtHYBRID	34
mtROADMAP	34
mtSATELLITE	34
mtTERRAIN	34
multiview	217

N

NameDetail	58
Name	87
NameResult	58
NavigationControl	47
navigation	47
NavigationControl	147
NavigationControlVisibility	1
NavigationPosition	47
NavigationPosition	147
navigation	111
NavigationStyle	147
ncvAuto	1
ncvHide	1
ncvShow	1

NextSearch	168
node	125
Nodes	66
Nominatim	397
NorthEastLatitude	27
NorthEastLatitude	111
NorthEastLongitude	27
NorthEastLongitude	111

O

observateur	262
observateur	178
OnAddRoute	111
OnAddShapeToView	217
OnAdress	397
OnAfterChangeMapApi	34
OnAfterChangeToTxt	66
OnAfterInstruction	111
OnAfterReLoad	405
OnAfterReload	34
OnAlert	111
OnAnimation	323
OnArrival	111
OnBeforeChangeMapApi	34
OnBeforeChangeToTxt	66
onbeforedraw	285
OnBeforeLoad	34
OnBeforeReLoad	405
OnBeforeReload	34
OnBeforeUrl	178
OnBeforeUrl	318
OnChangeMapBounds	423
OnChangeMapBounds	27
OnChangeMapTypeld	34
OnChangeMapZoom	27
OnChangeRoute	111
OnClick	125
OnCloseInfoWindow	103
OnCloseInfoWindow	318
OnConnectRoute	111

OnCreateShapeLinePoint	299
OnDeconnectRoute	111
OnDrawRoute	111
OnEarthViewChange	1
OnEarthViewClickPlacemark	1
OnEarthViewFrameEnd	1
OnEarthViewInit	1
OnEarthViewShow	1
OnEarthViewUpdateModels	1
OnEarthViewUpdatePlacemarks	1
OnEditOverlayPointClick	72
OnEditOverlayPointDragEnd	72
OnEditOverlayPointRClick	72
OnEndAnimation	323
OnEndMobile	97
OnEnterGeofence	50
OnErrorRoute	111
OnError	111
OnGeoLocation	50
OnImages	125
oninfowindowclose	420
OnInfoWindowOpen	420
OnInstruction	111
OnJavascriptError	50
OnKmlLayerClick	125
OnKmlLayerClick	72
OnLeaveGeofence	50
OnLink	108
OnLoaded	66
OnLoadOverlay	66
onLoad	262
OnLoadShapes	66
OnLoad	34
OnlyIfKeyExist	66
OnlyKeyValue	66
OnlyLocal	178
OnMapClick	405
OnMapDragEnd	27
OnMapDrag	27
OnMapDragStart	27
OnMapMouseMove	27
OnMapMove	87
OnMapMove	27

OnMapRightClick	47
OnMarkerAddress	87
OnMarkerClick	87
OnMarkerDbClick	87
OnMarkerDragEnd	87
OnMarkerDrag	87
OnMarkerDragStart	87
OnMarkerMove	405
OnMarkerMove	87
OnMarkerMove	97
OnMarkerRightClick	87
OnMouseOutCluster	262
OnMouseOverCluster	262
OnOverlayChange	72
OnOverlayClick	72
OnOverlayDbClick	72
OnOverlayMouseDown	87
OnOverlayMouseDown	72
OnOverlayMouseOut	72
OnOverlayMouseOver	72
OnOverlayMouseUp	87
OnOverlayMouseUp	72
OnOverlayMove	108
OnOverlayMove	72
OnOverlayPathChange	72
OnOverlayRightClick	72
OnPanoramioLayerClick	168
OnPanoramioSearch	168
OnPlacesDetail	58
OnPlacesSearch	58
OnRead	66
OnRouteChange	405
OnRouteChange	111
OnRouteError	111
OnRoutePath	111
OnSelect	397
OnSelect	299
OnSequenceColor	125
OnSequences	125
OnShapeClick	262
OnShapeDbClick	262
OnShapeDragEnd	262
OnShapeDrag	262

OnShapeMouseDown	262
OnShapeMouseOut	262
OnShapeMouseOver	262
OnShapeMouseup	262
OnShapeMove	262
OnShapeRightClick	262
OnSnap	262
OnStreetViewAvailable	147
OnStreetViewPosition	147
OnStreetViewPOV	147
OnStreetViewVisible	147
Opacity	111
opeanmapquest	7
OpenClycleMap	178
Open	103
openmapquest	4
openmapquest	1
OpenMapQuest	34
OpenStreetMap	58
OpenStreetMap	178
OPNV	178
OptimizeWaypoints	111
Origin	174
Origins	174
OSM XML	66
OSMFileToOLT	66
OSMViewer	206
ovCircle	72
OVER_QUERY_LIMIT	50
overlay	72
overlays	66
overlay	178
OverlayType	72
OverPassUrl	50
OverSizeForRotation	178
OverviewMap	1
ovGlobe	72
ovGroundOverlay	72
ovKml	72
ovLabel	72
ovLine	97
ovLine	72
ovMap	72

ovMarker	72
ovPolygone	97
ovPolygone	72
ovRectangle	72
ovRoute	97
ovRoute	72
owner_id	168
owner_name	168

P

Panoramio	423
PanoramioLayer	125
Panoramio	125
panoramio	404
Panoramio	34
PanoramioView	168
PanToBounds	142
PanToBounds	72
PanToBounds	27
PanTo	87
PanTo	27
Params	399
Path	72
PauseTour	1
photo_date	168
photo_file_url	168
photo_id	168
photo_lat	168
photo_lng	168
photo_title	168
PhotoID	168
pimMedium	168
pimMini_square	168
pimOriginal	168
pimSmall	168
pimSquare	168
pimThumbnail	168
Pitch	147
place	50
PlacemarkCamera	1

PlacemarkCount	1
PlaceMark	1
PlacemarkLatLngAlt	1
PlacemarkLookAt	1
Places	7
places	58
PlayTour	1
PluginVersion	1
poiEllipse	136
Point	72
poiOwnerDraw	136
poiRectangle	136
Pois	136
poiStar	136
poiTriangle	136
POIUnit	294
polygone	72
Polygones	72
Polyline	405
Polylines	72
Position	111
propertyValue	262

R

Radius	72
Radius	58
Range	1
RawDetail	58
RawResult	58
ReadKey	66
ReadValuesForKey	66
Ready	1
Ready	111
rectangle	72
Rectangles	72
ReDraw	147
Relations	66
ReleaseAllView	217
ReleaseView	217
ReLoad	66

Reload	34
RemoveAllOverlayTiles	178
remove	142
RemoveGuide	262
RemoveMarker	262
Request	111
ResetTour	1
Result	58
Results	58
Reverse	299
Roll	1
route	405
RouteDrawingTime	111
Route	97
route	404
route	111
routes	10
Routes	72
Routes	111
RouteType	97
routeType	111

S

SaveToFile	66
saveToFile	66
SaveToKMLFile	66
ScaleControl	47
ScaleFactor	258
ScaleLegend	1
ScaleMarkerToZoom	285
ScalePosition	47
Scale	285
Scale	206
ScreenShot	66
ScreenShot	27
ScreenShots	66
SearchAt	168
SearchImageClose	125
Search	66
Searching	58

search	50
Search	58
Selected	178
SelectionFrom	299
SelectionTo	299
SendToBack	262
SensStartEnd	111
Sequences	125
serveur	219
setCenter	27
SetHitBox	285
setImage	87
setImageShadow	87
setLatLngAlt	1
setPosition	103
setPosition	108
setPosition	87
SetPosition	147
setShape	87
Shadow	87
Shape	87
Shapes	262
show	142
ShowOnMap	87
Show	262
Slice	299
SnapDistance	262
snapdrag	262
SnapGuide	262
snap	262
SnapShape	262
SouthWestLatitude	27
SouthWestLatitude	111
SouthWestLongitude	27
SouthWestLongitude	111
Speed	97
SQL	125
SSL	34
StartAdress	111
StartLatitude	111
StartLongitude	111
StartMarkerFilename	111
Stations	401

StatusBar	1
Status	174
Status	58
Step	97
StopLoadOverlay	66
StreamPercent	1
streetview	420
StreetView	147
StreetView	34
styleicon	285
Style	125
style	262
styles	262
styles	34
style	206
style	34
Sun	1
SuppressInfoWindows	168
svg	285
symbol	87

T

Tag	87
Tag	168
TECAAnimationAutoHide	323
TECAAnimationDrawPath	323
TECAAnimationFadePoi	323
TECAAnimationMarkerFilename	323
TECAAnimationMarkerZoomFilename	323
TECAAnimationShapeColor	323
TECCesiumMap	1
TECCesiumMap	377
TECCesiumShapeInfoWindow	377
TECCesiumShapeLine	377
TECCesiumShapeMarker	377
TECCesiumShapeModel	377
TECCesiumShapePOI	377
TECCesiumShapePolygone	377
TECDistanceMatrixItem	174
TECDownLoadTiles	201

TECDownloadTiles	178
TECesiumMap	377
TECMapAdressEdit	397
TECMapEarthModel	1
TECMapEarthModels	1
TECMap	1
TECMapFusionTablesLayer	125
TECMapInfoWindow	103
TECMapInfoWindows	103
TECMapItemGroup	72
TECMapItem	72
TECMapMarker	87
TECMapMarkers	87
TECMapOverlay	72
TECMapPanoramioLayer	168
TECMappilaryLayer	125
TECMapPlaces	58
TECMapPois	136
TECMapRoutePath	405
TECMapRoutePathPoint	111
TECMapRoute	111
TECMapRoutes	111
TECMapStreetView	147
TECNativeMap	431
TECNativeMap	1
TECPlaceResults	58
TECShapeInfoWindow	318
TECShapeLine	299
TECShapeMarker	285
TECShapePoi	294
TECShapePolygone	315
TECShape	262
TECStaticMap	399
TECVelib	401
TerrainExaggeration	1
TextSearch	58
TGeoCoderReponses	50
Thumb1024	125
Thumb2048	125
Thumb320	125
Thumb640	125
Thumb	125
TileServer	178

tiles	201
Tilt	1
tilt	34
Timestamp	66
Title	87
TLatLngList	72
TListGeoCoderReponses	50
TMappilaryAPI	125
tmBicycling	111
tmDriving	174
tmDriving	111
tmWalking	174
tmWalking	111
TNativeMapServer	219
ToGpx	66
ToKml	1
toKml	66
ToKml	103
ToKml	125
ToKml	87
ToKml	72
ToKml	111
TopGroupZIndex	262
TopZIndex	262
ToShapes	66
TOSMFile	66
ToTxt	1
toTxt	66
ToTxt	103
ToTxt	108
ToTxt	125
ToTxt	87
ToTxt	97
ToTxt	72
ToTxt	168
ToTxt	111
ToTxt	147
ToTxtType	66
tour par tour	111
tour	1
TOverlayType	97
TPopupMenu	47
Tracking	97

TrackLine	262
TrafficArea	125
TrafficIconLowImpact	125
TrafficIconMinor	125
TrafficIconModerate	125
TrafficIconRoadClosed	125
TrafficIconSerious	125
TrafficLayer	125
traffic	125
TravelMode	174
TravelMode	111
tuiles vectorielles	206
TurnByTurn	111
type de carte	34

U

UEC_OSM_STYLESHEET	206
Undo	111
UnitSystem	174
Update	174
updateOptions	111
Update	111
UrlImage	125
Url	178
UseOpenMapQuestServices	50
useOSM	58
UserId	168
usMetric	174
UTFGrid	125

V

Values	66
vector icon	87
vector	262
vector	206
Vélib	401
ViewCount	217
view	217

Views	217
Visible	1
Visible	108
Visible	87
Visible	168
Visible	147
vue 3D	34

W

way	125
WayPoints	111
Ways	66
WeatherLayer	125
weather	125
weather	299
WebServer	377
WebSocket	219
Weight	111
Weight	206
Width	72

X

XAnchor	285
xapi	125
xapi	58
XMargin	178

Y

YAnchor	285
Yandex	178
YMargin	178

Z

Zindex	87
ZIndex	111
zip	201
ZoomAround	178
ZoomControl	47
zoom	47
ZoomPosition	47
Zoom	27
ZoomScaleFactorAround	178
ZoomScaleFactor	178
Zoom	147
Zoom	399