TomTom iPhone Mobile SDK User Guide

Welcome!

The purpose of this document is to introduce the basic functionality of the TomTom iPhone Mobile SDK. It is assumed that the reader is familiar with developing code in Objective C using Xcode.

For a more complete description of the library, please refer to the DOxigen files attached to this Release.

The TomTom iPhone Mobile SDK – from now on called the “SDK” - provides location-based services to iPhone/iPad applications.
These services include map rendering, geocoding, traffic and routing information.

Happy Developing !

The TTSD and LBS Teams.

Table of Contents

License Agreement

For Applications or product generated using the SDK, please add the following text in your License Screen, Copyright information or About page :

Using This TomTom product means that you are aware of TomTom's privacy policy http://www.tomtom.com/legal/privacy/ .

You agree to review the additional Copyrights at http://www.tomtom.com/legal/ and the TomTom Places Terms of Use at http://places.tomtom.com/disclaimer

THE LICENSED PRODUCTS AND MATERIALS ARE PROVIDED ON AN "AS IS" AND "WITH ALL FAULTS BASIS" AND TOMTOM AND ITS SUPPLIERS EXPRESSLY DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, SATISFACTORY QUALITY, ACCURACY, TITLE AND FITNESS FOR A PARTICULAR PURPOSE. NO ORAL OR WRITTEN ADVICE OR INFORMATION PROVIDED BY TOMTOM OR ANY OF ITS AGENTS, EMPLOYEES OR THIRD PARTY PROVIDERS SHALL CREATE A WARRANTY, AND LICENSEE IS NOT ENTITLED TO RELY ON ANY SUCH ADVICE OR INFORMATION. THIS DISCLAIMER OF WARRANTIES IS AN ESSENTIAL CONDITION OF THE AGREEMENT.

Acknowledgements:

The following have been used during the production of this application

TouchJSON:

Created by Jonathan Wight on 12/07/2005.

Permission is hereby granted, free of charge, to any person obtaining a copy of the toxics software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Route-Me:

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This software is based in part on the work of the Independent JPEG Group.

Creating your project

This section explains how to integrate the TomTom Mobile SDK in your own project.

Copy the libraries into your project folder.

With this release , you have a “.framework” folder, copy it into your project directory and add it to your project's frameworks.

Add system libraries

- CoreLocation
- libsqlite3.dylib
- libz.dylib
- QuartzCore
- SystemConfiguration
- CoreGraphics.framework
- UIKit.framework
- Foundation.framework

Add the following in the "Other Linker Flags" build settings.

- -ObjC
- -all_load (enable categories within the library)

Import TTLBSFramework where the SDK functionality is to be used

```
#import <TomTomLBS/TTLBSSDK.h>
```

Templates projects

Attached to this release you'll find the several templates that will help you started with the most used features of the SDK.

The followings are a few tips you might find useful:

1. As a suggestion we recommend you place your API key in the target's plist file

2. On the application delegate, configure the SDK Context: set the API key.

```
- (void)applicationDidFinishLaunching:(UIApplication *)application {

      // Set the API Key on the SDK context
    NSString* key = [[NSBundle mainBundle] objectForInfoDictionaryKey:@"api.key"];
      [[TTSDKContext sharedContext] setDeveloperKey:key];

    // Override point for customization after app launch
    [window addSubview:viewController.view];
    [window makeKeyAndVisible];
}
```

3. On the main view controller:
   - Open the XIB file in Interface builder. Configure your view.
   - Suggestion: add a view to hold the map view and set it to an IBOutlet, eg: called mapCanvas.

4. On the viewDidLoad, initialize your TTUIMapViewController with the frame of your map canvas UIView, and add the map controller's view to the map canvas view.

```
CGRect mapFrame=CGRectMake(0, 0,
mapCanvas.frame.size.width,mapCanvas.frame.size.height);
TTUIMapViewController* mapViewController =
      [[TTUIMapViewController alloc] initWithFrame: mapFrame];

[mapCanvas addSubview:mapViewController.view];
```

5. In the application plist, add the required properties with the hosts which will be used for the WMS tiles, Integer tiles, reachability test and location based services.

The name of these properties are :

1. tile.wms.base.url [arrray]
2. tile.integer.base.url [array]
3. api.lbs.base.url

for each item in the above list:

1. The url goes to : http://a.api.tomtom.com/lbs/wms (and “b” , “c” and “d”) and since they are all aliases, the objective is to balance the front end server. The API would rotate on this URLS for every call to fetch a WMS tile. This is a mandatory URL. The API will crash the application without it, because when we create a MapView, we add a tile layer default , which is a WMS layer (not Integer)

2. This url should go to http://a.api.tomtom.com/lbs. Important here is the "/lbs" ending, which send the request to the integer API. Note that you can also rotate the domain names from “a.api.tomtom.com” to “d.api.tomtom.com”.

3. This should point to : https://api.tomtom.com/lbs/services . Important here that the location in the server is "/lbs/services". This url is the access to the LBS services as Geocoding, Routes and fetching traffic incidents. Also note the “HTTPS”

NOTE: See the TemplateViewBasedApp-Info.plist file for these default values.

Now would be a good time to configure the SDK settings to better suit the application needs. Please see :

```
- (void) setMapViewConfigurations: (TTUIMapViewController*) mapViewController
```

Getting Started

The SDK is intended for use on any iPhone running 4.1.x or iPad running 3.2.x.

In order to access the SDK functionality an API key is required to access the services. If you have not yet received a key, please contact support.

The API key is used as follows:

```
[[TTSDKContext sharedContext] setDeveloperKey:key];
```

If you want to activate the use the newest style of icons and traffic tubes use:

```
[[TTSDKContext sharedContext] setTileStyle:2];
```

Services

The Basic services from the SDK are provided by the following classes :

TTAPILocation for Geocode and Reverse Geocoding.
TTAPITraffic for Traffic information and incidents.
TTAPIRouting for Routes generation and Routes directions fetching.
TTAPITileManager to fetch tiles directly from the servers.
TTAPIMapViewController for creating a Map View that can be added to an UIView in your project.

The sections below will describe in more detail how each service provider is accessed and used.

Geocoding.

The geocoding service provides two functions. The first is geocoding, which converts street address information to a physical location (i.e., latitude and longitude). The second is reverse geocoding, which as the name suggests, converts a physical location to street address information. This functionality is exposed through an interface called **TTAPILocation**.

Example:

```
TTAPILocation *location = [[TTAPILocation alloc ] init ];
TTAPIGeocodeData *data = [location geocode:[textField text]];
TTAPIGeoResult * firstResult = [[data getGeoResults] objectAtIndex:0];
```

For reverse geocoding, we could use :

```
TTCoordinates *coords = [mapViewController getMapCenterLocation];
TTAPILocation *location = [[TTAPILocation alloc ] init ];

startCoords = coords;
TTAPIReverseGeocodeData *results =
      [location reverseGeocodeWithLatitude:[coords latitude]
              andLongitude:[coords longitude]];
TTAPIReverseGeoResult * firstResult = [[results getReverseGeoResults] objectAtIndex:0];
```

Please refer to the Doxigen files for more information. Both methods return an array of objects.

Traffic information.

Traffic information can be retrieved be using the methods in the TTAPITraffic interface as seen in here:

```
trafficAPI = [[TTAPITraffic alloc] init];
[trafficAPI getTraffic:bbox withZoomLevel:zoom
      pixelExtent:[TTAPITraffic defaultPixelExtent]
      trafficModelId:trafficModelId];
```

The TrafficModelId can be “-1” to fetch the current traffic information. In order to get the current traffic model ID, you can use TTAPITrafficData , an object returned by “getTraffic” and get the “trafficId” Key (it is a NSString).

TTAPITrafficData contains an array of traffic incidents contained in the provided bounding box. See the Doxigen files for a complete description of the fields in this Array.

The Traffic Model ID changes in the server every 2 minutes. After this time, the traffic model ID is still valid to make way for older browser an applications to fetch traffic information and tiles, but after 4 minutes it will be invalid and trying to fetch traffic information using an invalid Traffic Model will generate an error.

In order to be aware of this Traffic Model changes (TMI) the SDK implements the following: Inside “TTAPILBSInfo” class, we see now

```
/**
Starts updating the TMI, periodically.Updates will be notified using the global
notification TTSDKUtils.TTSDKTrafficModelIdChangedNotification
 * @param seconds the update period
 */
- (void) startUpdatingTrafficModelID:(double)seconds;

//Stops updating the TMI.
- (void) stopUpdatingTrafficModelID;

/**
 * Gets the Current Traffic Model ID
 * @return The Traffic Model ID string
 */
   -  (NSString *) getCurrentTrafficModelId;
```

So, in order to be aware of the TMI changes we implement:

```
[[NSNotificationCenter defaultCenter] addObserver:self
      selector:@selector(tmiHasChangedNotification:)
      name:TTSDKTrafficModelIdChangedNotification object:nil];

// Check the server every 20 seconds.
[lbsInfo startUpdatingTrafficModelID:20];
```

This will make the “NotificationCenter” to call the method `“tmiHasChangedNotification” when the TMI changes.`

Routes

The Routing service is provided by the TTAPIRouting interface.

An example of this is as follows :

```
TTAPIRouting *routeManager = [[TTAPIRouting alloc]init];
TTAPIRoutingData *route =
     [routeManager getRouteWithStartPoint:startCoords
                   andEndPoint:endCoords
                   andRouteType:Quickest];
```

TTAPIRoutingData contains an array. This array of Coordinates objects contains the starting point, the destination point, and any desired intermediate points along the way. Please refer to the Doxigen files for more information.

The RoutingAPI also provides a way to get the Vector Path Points of a route.

```
TTAPIRoutingOptionalParameters *optionalParameters =
      [[TTAPIRoutingOptionalParameters alloc]init];

optionalParameters.includeDataPath = YES;
optionalParameters.dataPathZoomLevel = [mapViewController getZoomLevel];

TTAPIRoutingData *route=[routeManager getRouteWithStartPoint:startMarker.coordinates
                                                   andEndPoint:endMarker.coordinates
                                                  andRouteType:Quickest
                                          andOptionalParameters:optionalParameters];
```

In this snippet we see how to fetch these Vector Points, using a TTAPIRoutingOptionalParameters class. The values will be in the “route” object (TTAPIRoutingData) and can get fetched using:

```
[route getPathData]
```

This will return an array of TTCoordinate objects.

Map Tiles

The interface TTAPITileManager provides the service to fetch specific tiles from the server.

In order to fetch a traffic tile using a specific Traffic Model ID, we could use :

```
tileManager = [[TTAPITileManager alloc] init];
TTAPIWMSTileData *data =
     [tileManager getMapTileViaWMSWithCoordinates:location
     andZoomLevel: zoom];
```

This “data” object contains the map tile that can be used like :

```
UIImage *pic = [UIImage imageWithData:data.tile];
```

There are also several other methods in this interface to fetch different type of tiles. Please review the Doxigen files for more information.

Map View Controller

The Map view Controller is the interface to access the basic map services. It creates a Map View instance that can be added to an UIView in your project. The map can be manipulated using finger gestures on the screen or programmatically using methods to set the zoom level or setting a delegate when gesture events are generated or to rotate the map to a specific angle.

From the Template project attached to this release, we can see an example of how to use this interface.

To add a map to our UIView:

```
// Create a map controller with 100x10 dimensions.
controller = [[TTUIMapViewController alloc] initWithFrame:CGRectMake(0, 0, 100, 100)];

// Activate the free rotation
[controller setMapRotationsActive:YES];

//Center map on this coordinates
[controller centerOnLatitude:[NSNumber numberWithDouble:52.3] andLongitude:
[NSNumber numberWithDouble:4.9] withZoomLevel:10];

If you need to move the map only for a few pixels, then use the following
method:

CGSize Delta;
Delta.width = 10;
Delta.height= -10;
[MapController moveBy:Delta];
```

Adding a new Layer

```
// Add a new traffic layer ( use the Integer API )
[controller addLayer:[TTAPITileSourceFactory getIntegerBasicTrafficTileSource]];
// Add the map to our VIEW.
[mapContainer insertSubview: [controller view] atIndex:0];
```

Adding a Marker (Annotation)

The MapViewController also contains a MarkerManager functionality. This will allow us to add landmarks/POIs to the map.

To add a marker to the current map we could use :

```
TTUIMarker *marker = [TTUIMarkerFactory getLocationMarkerWithCoordinates:coordinates];
[controller addMarker:marker];
```

This will create a permanent marker(LandMark) on the map. In order to create your own marker (and Icon if needed) we could use:

```
TTUIMarker* marker = [[TTUIMarker alloc] initWithImage:[UIImage imageNamed:@"start.png"]
andOffset:offset andCoordinates:startCoords andData:nil];
```

NOTE: The object “offset” is an instance of “TTPoint” which represent how is the marker centered in the map. To center the marker in the middle of the Image, we use :

```
TTPoint *offset = [TTPoint pointWithX:[NSNumber numberWithDouble: 0.5] andY:[NSNumber
numberWithDouble: 0.5]] ; // centered at 50% width and 50% height.
```

Adding a Delegate

Developers can set a delegate (TTUIMapDelegate) class to get call backs from different events happening.

```
[controller setDelegate: delegate ];
```

a TTUIMapDelegate receives calls when a Marker is Tapped, or when there is a double tap somewhere in the Map. Please read the Javadoc Documentation on TTUIMapDelegate for more information.

Additional Features

Custom Layers

You can create custom layers to add your own Markers to it. These layers will be empty and they are useful for showing specific icons (like Weather or POIs).

To create an Empty Layer extend the TTAPITileSourceInteger class.
In the constructor add:

```
[super initWithBaseUrl:@"" parameters:nil];
 [self setType: TTMapLayerType_OtherTileSource];
 NSString *name = @"name of layer";
 self.ID = (long)[name hash];
 [self setName:name];
 [super setNetworkOperations:NO];
```

This will create a simple layer with no element. To add a marker to it use:

```
[myLayer addMarker: myMarker]
```

This layer will behave now similarly as the Traffic or Route layer: it can be hidden, shown or deleted.

Vector Based Route Layer

Developers can now add a new route layer in the View Controller that present the route as a drawn path. Line thickness and Colours can be customized. To create this type of layers use :

```
routePathLayer = [[TTUIVectorBasedRouteLayer alloc]initForController: mapViewController key:
route.summary.routeKey andParameters:optionalParameters];
```

Note that the new Layer needs a ViewController , the routeKey (from the RoutingData) and a copy of the optional parameters. Then the layer is added to the map with:

```
[mapViewController addVectorBasedLayer:routePathLayer];
```

It can be removed also with:

```
[mapViewController removeVectorBasedLayer:routePathLayer];
```

Developers can have more than one Vector Based Route Layer in any map.

Example of how to set features in this route layer.

Set the width of the route line in pixels.

```
source.lineWidth=8;
```

Set the colour fo the line

```
source.lineColor = color;
```

Set the color of the outline. This is the border. If nil then “lineColor” will be used.

```
source.outlineColor = [UIColor redColor];
```

Set the width of the outline in pixels. It must be less than “lineWidth” and greather than 0.

```
source.borderThickness = 2;
```

Set the transparency value for the line and outline. Value is from 0 to 1.0

```
source.transparency = value ;
```

Optimizing Tile Loading.

On iPhone 3G devices the process of downloading tiles using the network can be slow, even with WIFI access. When updating traffic or route tiles, it is possible to see “ghost” tiles when the download is taking process. To avoid this effect, there is a new setter in the “`TTUIMapViewController`”

```
TTUIMapViewController *controller;
[controller setTileLoadDelay: [NSNumber numberWithDouble:value]];
```

Value is a double from 0.2 to a max of 5 seconds. The default value is 0.7. If you set it to something lower than 0.2 you might experience low memory warnings.

The developer has also the option to disable the Slow Cache (SQLite database cache), so the loading tile process doesn't slow the painting thread.

```
[[TTSDKContext sharedContext] setDisableDBCache:disableDBCache];
```

If it is set to YES, then the Database cache is disabled.

To save the contents of the RAM Cache, use this method :

```
TTUIMapViewController *controller;
[controller saveCache];
```

Tile caching

The following information is generally not used or needed to use the iPhone SDK, and it is added here only for information purposes. The iPhone SDK has 4 levels of access to the map tiles, based from the Route-Me api:

1. Screen
    Tiles visible on the screen are kept in a memory structure.

2. Memory caching
    There is a small memory cache for tiles recently added to the screen.

3. Database caching
    A larger cache is implemented using a database to both speed up tile loading and provide offline map browsing.

4. Network
    Tiles are fetched from the TomTom's map tile services if not already on the device's caches.

Tile sources classification

1. Fast sources

Both Screen and Memory caching are considered fast caches.
Tiles on the fast caches are added to the current screen immediately when requested.

2. Slow sources

Both database cache and network are considered slow caches.
Tiles on these sources are loaded on separate threads, maintained on a queue.

Cache levels by layer type

1. Map tiles can use all levels of caching: screen, memory and database.

2. Route and traffic layers don't use the database cache because these layers are volatile.

Caching is independent for each layer, meaning that each layer will have its own caching policy and cache objects.

Default cache configuration

By default, memory cache can hold 32 tiles.
By default, database cache can hold 1000 tiles.

Cache configuration

Caching for the map tiles layers can be configured by including a routeme.plist file on the client application.

Caching can also be configured on the layer level.

routeme.plist configuration for the map tiles.

To configure the map cache through the routeme.plist file the developer will need to include the following xml in the file:

```
<plist version="1.0">
<dict>
	<key>caches</key>
	<array>
		<dict>
			<key>type</key>
			<string>memory-cache</string>
			<key>capacity</key>
			<integer>64</integer>
		</dict>
		<dict>
			<key>type</key>
			<string>db-cache</string>
			<key>capacity</key>
			<integer>1000</integer>
			<key>strategy</key>
			<string>LRU</string>
			<key>useCachesDirectory</key>
			<false/>
			<key>minimalPurge</key>
			<integer>100</integer>
		</dict>
	</array>
</dict>
</plist>
```

The memory cache is identified by the key "memory-cache" and the database cache is identified by the key "db-cache".
If the developer only wants to configure one of the caches, the other can be omitted.

For the memory cache it is only possible to configure its size.
For the database cache it is possible to configure its size, number of old tiles purged each time the cache is full and needs to be cleared, and the type of purging (last recently used or first in first out).

It is not possible to have two map tile sources with different cache configurations when the configuration is done with the routeme.plist file.

Programmatic source cache configuration

1. Create a new tile source.
2. Create a TTAPITileSourceCacheConfig object:
3. Set the parameters the developer wishes to configure: cache key (mandatory), capacities, types and purging strategy.
4. Replace the tile source's default configuration object with the one created;
5. Add the source to the map view.

Any modifications done to the cache configuration object after the source is added to the map will not have any effect on the cache's configuration.

If some of the cache configuration fields are not configured, the SDK will use its default values instead of the ones defined in the routeme.plist file.

The "useCachesDirectory" element is defunct and should be set to false.

Next is an example of how to create and configure a TTAPITileSourceCacheConfig instance defining an 1000 tiles DB cache and a 32 tiles memory cache:

```
TTAPITileSourceCacheConfig *cacheConfiguration = nil;
NSMutableArray *capacities = [[[NSMutableArray alloc] initWithCapacity:2] autorelease];
NSMutableArray *types = [[[NSMutableArray alloc] initWithCapacity:2] autorelease];

// Define a DB cache for 1000 tiles
[types addObject: [NSDictionary dictionaryWithObject: @"memory-cache" forKey: @"type"]];
[capacities addObject: [NSNumber numberWithInt: 1000]];

// Define a memory cache for 32 tiles
[types addObject: [NSDictionary dictionaryWithObject: @"db-cache" forKey: @"type"]];
[capacities addObject: [NSNumber numberWithInt: 32]];
cacheConfiguration = [[TTAPITileSourceCacheConfig alloc] initwithCacheKey: cacheKey
                          andCapacities: capacities andTypes: types];
```

Closing Notes

Thank you again for downloading the iPhone SDK for the LBS Platfom. Come visit us in our developer portal for the latest updates, samples and community forums at

http://developer.tomtom.com