TomTom Mobile SDK QuickStart Guide

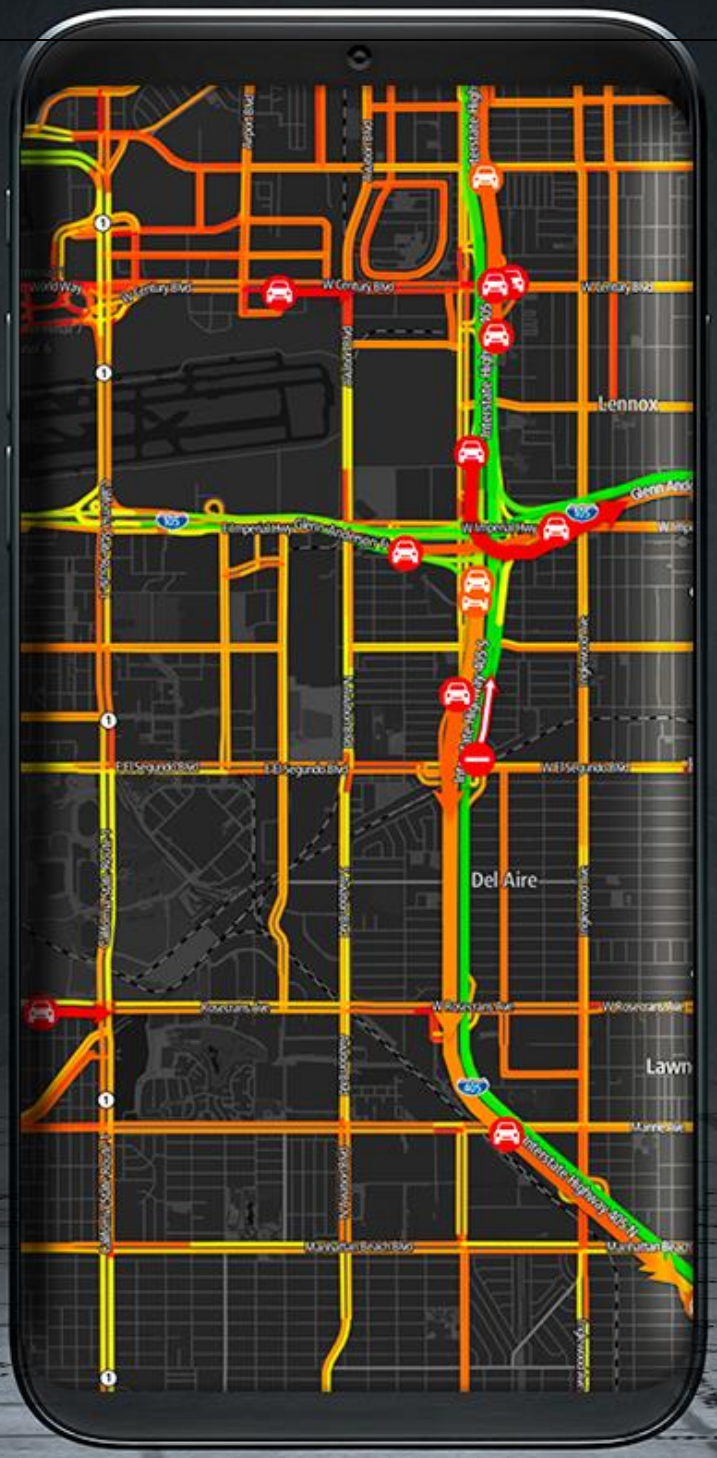

Table of Contents

Introduction

Welcome to the TomTom Mobile SDK QuickStart Guide! This document compiles various handy tutorials from the TomTom developer page to get you started fast.

The “Migrate to TomTom” section serves as a how-to for basics like displaying a map, adding custom markers, traffic and routing layers. Each section concludes with links to advanced examples to go even further. Let’s get started!

Migrate to TomTom – iOS

Prerequisites

Before you start writing code, prepare your environment:

1. Ensure that XCode and iOS simulator are installed.
2. Ensure that Cocoapods is installed on your system. The TomTom Maps SDK for iOS is delivered as a Cocoapods pod.
3. Create a new **Single View App** project with a deployment target of **9.1** or higher. After the project has been successfully created, close it and create an empty Podfile by executing a `'pod init'` command inside the project folder.
4. Open the Podfile and paste the following lines inside a 'target' section:

```
pod 'TomTomOnlineSDKMaps'
pod 'TomTomOnlineSDKRouting'
pod 'TomTomOnlineSDKMapsUIExtensions'
```

5. Save and close the Podfile.
6. Execute the command `'pod install'`. Cocoapods will install all of the required dependencies.
7. Open a `*.xcworkspace` file in Xcode.
8. Obtain an API Key

Initializing a map

Provide TomTom API key inside the *Info.plist* file.

```
<key>OnlineMap.Key</key>
<string>YOUR_KEY_GOES_HERE</string>
<key>OnlineTraffic.Key</key>
<string>YOUR_KEY_GOES_HERE</string>
<key>OnlineRouting.Key</key>
<string>YOUR_KEY_GOES_HERE</string>
```

1. Import the appropriate header inside the ViewController.h file:

```
#import <TomTomOnlineSDKMaps/TomTomOnlineSDKMaps.h>
```

2. Add a `TTMapView` property inside the *ViewController.m* file. Initialize the map by instantiating a `TTMapView` object and switching the ViewController view to the TTMapView view

```
@interface ViewController ()
@property TTMapView* mapView;
@end
@implementation ViewController
- (void)viewDidLoad {
   [super viewDidLoad];
   self.mapView = [[TTMapView alloc] initWithFrame:self.view.frame];
   CLLocationCoordinate2D amsterdamCoords = CLLocationCoordinate2DMake(52.377271, 4.909466);
   [self.mapView centerOnCoordinate:amsterdamCoords withZoom:11];
   self.view = self.mapView;
```

Displaying a marker

Use a `TTAnnotation` object to display a marker:

```
TTAnnotation *annotation = [TTAnnotation annotationWithCoordinate:amsterdamCoords];
[self.mapView.annotationManager addAnnotation:annotation];
```

Displaying traffic

TomTom provides two kinds of traffic information:

- ***Traffic flow*** *shows the difference between current and free flow speed. Green indicates that the speeds are the same, meaning there are no traffic jams. Red indicates that traffic is much slower than free flow, meaning that there are traffic jams.*
- ***Traffic incidents*** *indicates specific traffic problems such as closed roads, rain, or accidents.*

1. You can show traffic incidents, traffic flow, or both:

```
self.mapView.trafficIncidentsStyle = TTTrafficIncidentsStyleRaster;
self.mapView.trafficIncidentsOn = YES;
self.mapView.trafficFlowOn = YES;
```

Displaying a route/directions

1. Import the 'TomTomOnlineSDKRouting.h' header in *ViewController.h*:

```
#import <TomTomOnlineSDKRouting/TomTomOnlineSDKRouting.h>
```

2. Update *ViewController.h* so that it conforms to the TTRouteResponseDelegate protocol.

```
@interface ViewController : UIViewController <TTRouteResponseDelegate>
- (void)route:(TTRoute *)route completedWithResult:(TTRouteResult *)result;
- (void)route:(TTRoute *)route completedWithResponseError:(TTResponseError *)responseError;
@end
```

3. Create and initialize TTRoute and TTRouteQuery objects inside the viewDidLoad method in *ViewController.m*.

```
CLLocationCoordinate2D routeStart = CLLocationCoordinate2DMake(52.376368, 4.908113);
CLLocationCoordinate2D routeStop = CLLocationCoordinate2DMake(52.372281, 4.846595);
TTRouteQuery *routeQuery = [[[TTRouteQueryBuilder alloc] initWithDest:routeStop withOrig:routeStart] build];
TTRoute *route = [[TTRoute alloc] init];
[route planRouteWithQuery:routeQuery withAsyncDelegate:self];
```

The code above builds a TTRouteQuery object from two route points. It then initializes a TTRoute object and sends the 'planRouteWithQuery'message to it. The message has two parameters: the newly created routeQuery object and a pointer to the ViewController instance to allow the 'completedWithResult' method to be called.

4. Finally, provide an implementation for the 'completedWithResult' method inside of the *ViewController.m* file:

```
- (void)route:(TTRoute *)route completedWithResult:(TTRouteResult *)result {
   for(TTFullRoute *fullRoute in result.routes) {
      TTMapRoute *mapRoute = [TTMapRoute routeWithRouteData:fullRoute];
      [self.mapView.routeManager addRoute:mapRoute];
      mapRoute.active = YES;    }}
```

The above method provides a TTRouteResult object containing TTFullRoute objects. It then loops through the TTFullRoute collection. For each TTFullRoute object, it creates a new TTMapRoute instance and adds it to the MapView.

Going further

Now you're ready to explore more awesome TomTom functionality with some more advanced tutorials and examples:

- Map tiles
- Traffic layers
- Routing examples
- Search examples

Helpful links

- Getting a free API key and creating a local setup for the TomTom Maps SDK for iOS.
- Source code on GitHub

Migrate to TomTom – Android

Prerequisites

Before you start writing code, prepare your environment:

1. Install Android Studio. During installation, make sure that the component **"Android Virtual Device"** is selected.
2. If possible, start a new project (minimum **SDK API 19 – Android 4.4 "KitKat"**) with an Empty activity. You can use your own project as long as it meets the minimum Android SDK API level requirement
3. Update the Source Compatibility and Target Compatibility to 1.8 in the Project Structure dialog (click File > Project Structure).
4. Add the TomTom repository to your project's gradle.build file:

```
allprojects {
   repositories {
      google()
      jcenter()
      maven {
         url "https://maven.tomtom.com:8443/nexus/content/repositories/releases/" }}}
```

5. Create and add an API key to AndroidManifest.xml:

Copy and paste the API key to your AndroidManifest.xml inside the `<application>` tag:

```
<meta-data android:name="OnlineMaps.Key" android:value="<YOUR_KEY_GOES_HERE>" />
<meta-data android:name="OnlineTraffic.Key" android:value="<YOUR_KEY_GOES_HERE>" />
<meta-data android:name="OnlineSearch.Key" android:value="<YOUR_KEY_GOES_HERE>" />
<meta-data android:name="OnlineRouting.Key" android:value="<YOUR_KEY_GOES_HERE>" />
```

Initializing a map

1. Add dependencies to your module's *gradle.build* file:

```
api("com.tomtom.online:sdk-maps:2.085@aar") {
   transitive = true}
```

2. Add a map fragment to the main activity layout:

```
<fragment
android:id="@+id/mapFragment"
android:name="com.tomtom.online.sdk.map.MapFragment"
android:layout_width="match_parent"
android:layout_height="match_parent" />
```

Displaying a marker

1. Define field `map` in `MainActivity`:

```
private TomtomMap map;
```

2. Implement an `OnMapReadyCallback` interface in your `MainActivity` class and override the `onMapReady` method:

```
@Override
public void onMapReady(@NonNull TomtomMap tomtomMap) {
   this.map = tomtomMap;
   LatLng amsterdam = new LatLng(52.37, 4.90);
   SimpleMarkerBalloon balloon = new SimpleMarkerBalloon("Amsterdam");
   tomtomMap.addMarker(new MarkerBuilder(amsterdam).markerBalloon(balloon));
   tomtomMap.centerOn(CameraPosition.builder(amsterdam).zoom(7).build());}
```

3. Inside the MainActivity onCreate method, get the map fragment instance and set a callback object on MainActivity so that the onMapReady method is called:

```
MapFragment mapFragment = (MapFragment) getSupportFragmentManager()
   .findFragmentById(R.id.mapFragment);
mapFragment.getAsyncMap(this);
```

Displaying traffic

Create two simple buttons to work with traffic on the map. One enables the traffic layer, and the other disables it. TomTom provides two services that offer traffic information:

- ***Traffic flow** shows the difference between current and free flow speed. Green indicates that the speeds are the same, meaning there are no traffic jams. Red indicates that traffic is much slower than free flow, meaning that there are traffic jams.*
- ***Traffic incidents** indicates specific traffic problems such as closed roads, rain, ice on the road, or accidents.*

1. Create the buttons to handle traffic visualization and call the turnOnRasterTrafficIncidents and turnOnRasterTrafficFlowTiles methods on the map UiSettings object, choosing whether to display traffic flow tiles, traffic incident tiles, both, or none.

```
Button btnTrafficOn = findViewById(R.id.btnTrafficOn);
btnTrafficOn.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View v) {
      map.getUiSettings().turnOnRasterTrafficIncidents();
      map.getUiSettings().turnOnRasterTrafficFlowTiles();
   } });
Button btnTrafficOff = findViewById(R.id.btnTrafficOff);
btnTrafficOff.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View v) {
      map.getUiSettings().turnOffTraffic();
   } });
```

Displaying a route/directions

1. Add a dependency to the module's gradle.build file:

```
api("com.tomtom.online:sdk-routing:2.085@aar") {transitive = true }
```

2. Now add a button for displaying the route to the main activity layout XML file:

```
<Button
android:id="@+id/btnRouteShow"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_marginBottom="8dp"
android:layout_marginEnd="8dp"
android:layout_marginStart="8dp"
android:text="Route SHOW" app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintStart_toStartOf="parent" />
```

3. Finally, add a proper handler for our new button:

Planning a route requires at least two location points (e.g.: your house and the store). Create an instance of the RoutingApi object by calling OnlineRoutingApi.create method. Then construct a route query using the

selected location points. There are other options that allow the app to plot different kinds of routes (fastest, shortest, etc).

The `planRoute` returns an observable `RouteResult` object. You can get `FullRoute` objects from the `RouteResult.getRoutes()` method, create a new RouteBuilder object from each of them, and add them to your map.

```
Button btnRouteShow = findViewById(R.id.btnRouteShow);
btnRouteShow.setOnClickListener(new View.OnClickListener() {
  @Override
  public void onClick(View v) {
    LatLng amsterdam = new LatLng(52.37, 4.90);
    LatLng hague = new LatLng(52.07, 4.30);
    RoutingApi routingApi = OnlineRoutingApi.create(getApplicationContext());
RouteQuery routeQuery = new RouteQueryBuilder(amsterdam, hague).withRouteType(RouteType.FASTEST);
    routingApi.planRoute(routeQuery)
      .subscribeOn(Schedulers.newThread())
      .observeOn(AndroidSchedulers.mainThread())
      .subscribe(routeResult -> {
        for (FullRoute fullRoute : routeResult.getRoutes()) {
          RouteBuilder routeBuilder = new RouteBuilder(
            fullRoute.getCoordinates()).isActive(true);
          map.addRoute(routeBuilder);
        } }); }});
```

Going further

Now you're ready to explore more awesome TomTom functionality with some more advanced tutorials and examples:

- Map tiles
- Traffic layers
- Routing examples
- Search examples

Helpful links

- Getting a free API key and creating a local setup for the TomTom Maps SDK for Android.
- Source code on Github

Thank you

Thank you for using TomTom. We hope this guide was helpful and look forward to seeing what you will build. Thank you!