HERE iOS SDK

Developer's Guide

Premium Edition Version 3.15

Important Information
Notices

Topics:

This section contains document notices.

Legal Notices

Trademark Acknowledgements

HERE is trademark or registered trademark of HERE Global B.V.

Other product and company names mentioned herein may be trademarks or trade names of their respective owners.

Disclaimer

This content is provided "as-is" and without warranties of any kind, either express or implied, including, but not limited to, the implied warranties of merchantability, fitness for a particular purpose, satisfactory quality and non-infringement. HERE does not warrant that the content is error free and HERE does not warrant or make any representations regarding the quality, correctness, accuracy, or reliability of the content. You should therefore verify any information contained in the content before acting on it.

To the furthest extent permitted by law, under no circumstances, including without limitation the negligence of HERE, shall HERE be liable for any damages, including, without limitation, direct, special, indirect, punitive, consequential, exemplary and/ or incidental damages that result from the use or application of this content, even if HERE or an authorized representative has been advised of the possibility of such damages.

Document Information

Product	
Name:	HERE iOS SDK
Version:	Premium Edition Version 3.15
Document	
Name:	HERE iOS SDK Developer's Guide
ID:	42093a74-310f-4799-b7fb-c00027dfd0cd
Status:	FINAL
Date:	2020-Mar-26, 16:02 (GMT)

Contents

Chapter 1
Overview

Topics:

The articles that follow introduce HERE iOS SDK, explain essential concepts, and describe the common use cases it supports.

What Is the HERE iOS SDK?

HERE iOS SDK provides a set of programming interfaces that enable developers to build immersive, geographically-aware iOS applications by leveraging a powerful and flexible mapping platform. Through this SDK, developers can add rich location features such as routing, interactive maps, and global place search to their applications. The powerful client-side HERE iOS SDK also includes a sophisticated engine for rendering map data and calculated routes. In addition to dynamically downloading map data, the SDK also supports offline maps using previously cached map data or downloaded map packages.

Feature List

The main features offered by HERE iOS SDK are listed below.

Note: You may not have access to all features listed below. The enabled features are determined based on your business plan.

Mapping:

- Dynamically download vector maps for over 190 countries in over 60 languages
- Preload maps for offline usage
- Map styles: Normal, Night, Satellite with Streets, Terrain, and more
- Touch gestures such as tap, pan, and pinch
- Overlay objects on the map such as polylines, polygons, icons, routes
- Map marker clusters
- 3D landmarks
- Overlay custom raster tiles on the map (useful for features such as heat maps)
- Show real-time traffic flows and incidents
- 3D venue (indoor) maps
- 3D extruded buildings
- Transit object interaction
- 3D map objects

Search:

- Search through a broad set of geographical content across the globe (including streets, address points, and categorized places)
- Search Places for something specific or explore by categories
- Access rich details for a Point of Interest from third-party content sources (including images, ratings, reviews, and editorials)
- Perform geocoding and reverse geocoding lookups
- Offline Places search, Geocoding, Reverse Geocoding

Directions:

- Online Car, Public Transit, Bicycle, Truck, Scooter, and Pedestrian Route Directions

- Specify preferred route type (fastest or shortest) and route options (such as avoiding toll roads, motorways, and parks)
- Alternate routes
- Saving a route as a file
- Offline route calculation
- Driving directions with traffic taken into account
- Public Transit directions using online timetables information
- Indoor routing

Turn-by-turn Navigation:

- Turn-by-turn navigation for pedestrian, car, scooter, and truck routes
- Download and set additional navigation voices
- Natural-sounding guidance instructions such as "turn left at the gas station" and "at the next light, turn right"
- Recorded audio and speech synthesis voices in a variety of languages. For a list of the available languages, see the Developer's Guide.

HERE Positioning:

- Bluetooth network-based positioning including:
  - Indoor positioning
  - Private Indoor positioning

LiveSight (Augmented Reality):

- Project objects in a camera view
- Track position of the device and objects in space
- Gesture support allows the user to interact with content
- Configurable LiveSight engine allows the user experience to be customized

Other Features:

- Custom location and custom location geometry search
- Support for fleet dispatching and connectivity
- Map information for fleet vehicles and trucks
- Congestion toll zones and the typical traffic patterns for a given time of the week
- Ability to retrieve the toll cost of a route

Legal Requirements

In addition to the applicable *terms and conditions* under which you have licensed the SDK, the following shall apply.

Components of HERE SDK collect certain information from your application. Such information includes access credentials (`licenseKey`, `App_Id` and `App_Code` – see also *Authenticating Applications* on page 22) and the types of features utilized by your application when used by end users. The information does not identify an individual end user. However, your application privacy policy must disclose to the end users that you have

licensed products and services from HERE and that such information is collected from your application as it is being used by end users and that HERE collects and processes such information from the application.

Service Support

If you need assistance with this or any other HERE product, select one of the following options.

- If you have a HERE representative, contact them when you have questions/issues.
- If you manage your applications and accounts through *developer.here.com*, log into your account and check the pages on the SLA report or API Health. If this does not clarify the issue, then check *stackoverflow.com/questions/tagged/here-api*.
- If you have an evaluation plan, check *stackoverflow.com/questions/tagged/here-api*.
- If you have questions about billing or your account, *Contact Us*.
- If you have purchased your plan/product from a HERE reseller, contact your reseller.

Chapter 2
Quick Start

Topics:

This section provides information to help you start using HERE iOS SDK.

Obtain the SDK via CocoaPods

You can obtain HERE iOS SDK via CocoaPods. For more information, see *Run the Sample Application* on page 13.

Obtain the SDK via the Download Method

To obtain HERE iOS SDK, follow these steps:

1. Visit *developer.here.com* and click on **Get Started for Free**.
2. Register or sign in to your HERE Account.
3. Agree to HERE Terms and Conditions.
4. On **Platform Activiation** screen find the section for **HERE iOS SDK Premium Edition** and click on **Generate App ID and App Code** to obtain your app credentials and the SDK download link.
5. Follow the link to download HERE iOS SDK as a Zip package.

After you have downloaded and extracted the Zip package, follow one of the following tutorials to begin using the SDK.

Run the Sample Application

This tutorial provides instructions on how to run the Objective-C and Swift sample applications to render a map on an iOS device. The tutorial assumes you are using Xcode 11.3 and the iOS 12 SDK. For more details, see *System Requirements* on page 22.

Tasks for this basic application include:

- Acquire HERE Credentials.
- Launch the sample project using CocoaPods or manually from the package.

Acquire HERE SDK Credentials

Before working with HERE SDK you need to acquire a set of credentials by registering your application on *http://developer.here.com*. Each application requires a unique set of credentials.

Launch the Project Using Cocoapods

You can start trying a sample HERE iOS SDK project by using Cocoapods. CocoaPods is an open source dependency manager for Objective-C and Swift Xcode projects. To get started with using CocoaPods on macOS, follow the instructions in *Getting Started* guide on CocoaPods.org.

Once you have configured CocoaPods in your development environment, try the tutorial project by using the following command.

```
pod try HEREMaps
```

Note: This command is only intended for product demonstration purposes. It only puts the HERE SDK in a temporary location. For more information on using CocoaPods in an Xcode project, see *Create a Simple App Using HERE SDK* on page 14.

After invoking this command you are prompted to choose between an Objective-C or a Swift sample project. Make a selection, and the script opens a project in Xcode with HERE SDK dependencies already configured. You can then follow the instructions in the project `README.txt` to add app credentials and run the app.

App credential strings are located in `HelloMapAppDelegate.m` file with the following labels:

- `kHelloMapAppID`
- `kHelloMapAppCode`
- `kHelloMapLicenseKey`

Launch the Project from the Downloaded HERE iOS SDK Package

You can also open a sample project from HERE iOS SDK package which you can download from *http://developer.here.com*. The Xcode projects are available in the `sample-apps` folder. To run the project, double-click on `HelloMap.xcodeproj` or `SwiftHelloMap.xcodeproj`, then follow the instructions in the `README.txt` file.

Create a Simple App Using HERE SDK

This guide provides instructions on creating a simple HERE iOS SDK application to render a map on an iOS device. Users are able to navigate the map by way of touch gestures such as panning, rotating, tilting, and pinching. The contents of this guide apply to Xcode 11.3 and iOS 12 SDK.

This tutorial applies to development using Objective-C. For information on how to perform the same tasks using Swift see *Create a Simple HERE SDK App Using Swift* on page 205.

Note: HERE iOS SDK is now distributed as a dynamic framework instead of a static library. Please review the following steps and update your Xcode project configuration if you are upgrading from an older versions of HERE SDK. Also, ensure you first remove the old `NMAKit.framework`, `NMABundle.bundle`, and linked libraries from your Xcode project before you add the new dynamic framework.

Acquire HERE SDK Credentials

Before working with HERE SDK you need to acquire a set of credentials by registering your application on *http://developer.here.com*. Each application requires a unique set of credentials.

Create a New Single View Application

1. From Xcode menu select **File** > **New** > **Project** to open the New project dialog (or press Shift + Command + N).
2. Select **iOS** > **Application** > **Single View Application** as the application type you want to create. Press Next.
3. In the next dialog enter your **Product Name** (such as `HelloMap`) and **Organization Identifier** (such as `edu.self`). The resulting Bundle Identifier must be the same as you have registered on *developer.here.com*.
4. Next, choose "Objective-C" under **Language**, then click **Next**. Navigate to the directory where you want your project to be stored and then select **Create**.
5. The next step is to configure this project to use HERE SDK.

Add HERE iOS SDK

There are two ways to add HERE iOS SDK to your Xcode project:

- CocoaPods
- Manually import the dynamic framework

To use CocoaPods to import HERE SDK:

1. Create a Podfile in your project and add the following. Replace `YourApp` with your project name.

```
platform :ios, '12.0'
target 'YourApp' do
 pod 'HEREMaps'
end
```

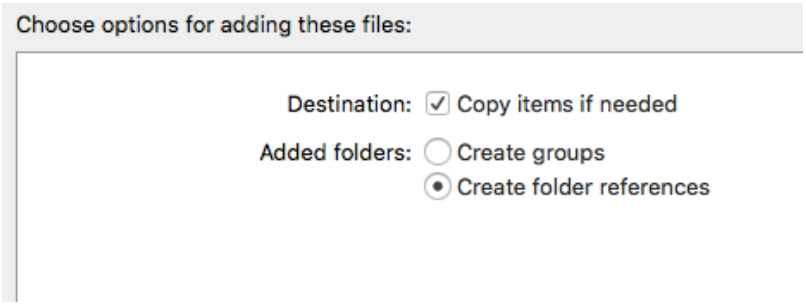
2. Next, open a command-line terminal on your workstation. From your project root folder run this command:

```
pod install
```

Alternatively, to manually import HERE SDK dynamic framework:

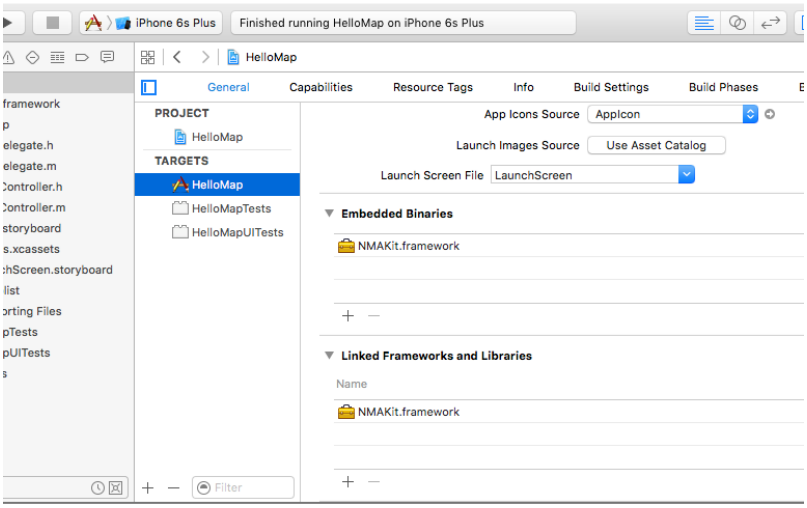
1. Download HERE iOS SDK package from *http://developer.here.com* and extract it to somewhere in your local file system.
2. Add the `NMAKit` dynamic framework to your Xcode project. Click on your app target and choose the "General" tab. Find the section called "Embedded Binaries", click the plus (+) sign, and then click the "Add Other" button. From the file dialog box select "NMAKit.framework" folder. Ensure that "Copy items if needed" and "Create folder reference" options are selected, then click **Finish**.

Figure 1: Adding NMAKit.framework



3. Ensure that `NMAKit.framework` appears in "Embedded Binaries" and the "Linked Frameworks and Libraries" sections.

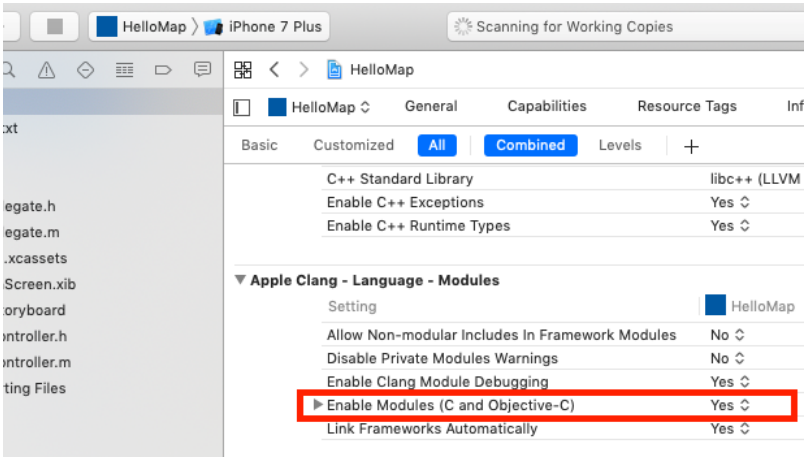
Figure 2: Added Embedded Binaries



Verify and Run the App

1. Next, ensure that modules are enabled. Click on the "Build Settings" tab and navigate to "Apple Clang - Language - Modules" section. Ensure that "Enable Modules (C and Objective-C)" has the value "YES".

Figure 3: Enable Modules



2. Run the application. From the Xcode menu bar select **Product** > **Run**. Ensure that the project runs in iOS Simulator without errors.
3. HERE iOS SDK is now ready for use in your Xcode project. Now that you have your project configured to work with HERE SDK, try extending the sample application to render a map.

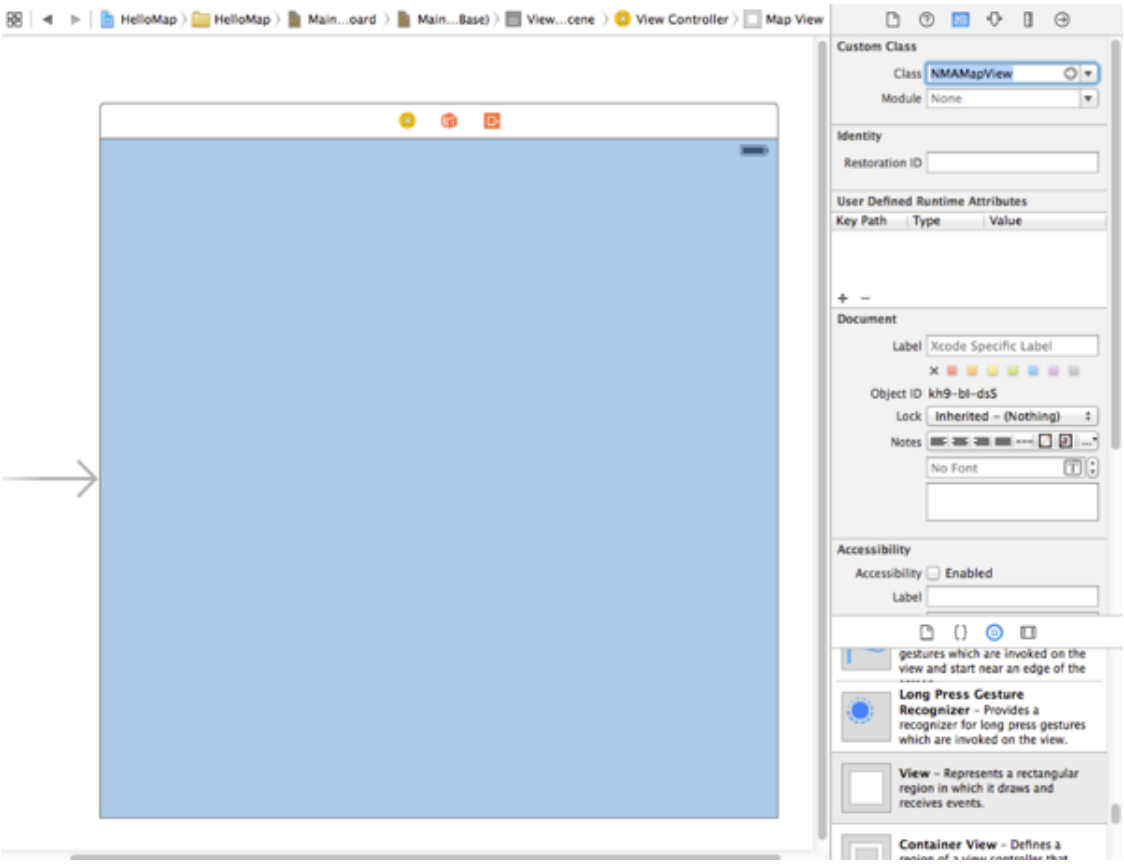
Create the Map View

In this section we utilize the `NMAMapView` and `NMAGeoCoordinates` classes to render a Map.

1. Create an `NMAMapView`.
   a. Select `Main.storyboard` in the navigator, then open the Utilities view by pressing the key combination Command + Option + Control + 3. Drag and drop a View object from the Object Library onto the View Controller. If necessary, resize the View so it takes up the entire viewable area.
   b. In the Interface Builder click on the created View and then open the Identity Inspector in the Utilities view by pressing the key combination Command + Option + 3. Change the class value from `UIView`

to `NMAMapView` and press return. In the Document Outline you should see that the name of the View has changed from *View* to *Map View*.

Figure 4: MapView



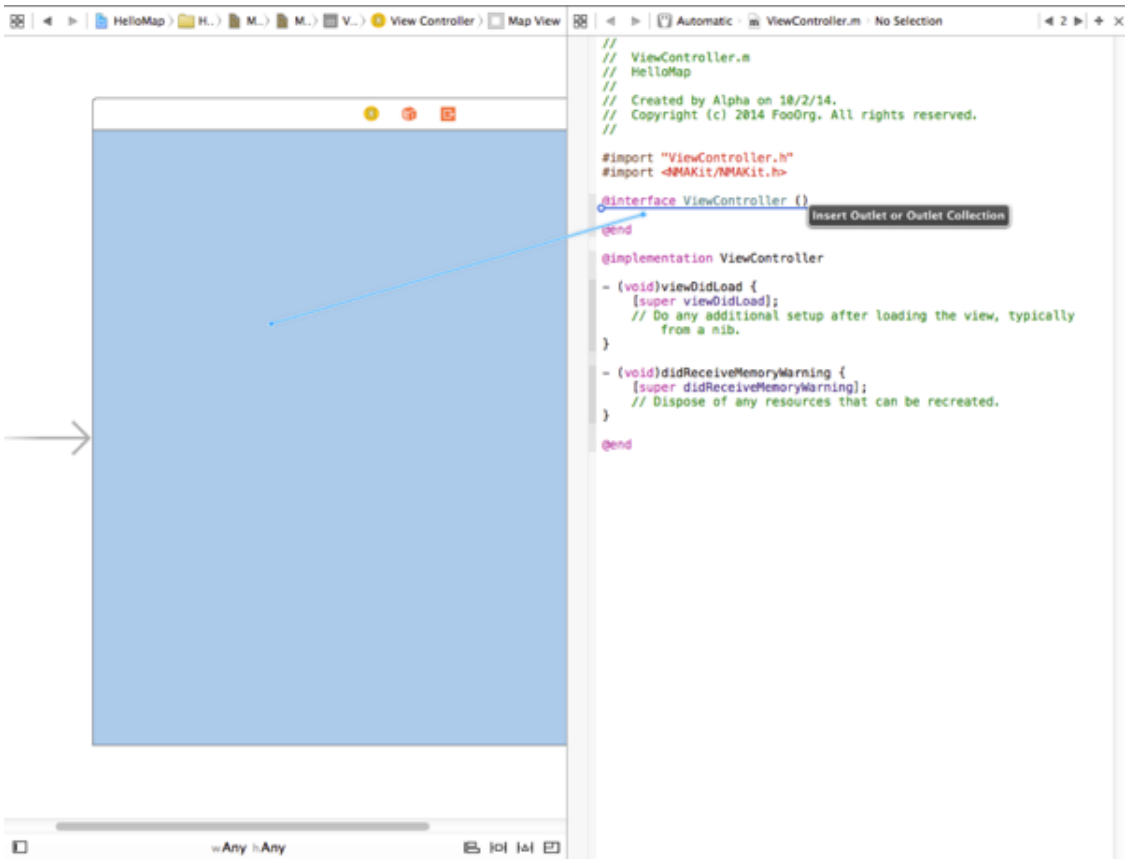
2. Create an outlet to `NMAMapView` in *ViewController*.
   a. Select `Main.storyboard` in the navigator.
   b. Press Command + Option + Return to open the Assistant Editor. It should show *ViewController.m*.
   c. Add the following import statement to the top of this file:

```
@import NMAKit;
```

   d. Hold the Control key on the keyboard and click to drag from the Map View to the interface block in *ViewController.m*. You should see a blue line and a tooltip which says "Insert Outlet or Outlet Connection". Release the mouse button and a dialog appears allowing you to create an outlet.

e. Name the outlet *mapView*, keep the other default options, and then select Connect.

Figure 5: Create an Outlet



3. Now an outlet to `NMAMapView` is set. The modified file should be as follows:

```
#import "HelloMapViewController.h"

@import NMAKit;

@interface HelloMapViewController ()
@property (weak, nonatomic) IBOutlet NMAMapView *mapView;
@end

@implementation HelloMapViewController

- (void)viewDidLoad
{
 [super viewDidLoad];
}

- (void)didReceiveMemoryWarning
{
 [super didReceiveMemoryWarning];
}

@end
```

4. Implement `NMAMapView` setup and lifecycle code by modifying the `viewDidLoad` method as shown:

```
- (void)viewDidLoad
{
 [super viewDidLoad];
```

```
 //set geo center
 NMAGeoCoordinates *geoCoordCenter =
  [[NMAGeoCoordinates alloc] initWithLatitude:49.260327 longitude:-123.115025];
 [self.mapView setGeoCenter:geoCoordCenter withAnimation:NMAMapAnimationNone];
 // Note, logo is not shown when its size is greater than 1/8 of NMAMapView's height or width.
 self.mapView.copyrightLogoPosition = NMALayoutPositionBottomCenter;

 //set zoom level
 self.mapView.zoomLevel = 13.2;
}
```

5. Add your HERE application credentials.
   a. Open *AppDelegate.m* and import *NMAKit* by adding the following import statement to the top of the file.

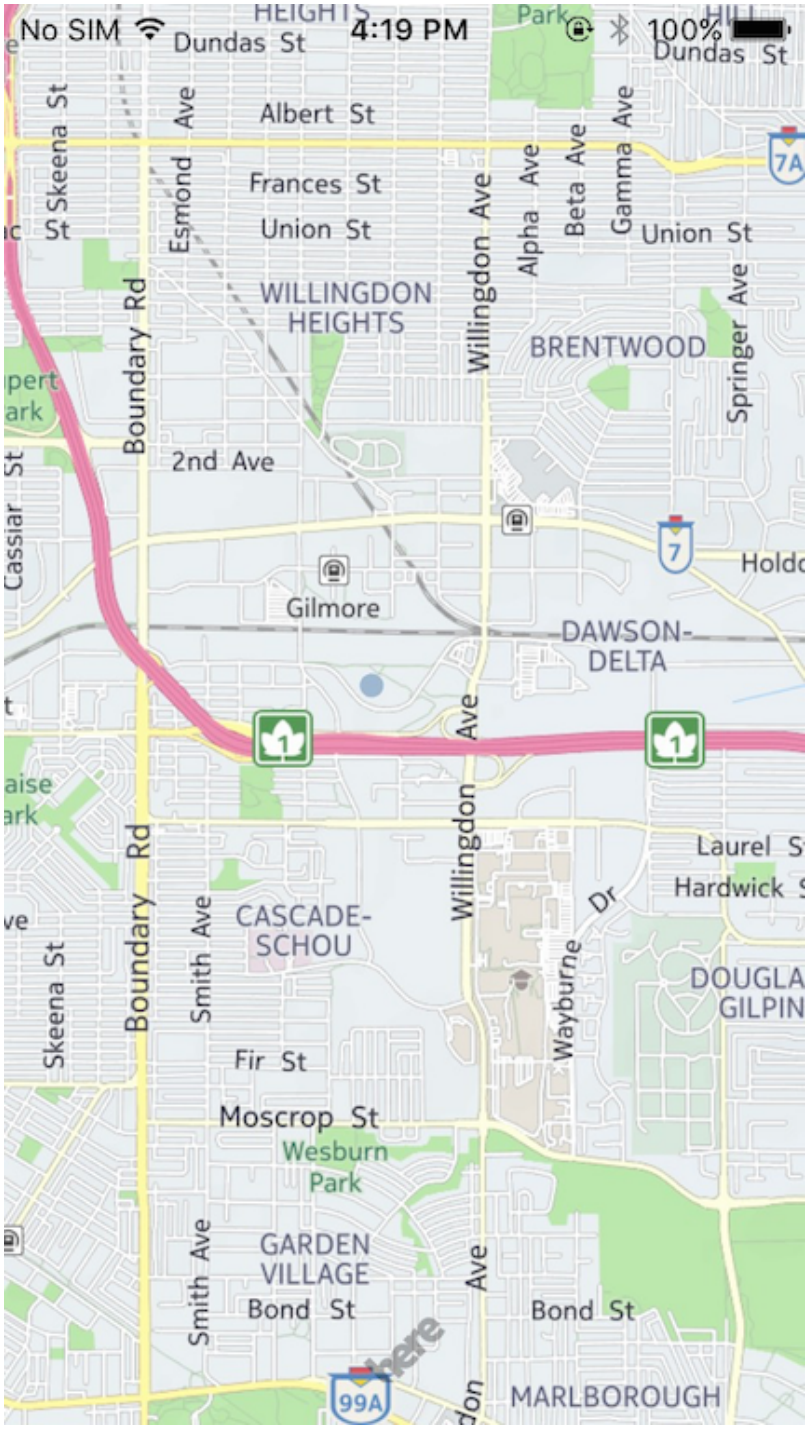
```
@import NMAKit;
```

   b. Add the following to `didFinishLaunchingWithOptions` method replacing `YOUR_APP_ID`, `YOUR_APP_CODE` and `YOUR_LICENSE_KEY` with the credentials that you received from *http://developer.here.com*.

```
[NMAApplicationContext setAppId:@"YOUR_APP_ID"
      appCode:@"YOUR_APP_CODE"
      licenseKey:@"YOUR_LICENSE_KEY"];
```

6. Build and run the application. If the build is successful, you now have an application that displays a map similar to the following screenshot and allows you to manipulate it using gestures.

Figure 6: Running the App



Chapter 3
User Guide

Topics:

The articles in this section provide a guide to using HERE iOS SDK.

System Requirements

- HERE SDK supports iOS 12 or above. iOS 12 is recommended for optimal operation.
- HERE iOS SDK is designed and tested for use with 64-bit iOS phones and tablets. Supported devices are:
  - iPhone 5s or newer
  - iPad Air or newer including iPad (2017) and iPad Pro
  - iPad Mini 2 or newer
- Apps should be developed using Xcode 11.3 or above running on macOS 10.15.2 or above.
- A minimum of 120MB per application should be made available for the storage of HERE SDK libraries.
- A minimum of 100MB should be made available for the storage of map data.
- Data connectivity (Wi-Fi or Cellular) is required to download map data and ensure map data is updated when new versions are made available.

Authenticating Applications

Developers using HERE SDK with their app are required to register for a set of HERE credentials and to specify these credentials (`App_Id`, `App_Code`, and `licenseKey`) in their application. Failure to do so results in blocked access to certain features and degradation in the quality of other services.

To obtain these credentials, visit the developer portal at *https://developer.here.com/plans* and register for a license. Once your project is created, you can generate these credentials on your Project Details page. If you already have a plan, you can also retrieve these credentials from your Project Details page.

Note: Credentials are unique to your application bundle identifier. Do not reuse credentials across multiple applications.

Important: When switching from an Evaluation plan to a commercial plan new HERE credentials must be taken into use. Applications **must not** be commercially released (such as submitted to a store) using a license key obtained as part of an Evaluation plan. Once you have upgraded to a commercial license, you need to obtain your new license key on the Project Details page, add it to your app, and re-deploy your app. See *Service Support* on page 11 to contact us for more details.

Adding Credentials

Ensure that you have provided the `app_id`, `app_code` and `licenseKey` before using HERE SDK. For example, set them in your app delegate:

```
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
 [NMAApplicationContext setAppId:@"{YOUR_APP_ID}"
 appCode:@"{YOUR_APP_CODE}"
 licenseKey:@"{YOUR_LICENSE_KEY}"];

 return YES;
}
```

Examples on GitHub

You can find more HERE SDK sample projects on GitHub: *https://www.github.com/heremaps*

HERE Map Data Download

Some key functionality offered through HERE SDK depends on HERE Map Data being downloaded and cached on the device. Rendering a map on the screen, for example, is not possible without first downloading map data to the device. Similarly, it would not be possible to provide accurate turn-by-turn navigation without downloading map data to the device. Offline operations such as offline routing and search also require map data to be downloaded in advance to the device. This section describes different approaches you can take to manage map data download.

Passive Download Approach

The passive approach is where you allow the SDK to download map data as needed. A typical example is when a user pans the map and triggers an on-demand map data download to render the map.

Map data downloaded in this way is stored in a persistent cache with a default size of 256 MB. Cached map data can be used for offline operations in cases where a network connection is not available or not desired, such as when the device is in roaming mode. However, there is no way for you to know if sufficient data have been downloaded to enable all offline operations, e.g. offline search or routing.

Note: To ensure the version of the map data retrieved by the passive approach is the latest version available, use the `NMAMapLoader` APIs to update to the latest version.

Note: Use `[NMAApplicationContext setDiskCacheSize:NSUInteger]` to change the default disk cache size.

Active Download Approach

HERE SDK provides two alternatives to actively fetch map data:

- Map data may be downloaded in the form of map packages for a predefined region or country.
- Map data may be downloaded for an arbitrary bounding box or a radius around a route.

The first active approach is where you request the download of map data packages which cover an entire country or region using the `NMAMapLoader` APIs. You do this by selecting from a list of map packages. A map package may be a state (such as California), region, or a country (such as Belgium).

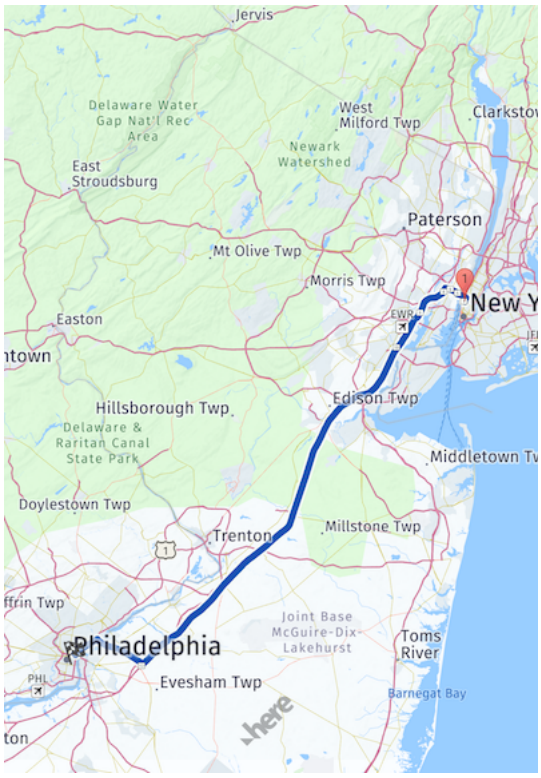
Note: This preloaded map data is stored separately from the map data cache mentioned in the passive download approach above. The amount of space available for map data packages is only limited by the amount of free space on the device.

The second active approach is where you explicitly trigger a fetch of map data through the `NMAMapDataPrefetcher` APIs by specifying a bounding box or a radius around a route. The resulting downloaded map data is stored in the same cache used in the passive download approach, where the cache size by default is 256 MB. The SDK does not place a limit on the size of the area requested for download but it is expected developers are aware of this cache limit and only request areas that result in map data

download with size under this limit. You can get a size estimate of the map data that will be downloaded through the `NMAMapDataPrefetcher` APIs.

To illustrate how much data may be downloaded, consider a bounding box covering an area of 200 km by 200 km in New York City as illustrated in the following screenshot. In this case approximately 250 MB of map data is downloaded.

The next example shows a 160 km route from New York to Philadelphia with a radius (route corridor width) of 500 m. The map data downloaded is about 100 MB.



A comparison of the different approaches for downloading map data is shown below:

Table 1: Map Data Download Approaches

	Passive Approach	Active Approach	
	On-demand	Map Packages	Bounding Box / Route
Downloaded map data can be used for offline operation	Limited (1)	Yes	Yes
Complexity involved in managing downloads	None	Medium	Low
Size of map data downloads	Medium (10s of MBs)	Large (100s of MBs)	Medium (10s of MBs)
Upper size limit of cache where data is stored (2)	256 MB - 2GB (2)	None (3)	256 MB - 2GB (2)
Option to check areas for which map data has previously been downloaded (4)	No	Yes	No
Option to check in advance the size of map data to be downloaded	No	Yes	Yes
Option to selectively remove downloaded map data	No (5)	Yes	No (5)
Can perform incremental updates between map data versions	Yes (6)	Yes (6)	Yes (6)

- (1): Map data downloaded on-demand may support some offline operations such as rendering and search while others, such as routing, do not work correctly as some essential data are missing.
- (2): The default value of disk cache upper limit is 256 MB. To change the upper limit of disk cache size use `[NMAApplicationContext setDiskCacheSize:NSUInteger]` method.

- (3): The number of map data packages which can be downloaded is only limited by the space available on the device.
- (4): For example, if you want to display to the user what map data has been downloaded and is available for offline use.
- (5): Only option is to completely clear the cache using `MapDataPrefetcher` APIs removing all map data downloaded on demand and by specifying a bounding box or route. Downloaded map packages are not removed.
- (6): Incremental updates are available when updating to the latest map data release from the two previous releases. Incremental updates are typically small downloads as only the changes are downloaded. For example, when updating to the Q1 2018 map data release from the Q4 2017 or Q3 2017 release, an incremental update or patch is used. Where a patch is not available (such as updating from Q2 2017 to Q1 2018), all map data packages are re-downloaded resulting in a much larger download size, and map data that was downloaded on-demand or by specifying a bounding box/route is removed.

Many applications may use a combination of all three approaches. For example, a UI may be provided to allow users to select map packages to download. Also, an option may be provided to download map data for a route prior to starting turn-by-turn navigation. Map data download on-demand would always be available as a fallback for the case when map data was not downloaded using the other approaches.

Note: In case of rerouting during turn-by-turn navigation at the very first time fast offline rerouting is performed. If it fails due to missing essential data, online request is sent to create a new route. For fast and reliable rerouting we recommend to have map data for a route downloaded prior to starting turn-by-turn navigation.

App Submission Requirements

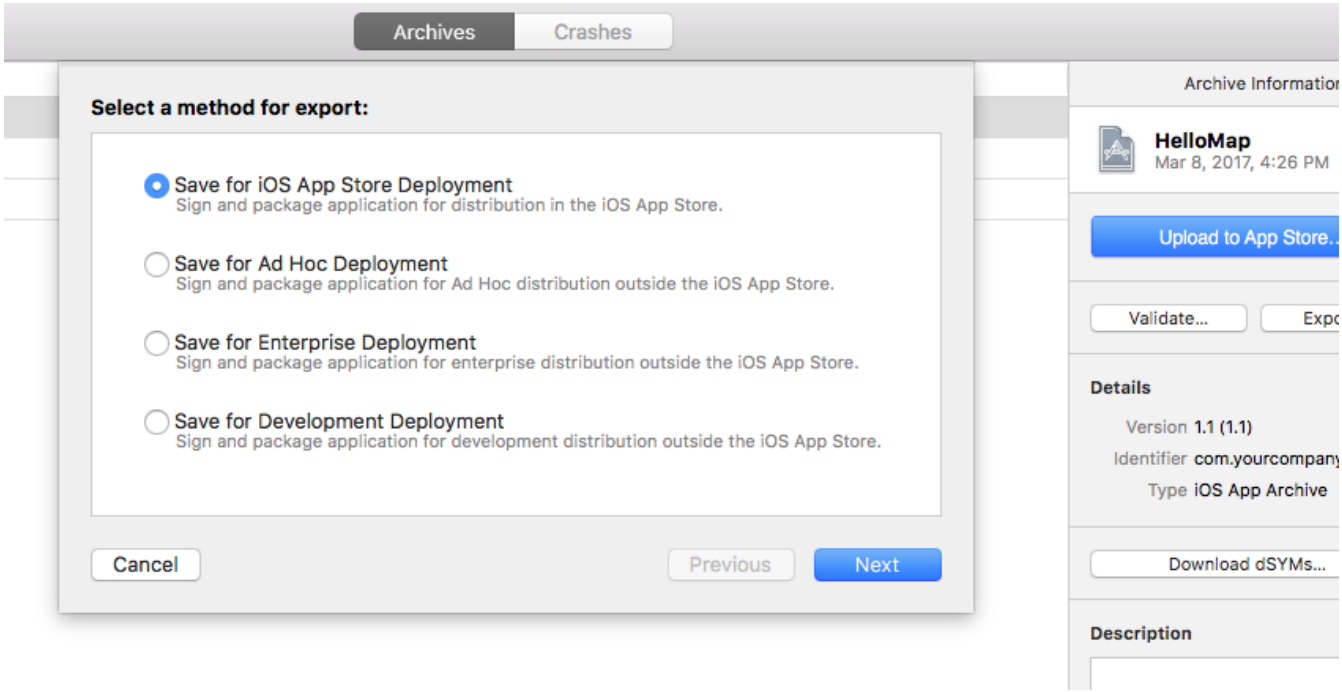
This section contains important information on how to prepare your HERE iOS SDK-enabled app for App Store submission. Be sure to follow the advice here to avoid store rejections caused by HERE SDK.

Remove Simulator Architectures

Apple rejects apps that are submitted to the App Store with simulator-specific architectures (`x86_64`, `i386`). Similarly, Xcode also gives an error if you try to export these apps for other types of deployment. Prior to

exporting your app you should strip simulator-specific architectures from the `NMAKit` framework binary and rebuild your app.

Figure 7: Exporting According to Deployment Type



A script is provided with HERE SDK to strip the simulator architectures from `NMAKit` library. Before you build your app to upload to the store, make a copy of `NMAKit.framework` and run the script to create a framework that only contains ARM device architectures (`arm64`, `armv7`). This modified version of the framework should be included in any builds destined for the app store.

This script is located at the following location:

```
{SDK Root}/framework/strip_sim.sh
```

To run the script, open a terminal, go to the directory containing the script, and execute it from there.

You can also add a script as part of your Xcode build process to automatically strip unnecessary architectures from frameworks. Depending on your build environment this may work better than managing separate versions of `NMAKit.framework`. For more details about this solution see *this article on Stack Overflow*.

Declare Private Data Access

Starting with iOS 10 you should statically declare access to private user data such as address book and device location before submitting your app to the App Store. Each declaration requires adding a key and a purpose string in your app `Info.plist` file.

The following shows the required HERE iOS SDK data access declarations. You can use the recommended `<string>` entries in your app `Info.plist`, or you can choose strings that are appropriate for your app.

Note: All HERE iOS SDK apps must have these declaration entries for the App Store submission process. App users are only prompted for the appropriate data access if your app uses the relevant HERE SDK feature.

```
<key>NSLocationWhenInUseUsageDescription</key>
<string>This is needed to determine your current location</string>
<key>NSCameraUsageDescription</key>
<string>This is needed for the LiveSight augmented reality feature</string>
<key>NSBluetoothPeripheralUsageDescription</key>
<string>This is needed for the Bluetooth-based indoor positioning feature</string>
<key>NSMotionUsageDescription</key>
<string>This is needed for the indoor positioning feature</string>
<key>NSLocationAlwaysUsageDescription</key>
<string>This is needed to determine your current location</string>
<key>NSBluetoothAlwaysUsageDescription</key>
<string>This is needed for the indoor positioning feature</string>
```

You can find more information about these keys in the following article: *https://developer.apple.com/library/content/documentation/General/Reference/InfoPlistKeyReference/Articles/CocoaKeys.html*.

Questions About Privacy Access Settings

HERE SDK contains references to the following methods:

- `[CLLocationManager startMonitoringForRegion:]`
- `[CLLocationManager startRangingBeaconsInRegion:]`
- `[CLLocationManager requestAlwaysAuthorization]`

However, these calls are only performed if your app uses *HERE Positioning* feature.

Due to the presence of these calls in HERE SDK framework, you may be asked additional questions about Location and iBeacons during App Store submission even if you are not using HERE Positioning. The following contains some information to help with your App Store submission process.

If your app does *not* use HERE Positioning:

- Q: Does this app detect `startMonitoringForRegion:`, `startRangingBeaconsInRegion:`, or both?

  A: These methods are referenced by HERE iOS SDK framework which is embedded in the app. However, due to the manner in which the App uses HERE iOS SDK these methods are never actually called.
- Q: What is the user experience when the app detects the presence of a beacon?

  A: Please see above, the beacons are not utilized by the app due to the manner in which the app uses HERE iOS SDK.
- Q: What features in this app use background location?

  If your app utilizes background navigation (for example, if your app calls `[NMANavigationManager backgroundNavigationEnabled]`), then you can answer "Navigation". Otherwise you can answer "Background location is not utilized due to the manner in which the app uses HERE iOS SDK".
- If this app uses 3rd party SDKs for iBeacons, please provide links to their documentation showing that background location is required for it to function.

  A: Please see above, iBeacons are not utilized due to the manner in which the app uses HERE iOS SDK.

If your app uses HERE Positioning:

- Q: Does this app detect `startMonitoringForRegion:`, `startRangingBeaconsInRegion:`, or both?

A: Both methods are used by embedded HERE iOS SDK framework to more accurately determine the user's current location indoors.

- Q: What is the user experience when the app detects the presence of a beacon?

  A: HERE iOS SDK uses the presence of beacons to provide accurate 3D location information indoors.

- Q: What features in this app use background location?

  A: Background location is used to accurately determine the user's current position indoors and also navigation if your app uses background navigation. (For example, if your app calls `[NMANavigationManager backgroundNavigationEnabled]`)

- Q: If this app uses 3rd party SDKs for iBeacons, please provide links to their documentation showing that background location is required for it to function.

  A: You can find HERE iOS SDK and its indoor positioning feature documentation at *https://developer.here.com/mobile-sdks/documentation/ios-hybrid-plus/topics/advanced-positioning.html*

App Transport Security (ATS) Exception

If your application uses HERE Positioning feature, you need to add an exception to your application `Info.plist` file. This exception key triggers additional review during the App Store submission process, and you are required to provide justification for including it.

Feature	Info.plist Entry	Explanation

HERE Positioning	`<key>NSAppTransportSecurity</key>`
	`<dict>`
	`  <key>NSExceptionDomains</key>`
	`  <dict>`
	`  <key>here.com</key>`
	`  <dict>`
	`  <key>NSExceptionAllowsInsecureHTTPLoads</key>`
	`  <true/>`
	`  <key>NSIncludesSubdomains</key>`
	`  <true/>`
	`  </dict>`
	`  </dict>`
	`</dict>`
		This key is required because HERE SDK currently downloads indoor radio maps via HTTP.

Mapping

Maps

The core feature of HERE iOS SDK is Mapping. The key concepts covered in this section include adding a map to an iOS application, changing the location displayed by the map, and modifying its properties. The primary component of the mapping API is the `NMAMapView`, which is integrated with the Cocoa Touch framework as a `UIView` subclass. `NMAMapView` represents the view to display map and various properties. The `NMAMapView` is derived from `UIView` and is part of iOS Cocoa Touch framework.

Note: To create a simple map application, refer to *Quick Start* section.

The first step to integrate a map into an application is to insert a `NMAMapView` to your view controller `.xib` file or the storyboard of the application using the Interface Builder. Alternatively, you can also add `NMAMapView` to your view controller programmatically as follows:

```
- (void)viewDidLoad
{
 mapView = [[NMAMapView alloc] initWithFrame:self.view.frame];
 [self.view addSubview:mapView];
}
```

The `NMAMapView` class handles all user interactions in the form of touch gestures. More details about the supported gesture types can be found in *Map Gestures* section.

Map Rendering Example on GitHub

You can find an example that demonstrates this feature at *https://github.com/heremaps/* (Obj-C) and *https://github.com/heremaps/* (Swift).

Working with NMAMapView

Once the `NMAMapView` is initialized, it can be manipulated and interacted in a variety of ways. Some key attributes of the `NMAMapView` are its orientation, tilt, geographical center (`geoCenter`), and zoom level (`zoomLevel`). These properties may be used to customize the `NMAMapView` display. For example, the following code demonstrates how to show a view of Vancouver, Canada.

```
NMAGeoCoordinates *geoCoordCenter = [[NMAGeoCoordinates alloc]
 initWithLatitude:49.260327 longitude:-123.115025];
[self.mapView setGeoCenter:geoCoordCenter withAnimation:NMAMapAnimationNone];
```

In the preceding code:

- The geographical location `[NMAGeoCoordinates]` for the new map center is created by a call to `-(id)initWithLatitude:(double)aLatitude longitude:(double)aLongitude` method.
- When setting the center of a map the transition can be animated by passing either `NMAMapAnimationLinear` or `NMAMapAnimationBow` enum value to `animation` parameter. Animation can also be suppressed by using `NMAMapAnimationNone` value.

The beginning and ending of these events may be observed by assigning an object to `NMAMapView` delegate property. The object should implement the methods of `NMAMapViewDelegate` protocol corresponding to the events you wish it to receive. This delegate can also be used to detect *map object selection*.

Note: For optimum performance avoid resizing a map after it has been created. If resizing is necessary, create the map at the largest size to be used and reduce it later.

Map Projection Mode

By default the map is set to a globe projection mode. You can change it to use *Mercator* projection by setting `projectionType` property. For example:

```
mapView.projectionType = NMAProjectionTypeMercator;
```

It may be useful to set the projection modes when you need to predictably display certain types of map information such as custom raster tiles.

Figure 8: Globe Projection

Figure 9: Mercator Projection

Map Snapshots

The `NMAMapView` class provides a class method and an instance method for creating map snapshots. These methods allow developers to create small static maps — such as thumbnails — that use the latest data from HERE SDK.

- ```
  +(void)snapshotWithGeoCoordinates:(NMAGeoCoordinates*)coordinates
        zoomLevel:(float)zoom
        orientation:(float)orientation
        size:(CGSize)size
        block:(void (^)(UIImage *snapshot))resultBlock;
  ```
- ```
  -(void)snapshotWithBlock:(void (^)(UIImage *snapshot))resultBlock;
  ```

For more information on using these methods consult the API reference.

Properties of NMAMapView

The following examples show how to work with some of the properties in `NMAMapView`:

Map Center

The center of the map determines the geographical area to be displayed. It can be read using the `NMAMapView geoCenter` property and set using one of `setGeoCenter:` methods. Its type is `NMAGeoCoordinates`.

```
// Move the map to London
NMAGeoCoordinates *geoCoordCenter = [[NMAGeoCoordinates alloc]
initWithLatitude:51.51 longitude:-0.11];
[self.mapView setGeoCenter:geoCoordCenter withAnimation:NMAMapAnimationNone];
```

Zoom Level

The size of geographical area displayed on the map can be controlled by changing zoom level. The zoom level ranges from `NMAMapView::minimumZoomLevel` to `NMAMapViewMaximumZoomLevel`, with a higher zoom value being closer to the ground. The following code sets the zoom level to the median zoom level:

```
// Set the zoom level to the median
```

```
mapView.zoomLevel =
(mapView.minimumZoomLevel + NMAMapViewMaximumZoomLevel)/2.0f;
```

Note: `NMAMapView::minimumZoomLevel` property provides the effective minimum zoom level depending on the current map and device display parameters such as display resolution. To retrieve the minimum zoom level that does not take these parameters into account, use `NMAMapViewMinimumZoomLevel`.

Orientation

The orientation determines which cardinal direction corresponds to which screen direction of the `NMAMapView`. The valid range is from 0 to 360 degrees. The default value of 0 orients the map so that true north is toward the top of the view. The following code shows how to set the orientation to south-up:

```
// Rotate by 180 degrees
mapView.orientation = 180;
```

Tilt

The tilt value represents the angle at which the map is viewed. By default the map tilt value is 0. This provides a top-down two-dimensional view of the map. You can set a tilt value using `tilt` property in `NMAMapView` but since the valid range of tilt values is dependent on the current zoom level, this `tilt` property is converted before it is used on screen. You can also use `maximumTiltProfile` property in `NMAMapView` to specify a block and define how zoom levels affect the maximum tilt.

To find the actual tilt value that is being used, call `clippedTilt` accessor method. To find the tilt limits for a given zoom level, call `minimumTiltAtZoomLevel:` and `maximumTiltAtZoomLevel:` methods.

```
// Set the tilt to 45 degrees
mapView.tilt = 45;
```

Animations

The `NMAMapView` supports the following animation settings to be used while changing properties, defined by `NMAMapAnimation` enum:

- `NMAMapAnimationNone`
- `NMAMapAnimationLinear`
- `NMAMapAnimationRocket`
- `NMAMapAnimationBow`

```
// Rotate by 90 degrees using a linear animation
[mapView setOrientation:90 withAnimation:NMAMapAnimationLinear];
```

```
// Move to London using bow animation
NMAGeoCoordinates *geoCoordCenter = [[NMAGeoCoordinates alloc]
  initWithLatitude:51.51 longitude:-0.11];
[mapView setGeoCenter:geoCoordCenter withAnimation:NMAMapAnimationLinear];
```

Setting Multiple Attributes

An extended API is provided to change one or more attributes at the same time.

```
-(void) setGeoCenter:(NMAGeoCoordinates*) coordinates
      zoomLevel:(float) level
       orientation:(float) orientation
        tilt:(float) tilt
```

```
      withAnimation:(NMAMapAnimation) animation
```

To leave a map attribute unchanged, pass `NMAMapViewPreserveValue` constant to the relevant method parameter.

```
// Move to Vancouver using bow animation, zoom level 17, 180
// degree orientation and preserve tilt
NMAGeoCoordinates* coord = [[NMAGeoCoordinates alloc]
 initWithLatitude:49.0
 longitude:123.0];
[mapView setGeoCenter:coord
 zoomLevel:17.0f
 orientation:180
 tilt:NMAMapViewPreserveValue
 withAnimation:NMAMapAnimationBow];
```

Map Event Blocks

Map event blocks provide a simple and versatile way to add custom handling for map events. With this mechanism an application can have arbitrary code executed when a specific map event occurs. Your application can define this code using `NMAMapEventBlock`. Each block can be registered to respond to one or more events, and it can dynamically control whether it continues to respond to a particular event type.

Note: The functionality of a map event block may overlap with `NMAMapViewDelegate` callbacks. In these situations it is recommended that you use `NMAMapEventBlock` to implement custom map event handling.

The types of event that can be observed are represented by `NMAMapEvent` enum and include:

- `geoCenter` has changed
- `zoomLevel` has changed
- `orientation` has changed
- `tilt` has changed
- A gesture has begun
- A gesture has ended
- An animation has begun
- An animation has ended
- Transformation has begun
- Transformation has ended

Transformation refers to any changes to the map geocenter, zoom level, orientation, and tilt value, regardless of whether the change was caused by the user or the application. There may be multiple transformations from different sources, including animations, gestures and others, between a single set of transformation begin and transformation end events. For example, if the application triggers a map animation and the map was in a stationary state, then `NMAMapEventTransformationBegan` and `NMAMapEventAnimationBegan` events occurs. If the user then performs a gesture before the animation completes, then `NMAMapEventAnimationEnded` and `NMAMapEventGestureBegan` events occurs. Finally, when the user finishes the gesture, `NMAMapEventGestureEnded` and `NMAMapEventTransformationEnded` events occurs.

To register a map event block, use `respondToEvents:withBlock:` method and set the events that you would like the block to respond to. For example, to hide an overlay whenever there is a map gesture or animation:

```
[_mapView respondToEvents:(NMAMapEventGestureBegan | NMAMapEventAnimationBegan)
 withBlock:^(NMAMapEvent event, NMAMapView *mapView, id eventData) {
 // _myOverlay is an informative overlay on the map
 _myOverlay.hidden = YES;
 return YES;
}];

[_mapView respondToEvents:(NMAMapEventGestureEnded | NMAMapEventAnimationEnded)
 withBlock:^(NMAMapEvent event, NMAMapView *mapView, id eventData) {
 // _myOverlay is an informative overlay on the map
 _myOverlay.hidden = NO;
 return YES;
}];
```

If the event block returns `YES`, the event block continues to run when the relevant event is triggered. If the event block returns `NO`, then it is removed for that particular triggering event. (If the same block is also registered for other event types, then it still responds to those events.) For example, the following code only performs a one-off action when the current animation finishes:

```
[_mapView respondToEvents:NMAMapEventAnimationEnded withBlock:^(NMAMapEvent event, NMAMapView
 *mapView, id eventData) {
 // Do something...
 return NO; // This block will be removed after being called once
}];
```

`respondToEvents:withBlock:` method returns a unique integer value to identify the registered block. To unregister the block, call `removeEventBlockWithIdentifier:` with this integer value.

Note: When creating event handler blocks care should be taken to avoid retain cycles. Any objects strongly referenced in the block are not released as long as the block remains. This includes any explicit or implicit (such as through an instance variable) references to `self`. Ensure that any strongly referenced objects inside the block are meant to live as long as the block does; if not, use a weak reference.

Map Display Layers

Map information such as street labels and park names is grouped into display layers. These layers are grouped as categories and represented by `NMAMapLayerCategory` enums. By default all layers are enabled to be displayed on the map view and you can use the following `NMAMapView` methods to control whether they should be hidden.

- `isMapLayerCategoryVisible:` - checks whether a category of map layers is visible
- `setVisibility:forMapLayerCategories:` - sets whether the specified categories of map layers should be visible
- `visibleMapLayerCategories` - retrieves an `NSArray` representing the currently visible categories of map layers

For a full list of map display layer categories see the API Reference for `NMAMapLayerCategory`.

Categories of Places

`NMAMapView` provides the following methods to show and hide categories of places on the map. Categories of places are represented by `NMAMapPoiCategory` enum.

- `poiCategories` - retrieves the names of all place categories
- `setVisibility:forPoiCategory:` - sets whether a category of places is visible
- `isPoiCategoryVisible:` - checks whether a category of places is visible

For example, you can hide all gas stations by calling the following:

```
[mapview setVisibility:NO forPoiCategory:NMAMapPoiCategoryPetrolStation];
```

Pedestrian Features

By default icons that indicate pedestrian access features (such as stairs or escalators) are not displayed on the `NMAMapView`. To display a pedestrian feature on the map view, call `showPedestrianFeature:pedestrianFeature` method with the desired `NMAMapPedestrianFeatureType`. Similarly, you can hide a pedestrian feature by calling `hidePedestrianFeature:pedestrianFeature` method. To find out whether a feature type is enabled, call `isPedestrianFeatureShown:pedestrianFeature` method.

Figure 10: Pedestrian Feature Icons

Interruption Handling

By default `NMAMapView` pauses map rendering when the application is no longer in the active state, e.g. while displaying the Control Center menu or while receiving a system notification. You can disable this behaviour by setting `pauseOnWillResignActive` property to `NO`. If automatic pausing is disabled, the application must manually pause the map view rendering when it goes to the background by setting `renderAllowed` property to `NO`.

Note: Rendering must be paused before returning from `applicationDidEnterBackground` method of `UIApplicationDelegate`.

For more information about the APIs introduced and demonstrated in this section refer to the API Reference documentation.

Map Gestures

The `NMAMapView` class responds to a number of predefined touch gestures. The default behavior of the map for each gesture type may be used as is, supplemented, or replaced entirely. The following table is a summary of the available gestures and their default behavior.

To select a visible map object, tap the screen with one finger.

To zoom the map in a fixed amount, tap the screen twice with one finger. Tap continuously to make a continuous zoom.

To zoom out a fixed amount, tap the screen with two fingers. Tap continuously to make a continuous zoom.

To move the map, press and hold one finger to the screen and move it in any direction.

To tilt the map, press and hold two fingers to the screen in a horizontal orientation, then move them in a vertical direction.

To rotate the map around the screen center, press and hold two fingers to the screen, then move them in a horizontal direction.

Note: If two-finger panning is used during a pinch or rotate gesture, the map pans instead of tilt or rotate.

To pan the map with momentum, press and swipe one finger on the screen. The map continues to move in the same direction and gradually slows down to a stop.

To continuously zoom in or out, press and hold two fingers to the screen and increase or decrease the distance between them.

To rotate the map, press and hold two fingers to the screen and rotate them together in a circle.

Pressing and holding one finger to the screen activates the long press gesture. This gesture does not have a predefined map action.

The time required to trigger a long press gesture can be customized using `NMAMapView longPressDuration` property. The default value for this property is one second.

Map Gestures Example on GitHub

You can find an example that demonstrates this feature at *https://github.com/heremaps/* (Obj-C) and *https://github.com/heremaps/* (Swift).

Controlling the NMAMapView Gesture Response

Any of the gestures listed above may be selectively enabled or disabled on an `NMAMapView` instance using `enableMapGestures:` and `disableMapGestures:` methods. These methods take a single input parameter that is an "or" combination of `NMAMapGestureType` values, which are defined in `NMAMapGesture.h`. The state of a specific gesture may be checked with `isMapGestureEnabled:`.

The following code shows how to disable all panning gestures:

```
// mapView is a valid NMAMapView instance
[mapView disableMapGestures:(NMAMapGestureTypePan | NMAMapGestureTypeTwoFingerPan)];
```

Gesture Delegation

For custom map gesture behaviors a gesture delegate may be installed on an instance of `NMAMapView`. The delegate is accessed via `gestureDelegate` property and must implement `NMAMapGestureDelegate` protocol. To replace the behavior of a given gesture, the corresponding handler method of the protocol must be implemented. Any or all of the default gesture behaviors may be replaced in this way. The following example shows how to replace the `NMAMapView` tap handling:

```
// mapView is an instance of NMAMapView
// myGestureDelegate is an instance of the custom gesture handling class
mapView.gestureDelegate = myGestureDelegate;
```

```
// In the MyGestureDelegate class definition:
@interface MyGestureDelegate : NSObject <NMAMapGestureDelegate>
```

```
@implementation MyGestureDelegate
// ...
-(void)mapView:(NMAMapView*)mapView didReceiveTapAtLocation:(CGPoint)location
{
 // Custom gesture behavior
 // Tap location is available through the "location" parameter
}
@end
```

If the default behavior is required in addition to the custom behavior for a given gesture type, the delegate handler method for that type should invoke the appropriate method in the `NMAMapView` default handler, `defaultGestureHandler`. E.g. the following example shows how to use the default tap gesture handling behavior from a custom tap-handling method:

```
-(void)mapView:(NMAMapView *)mapView didReceiveTapAtLocation:(CGPoint)location
{
    // Some custom behaviour...
    [mapView.defaultGestureHandler mapView:mapView didReceiveTapAtLocation:location];
    // More custom behaviour...
}
```

Any gesture types not handled by your custom gesture delegate also trigger the default behavior.

Note: Disabling the `NMAMapView` handling of a specific gesture type also disables the delegate handling of the same gesture type.

Gesture Delegation through Recognizers

Note: As of version 2.1 these types of handler methods are deprecated. It is recommended that you migrate to delegate handler methods mentioned in the previous section.

Another way to code custom gesture behaviors is to use callback methods with *UIGestureRecognizer*. To replace the behavior of a given gesture, set `NMAMapView usesGestureRecognizers` to `YES` and implement the corresponding `didReceive...FromRecognizer` handler method from the `NMAMapGesture` delegate protocol. Any of the default gesture behaviors may be replaced this way. When `usesGestureRecognizers` is enabled, the other type of gesture delegation (as mentioned above) is disabled.

Note: The map view (or any of its subviews) intercepts all touch events delivered to it when gestures are recognized using the `UIGestureRecognizer`.

The following example shows how to replace the `NMAMapView` tap handling:

```
@implementation MyGestureDelegate
// ...
-(void)mapView:(NMAMapView *)mapView didReceiveTapFromRecognizer:(UITapGestureRecognizer
 *)recognizer
{
 // Custom gesture behavior
 // Information about the gesture is accessed through the recognizer
}
@end
```

Subviews

The `NMAMapView` class is capable of passing the detected gestures to its subviews. For this to occur, three conditions must be met:

1. The gesture must originate inside the bounds of the subview
2. The subview must implement the `NMAMapGestureDelegate` protocol
3. The subview must implement the handler function for the type of gesture detected

Though subviews must implement the `NMAMapGestureDelegate` protocol to receive gestures, they do not need to be installed as the map view delegate. When a gesture is detected, the map view automatically

detects whether a subview capable of handling the gesture is present. If a suitable subview is found, the gesture is passed on and the map view takes no action.

As mentioned, the map view intercepts all touch events delivered to its children. Consequently, subviews that depend on touch events (such as `UIKit` classes) do not function properly. If such views are required on top of the map view, they should be placed in the view hierarchy as siblings of the map view rather than children.

Note: Disabling the `MapView` handling of specific gesture types does not stop the map view from receiving the corresponding touch events, detecting the gestures, and passing the gestures to subviews.

Map Schemes

Specific map schemes are available to offer your application users a choice among different kinds of map appearance.

Setting the Scheme

`NMAMapScheme.h` file defines schemes that HERE map service supports. You can set a desired scheme by changing `mapScheme` property of `NMAMapView`. For example:

```
mapView.mapScheme = NMAMapSchemeNormalDay;
```

Map Scheme Example on GitHub

You can find an example that demonstrates this feature at *https://github.com/heremaps/* (Obj-C) and *https://github.com/heremaps/* (Swift).

Examples of Map Schemes

All available schemes are defined as constant strings in `NMAMapScheme.h` file. The string values that your application can use to set a map scheme include:

Figure 11: NMAMapSchemeNormalDay

Figure 12: NMAMapSchemeSatelliteDay

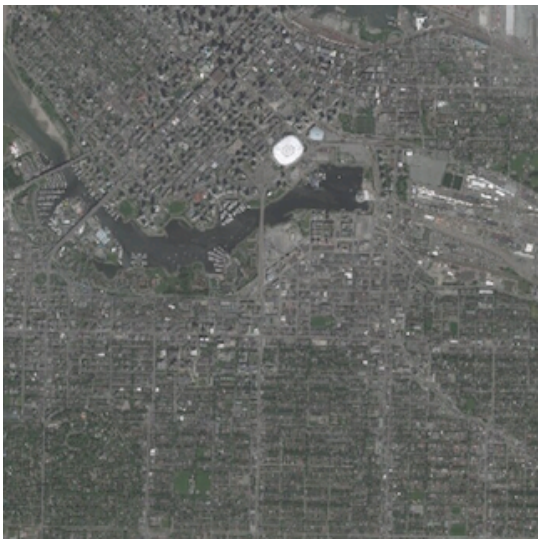


Figure 13: NMAMapSchemeHybridDay

Figure 14: NMAMapSchemeNormalNight

Note: Your application also needs to switch to one of the following schemes if you set traffic information as visible. These map schemes are otherwise identical to their non-traffic counterparts.

- `NMAMapSchemeNormalDayWithTraffic`
- `NMAMapSchemeNormalNightWithTraffic`
- `NMAMapSchemeHybridDayWithTraffic`
- `NMAMapSchemeHybridNightWithTraffic`
- `NMAMapSchemeCarNavigationDayWithTraffic`
- `NMAMapSchemeCarNavigationNightWithTraffic`
- `NMAMapSchemeHybridCarNavigationDayWithTraffic`

Note: In addition to the preceding schemes `NMAMapSchemeSatelliteNight` is also available. It is similar to `NMAMapSchemeSatelliteDay` but the color of the sky is different when the map is tilted.

Figure 15: NMAMapSchemeNormalDayTransit

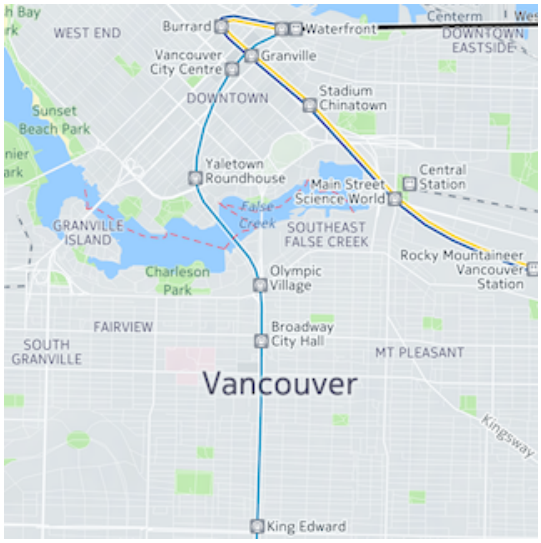


Figure 16: NMAMapSchemeNormalNightTransit

Figure 17: NMAMapSchemeTerrainDay

Figure 18: NMAMapSchemeHybridDayTransit

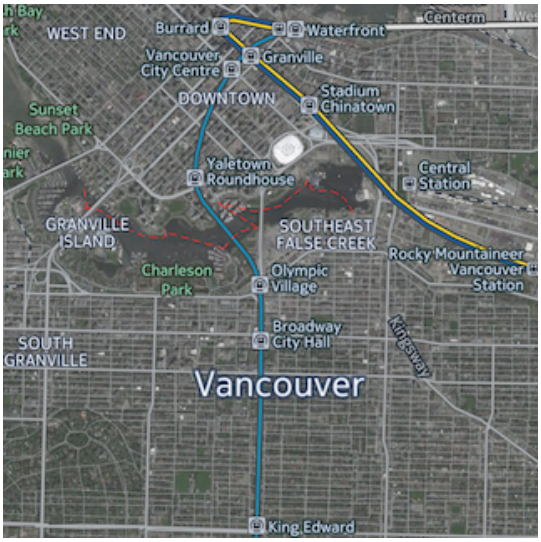


Figure 19: NMAMapSchemeReducedDay

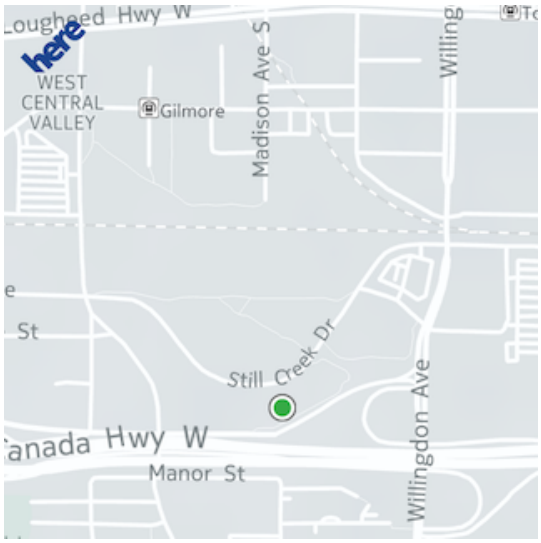


Figure 20: NMAMapSchemeReducedNight

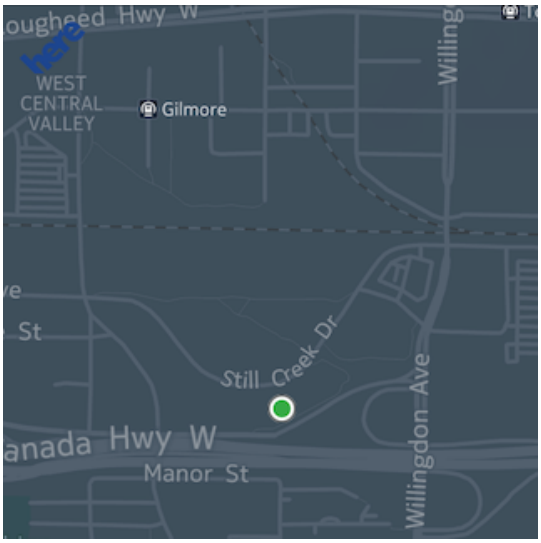
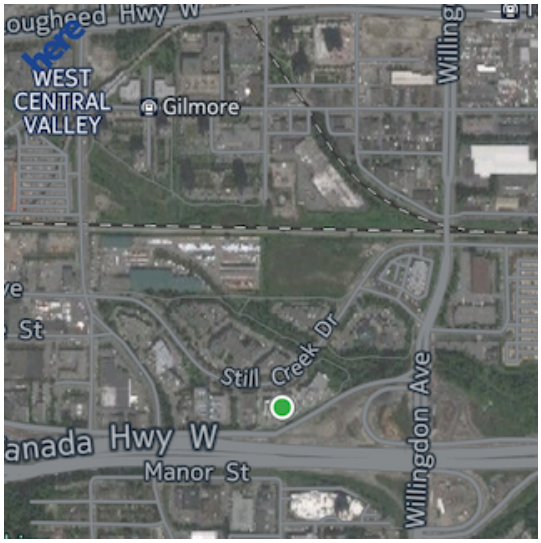


Figure 21: NMAMapSchemeHybridReducedDay



Navigation Schemes

HERE SDK also offers the following schemes to be used with navigation:

- `NMAMapSchemeCarNavigationDay`
- `NMAMapSchemeCarNavigationNight`
- `NMAMapSchemeHybridCarNavigationDay`
- `NMAMapSchemeHybridCarNavigationNight`
- `NMAMapSchemeCarNavigationDayWithTraffic`
- `NMAMapSchemeCarNavigationNightWithTraffic`
- `NMAMapSchemePedestrianDay`

- `NMAMapSchemePedestrianNight`
- `NMAMapSchemeHybridPedestrianDay`
- `NMAMapSchemeTruckNavigationDay`
- `NMAMapSchemeTruckNavigationNight`
- `NMAMapSchemeHybridTruckNavigationDay`
- `NMAMapSchemeHybridTruckNavigationNight`

If you are using a pedestrian navigation scheme, it is recommended that you also enable the pedestrian features in `NMAMapView`. See *Maps* on page 29 for more details.

Note: HERE SDK does not automatically switch map schemes during navigation mode. Before starting car or pedestrian navigation be sure to save the current map scheme and switch to the appropriate navigation map scheme. When navigation has completed, your application code should switch back to the previously saved scheme.

For more information on how to perform navigation operations see *Turn-by-Turn Navigation for Walking and Driving* on page 159.

For information on the fleet map scheme see *Mobile Asset Management* on page 69.

Objects and Interaction

HERE SDK allows the addition of a variety of objects, each with a specific purpose, to a map view. The types of available object include map markers, routes, polylines, and overlays. These objects are described in more detail below.

Map Objects Example on GitHub

You can find an example that demonstrates this feature at *https://github.com/heremaps/* (Obj-C) and *https://github.com/heremaps/* (Swift).

NMAMapObject Class

`NMAMapObject` class provides a generic base class from which most types of specialized map object are inherited. Functionality common to all these object types is encapsulated in `NMAMapObject`. The following is a list of important properties and methods in `NMAMapObject`.

- `zIndex` - determines the objects stacking order, which controls how the object is displayed on the map relative to other objects that may overlap it
- `visible` - determines whether or not the object is drawn when the map is rendered
- `type` - contains the type of map object such as marker, polyline, and route. For the full list see the `NMAMapObject` API reference
- `parent` - the `NMAMapContainer` instance holding this object, if any
- `mapLayerType` - `NMAMapLayerType` representing the display layer where this map object is rendered. By default map objects are assigned to the foreground
- `uniqueId` - uniquely identifies the object for the duration of application launch

Note:

- `NMAMapObject` serves as a base class to other map object types and should not be instantiated directly.

- Any change in a map object visual appearance causes the entire map view to be redrawn since map objects are drawn as part of the map itself. For optimal performance map objects should not be frequently updated unless it is necessary.
- If an `NMAMapObject` is assigned with an `NMAGeoCoordinates` object that contains a positive `altitude` value, then this map object is rendered as floating above the map. Since map objects are always rendered as facing the camera, it may not be obvious to the user that an object is floating until the map is tilted. When a map is tilted, map objects with high altitude values may become hard to locate. For the best results always set the `altitude` to `0` or a low positive value.

NMAMapContainer class

Map containers are a special type of map object that can be used to group together other map objects of certain types. The types of objects allowed are `NMAMapMarker`, `NMAMapCircle`, `NMAMapPolygon`, and `NMAMapPolyline`. Containers provide a convenient way to control the stacking order and visibility of a large group of objects.

To use a map container, create one or more map objects and add them to the container using `addMapObject` method. To show the objects on a map, add the map container to the map with `addMapObject` method of `NMAMapView`.

Note: A container may also hold other instances of `NMAMapContainer`.

NMAMapCircle class

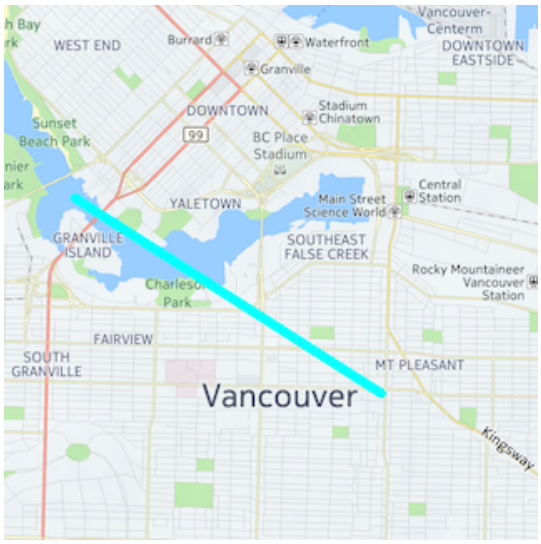
`NMAMapCircle` class is used to draw a circle on the map at a fixed geographical location; custom border and fill colors may be defined.

Figure 22: A `MapCircle` object

NMAMapPolyline class

`NMAMapPolyline` class is used to draw one or more connected line segments on the map. The segment vertices are specified by a series of `NMAGeoCoordinates`. The visual appearance of the polyline can be customized.

Figure 23: A `MapPolyline` object



NMAMapPolygon interface

`NMAMapPolygon` class is similar to `NMAMapPolyline` but the first and last points of the line are automatically joined to create a closed shape. Polygon objects can have different border and fill colors.

Figure 24: A `MapPolygon` object with transparent fill and a border

Note: Polygons that have a path that crosses over itself are not supported. For example, it is not possible to create a "bowtie" shape using four line segments where one line segment crosses another. However, you can create the shape using two triangles.

NMAMapMarker class

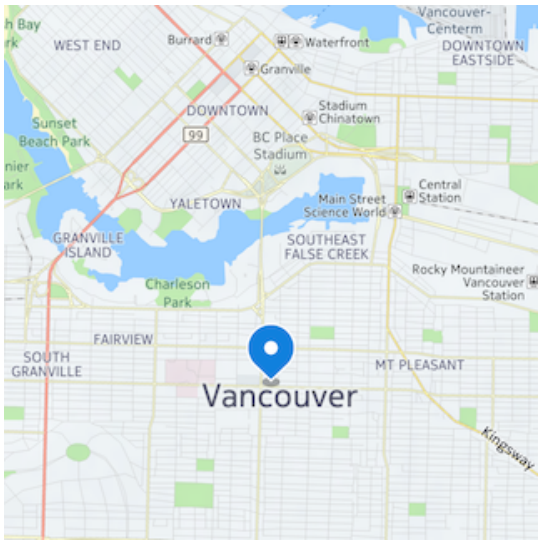
`NMAMapMarker` class is used to display a custom icon at a fixed geographical position on the map.

Custom icons can be one of the following file formats:

- BMP
- SVG
- JPEG
- PNG

Please see the `NMAImage` API Reference for more information on working with these file formats.

Figure 25: An `NMAMapMarker` object



You can set `NMAMapMarker` to be draggable by setting `draggable` property to `YES`. To listen for drag events, such as marker position changes, use `respondToEvents:withBlock:` method in `NMAMapView`.

NMAMapLocalModel

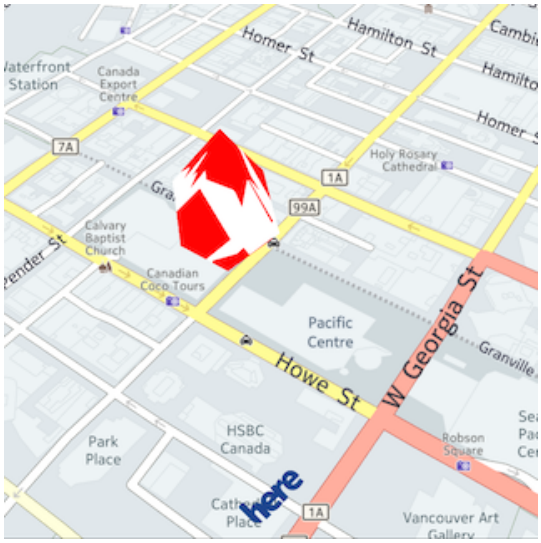
`NMAMapLocalModel` is an arbitrary 3D map object that is drawn using a local coordinate (as opposed to a geocoordinate) mesh. You can create a custom `NMAMapLocalModel` by setting the model mesh, texture, and geographical location before adding it to the map. For example:

```
NMAFloatMesh *mesh = [[NMAFloatMesh alloc] init];
float size = 40.f;
float vertices[6 * 3] = {0.0f,  0.0f,   0.0f,
    -size,  -size,  2 * size,
    size,   -size,  2 * size,
    size,   size,   2 * size,
    -size,  size,   2 * size,
    0.0f,   0.0f,   4 * size};
[mesh setVertices:vertices withCount:6];
float textureCoordinates[6 * 2] = {0.5,   0.5,
    0.5,   1,
    1,     0.5,
    0.5,   0,
    0,     0.5,
    0.5,   0.5};
```

```
[mesh setTextureCoordinates:textureCoordinates withCount:6];
short triangles[8 * 3] = {0, 2, 1,
    0, 3, 2,
    0, 4, 3,
    0, 1, 4,
    5, 1, 2,
    5, 2, 3,
    5, 3, 4,
    5, 4, 1};
[mesh setTriangles:triangles withCount:8];
NMAMapLocalModel* _diamond = [[NMAMapLocalModel alloc] initWithMesh:mesh];
NMAImage *image = [NMAImage imageWithUIImage:[UIImage imageNamed:@"flag.png"]];
[_diamond setTexture:image];
[_diamond setCoordinates:geoCoordCenter];
[self.mapView addMapObject:_diamond];
```

While translating the 3D model mesh to the map a unit of `1.0f` represents 1 meter in the real world. For example, a set of vertices in `{100, 200, 300}` represents an offset of +100 meters in the x-axis (East), +200 meters in the y-axis (North), and +300 meters in the z-axis direction (Up). You can further control the size of the 3D model mesh by setting a scaling factor with `scale` property.

Figure 26: An NMAMapLocalModel object



Aside from setting a texture, an `NMAMapLocalModel` can also be customized by adding a directional light. For example, the following code sets a light source to the `NMAMapLocalModel`.

```
NMAVector3d vec3d;
vec3d.x = 0.f; vec3d.y = 0.5f; vec3d.z = -1.f;

// This light shines from above in the Z axis
NMADirectionalLight* light = [NMADirectionalLight lightWithDirection:vec3d];
// assume 'model' is an NMAMapLocalModel that is already on the map
[model addLight:light];
```

Note:

- As 3D objects consume large amounts of memory, avoid using `NMAMapLocalModel` and `NMAMapGeoModel` to replace 2D map markers. Two examples of recommended uses of these classes are adding a few 3D structures to the map, or showing a realistic car model during guidance.
- If you use `NMAMapLocalModel` to create a two-dimensional object, and if you use an anchor with an undefined or zero altitude value, there is a known rendering issue with OpenGL where parts of

the object may conflict with the map layer causing the object to flicker. To get around this issue, use a z-coordinate offset that is greater than 0. For example, you can use a small floating point number such as `0.001`, so that the user is unable to distinguish between the object altitude and the map.

NMAMapGeoModel

`NMAMapGeoModel` is an arbitrary 3D map object that is drawn using geocoordinate vertices. You can create an `NMAMapGeoModel` by using an `NMAGeoMesh` and a texture `NMAImage`. For example:

```
NMAGeoMesh *geoMesh = [[NMAGeoMesh alloc] init];

double vertices[5 * 3] =
    {   -123.133392, 49.265275, 0,
        -123.127813, 49.265275, 0,
        -123.133392, 49.261803, 0,
        -123.127813, 49.261803, 0,
        -123.130903, 49.263931, 1250
    };
[geoMesh setVertices:vertices withCount:5];

float textureCoordinates[5 * 2] =
{   0.0, 0.0,
    1.0, 0.0,
    0.0, 1.0,
    1.0, 1.0,
    0.5, 0.5
};
[geoMesh setTextureCoordinates:textureCoordinates withCount:5];

short triangles[4 * 3] =
{
    0, 4, 1,
    2, 3, 4,
    0, 2, 4,
    1, 4, 3};

[geoMesh setTriangles:triangles withCount:4];

NMAMapGeoModel* _textureOverArea =
    [[NMAMapGeoModel alloc] initWithMesh:geoMesh];
NMAImage *image =
    [NMAImage imageWithUIImage:[UIImage imageNamed:@"flag.png"]];
[_textureOverArea setTexture:image];

[self.mapView addMapObject:_textureOverArea];
```

Note: As with `NMAMapLocalModel`, you can set the lighting for an `NMAMapGeoModel` using `addLight:` method.

Figure 27: An NMAMapGeoModel object

Map Object Selection

All user-defined objects with a visual representation can be selected. Selection occurs when a visible object on the map is tapped. By default the map does not take any action when objects are selected. To implement selection handling, a custom class must implement the `NMAMapViewDelegate` protocol and its `onMapObjectsSelected` method. An instance of the class must then be installed as a map view observer through `addMapViewObserver` method. `onMapObjectsSelected` callback returns an array that contains instances of `NMAViewObject`, which is a superclass of `NMAMapObject`.

Object selection can also be programmatically invoked by using `objectsAtPoint:` or `visibleObjectsAtPoint:` method. Each of these methods takes a `CGPoint` screen coordinate and returns an `NSArray` of `NMAMapObject` at that location. `visibleObjectsAtPoint` method does not return any object that has `visible` property set as `NO`.

For more information see the `NMAMapView` API documentation.

Note: In addition to user-defined objects, certain types of internal map objects are also selectable. See Proxy Objects section for more details.

NMAMapOverlay Class

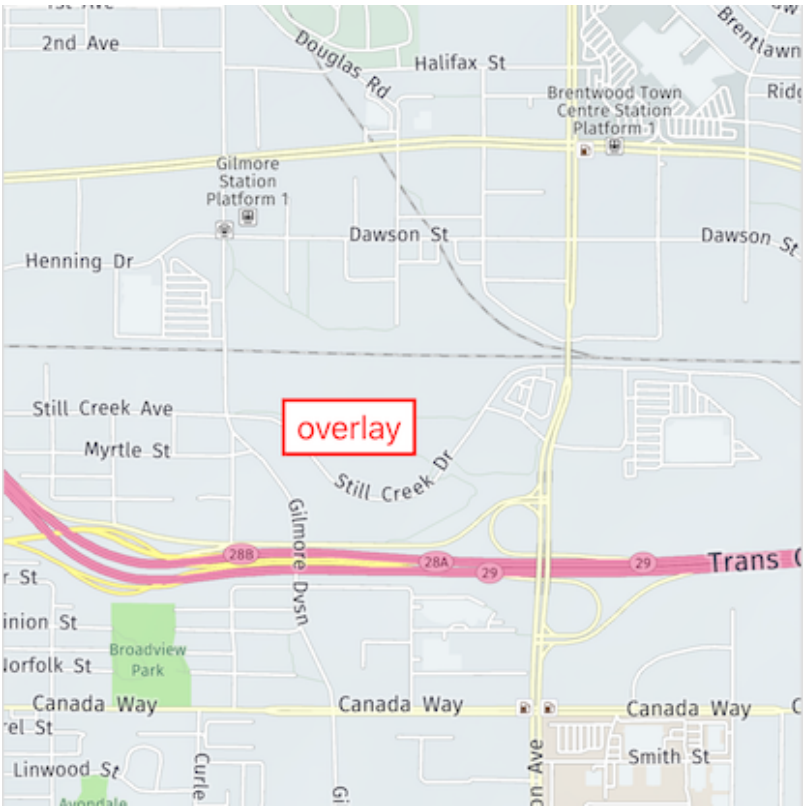
`NMAMapOverlay` class represents a special type of map object that does not inherit from `NMAMapObject` base class. Instead, it inherits from `UIView` class of Apple `UIKit` framework. Thus, it provides a way for any `UIView` or `UIView` subclass to be displayed at a fixed geographical location on the map.

For example, the following code shows how to use a `UILabel` on a map:

```
/*
 * 'location' is a CGPoint on screen
 * 'mapView' is an NMAMapView object
 */
NMAGeoCoordinates *coords = [mapView geoCoordinatesFromPoint:location];
```

```
UILabel *label = [[UILabel alloc] initWithFrame:CGRectZero];
label.textColor = [UIColor redColor];
label.text = @"overlay";
label.textAlignment = NSTextAlignmentCenter;
label.backgroundColor = [UIColor whiteColor];
label.layer.borderWidth = 2;
label.layer.borderColor = [UIColor redColor].CGColor;

label.frame = CGRectMake(0, 0, 70, 30);

NMAMapOverlay *overlay = [NMAMapOverlay mapOverlayWithSubview:label geoCoordinates:coords];

[mapView addMapOverlay:overlay];
```

Figure 28: Example of Map Overlay



Content can be added to a map overlay in the same manner as a normal `UIView`, with `addSubview` method. If complex view content is required, such as a view with subviews of its own, the content should be fully initialized before adding it to the map overlay.

Due to the extra performance cost of `UIViews` it is recommended that `NMAMapOverlay` only be used in situations where the additional functionality provided by `UIView`, such as animation, is needed. If the map object only needs to display a static image, `NMAMapMarker` should be used.

Note:

- `NMAMapOverlay` does not inherit from `NMAMapObject`, and thus overlays are not selected from a tap gesture by default. To achieve this behavior, the appropriate gesture handling must be implemented either in an `NMAMapOverlay` subclass, or in a custom view that is added as a subview to a standard `NMAMapOverlay`. For more information see *Map Gestures*.
- The geometry properties of `NMAMapOverlay` inherited from `UIView`, such as center and frame, should not be modified directly.

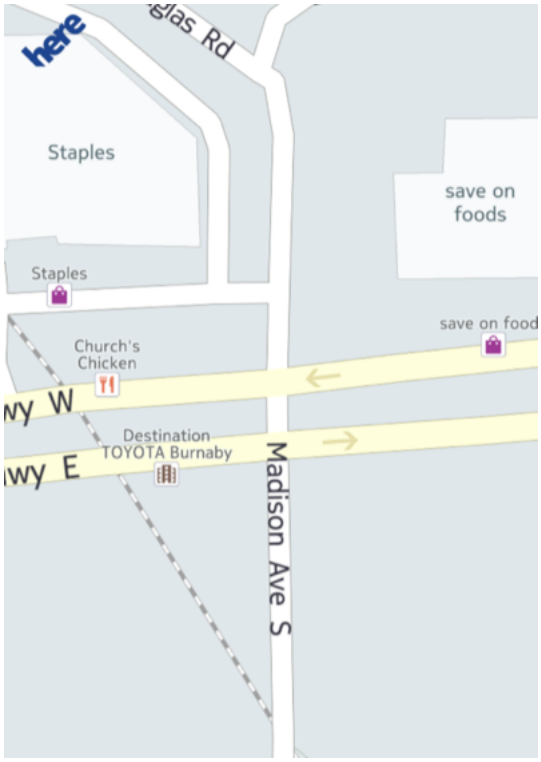
Proxy Objects

Some objects are added to the map by the system to represent real-world information such as traffic events, public transportation, or points of interest. These objects cannot be modified but many can be selected. All proxy objects are subclasses of `NMAProxyObject` which shares `NMAViewObject` class ancestor with `NMAMapObject`.

NMAPoiObject

Points of interest are represented by instances of `NMAPoiObject` proxy object class.

Figure 29: Examples of Points of Interest



In the above screenshot there are four points of interests: two shops, one restaurant, and one car dealership. Each of these points of interest may be selected by either tapping on the map, which returns the objects from `mapView:didSelectObjects:` callback method in `NMAMapViewDelegate`, or by calling `objectsAtPoint:` method in `NMAMapView`.

The following is an example of how to retrieve point of interest information from a tapped `NMAPoiObject`:

```
-(void)mapView:(NMAMapView *)mapView didSelectObjects:(NSArray *)objects
{
 for(NMAViewObject *object in objects) {
  if ([object isKindOfClass:[NMAPoiObject class]]){
   NMAPoiObject *poiObject = (NMAPoiObject*) object;
   NSString* placeName =
    [poiObject.locationInfo valueForField:NMALocationInfoFieldPlaceName];
   NSString* placeCategory =
    [poiObject.locationInfo valueForField:NMALocationInfoFieldPlaceCategory];
   NSString* placePhoneNUmber =
    [poiObject.locationInfo valueForField:NMALocationInfoFieldPlacePhoneNumber];
  }
```

```
  }
}
```

Marker Clustering

You can use marker clustering to reduce the visual overload caused by too many markers being displayed on the map at once. With this feature markers that are close together are automatically replaced by numbered cluster markers to indicate that multiple map markers are represented. For better user experience, the replacement algorithm depends on the device pixel density. Therefore you can observe different behavior on different devices.

Figure 30: Cluster Markers

Showing Cluster Markers

You can enable cluster markers on a map by using an `NMAClusterLayer` and adding map markers to it. All markers that are on a layer are automatically clustered based on a grid-based clustering algorithm that depends on the map zoom level.

The following steps demonstrate how to use `NMAClusterLayer` class:

1. Create map markers as usual:

```
NMAMapMarker *mm =
 [NMAMapMarker
  mapMarkerWithGeoCoordinates:[NMAGeoCoordinates geoCoordinatesWithLatitude:54.54
   longitude:13.23]
  image:[UIImage imageNamed:@"marker@1x.png"]];
```

2. Create a `ClusterLayer` object.

```
NMAClusterLayer *cl = [[NMAClusterLayer alloc] init];
```

3. Add markers to the cluster layer instead of the map directly.

```
[cl addMarker:mm];
```

You can also add an `NSArray` of `NMAMapMarker` instead of just adding a single marker by using `addMarkers:` method.

4. Add the cluster layer to the map.

```
[activeMapView addClusterLayer:cl];
```

Note: The order of these two steps is not important. You can also add the cluster layer to the map first and add markers to the cluster layer afterwards.

5. To remove a marker or collection or markers from the cluster layer again, call:

```
[cl removeMarker:mm];
```

You can also retrieve all markers on a cluster layer with `markers` property in `NMAClusterLayer`. This is useful in case if you would like to remove all markers by using `removeMarkers:` method.

Theming

You can customize clusters by assigning an `NMAClusterTheme` object to the `NMAClusterLayer`. Every theme consists of several styles, where a cluster style defines the look of marker cluster objects at a particular density. Cluster density is an indication of how many markers a cluster represents.

Figure 31: A Cluster with Density of 7

There are three available cluster styles that you can use with an `NMAClusterTheme`:

- Default cluster style - the predefined markers behavior. This is the default style used if you do not set a theme. It is also used for ranges that are not covered by your own theme.
- `NMABasicClusterStyle` - similar to the default style but you can change the fill color, text color, and stroke color for the markers.
- `NMAImageClusterStyle` - use your own bitmap image as a marker.

To set a style, use `setStyle:forDensityRange:` method in `NMAClusterTheme`. For example, if you want red for all clusters between density `10` to `19`, and green for `20` to `30`, and the default blue for all other cases, you can use `NMABasicClusterStyle` as follows:

1. Create a style with a red circle and a style with a green one:

```
NMABasicClusterStyle *redStyle = [NMABasicClusterStyle style];
redStyle.fillColor = [UIColor redColor];
NMABasicClusterStyle *greenStyle = [NMABasicClusterStyle style];
```

```
greenStyle.fillColor = [UIColor greenColor];
```

2. Create a new theme and add those styles to the theme along with defining the density ranges they should be used for:

```
NMAClusterTheme *theme = [[NMAClusterTheme alloc] init];
[theme setStyle:redStyle forDensityRange:NSMakeRange(10, 9)];
[theme setStyle:greenStyle forDensityRange:NSMakeRange(20, 10)];
```

   Note: Make sure that density ranges do not overlap, or otherwise `setStyle:forDensityRange:` returns `NO`.

3. Finally, add this theme to the cluster layer you use:

```
cl.theme = theme;
```

To use your own image as a cluster icon, set a `UIImage` to an `NMAImageClusterStyle` instance before setting the style to the cluster theme. For example:

```
NMAImageClusterStyle *imageStyle =
 [NMAImageClusterStyle
  styleWithUIImage:[UIImage imageNamed:@"cluster-icon.png"]];

NMAClusterTheme *theme = [[NMAClusterTheme alloc] init];
 [theme setStyle:imageStyle forDensityRange:NSMakeRange(2, 8)];
 cl.theme = theme;
```

Although you can only set one theme per layer, you can mix styles for different densities in a single theme. For example, you can set an `NMABasicClusterStyle` for density from `10` to `19` and an `NMAImageClusterStyle` from `20` to `30`. The default theme applies for all other densities that are not covered by the custom themes.

Cluster Marker Events

Cluster markers are similar to normal markers on the map. You can also use map object gesture delegates in similar way as normal map markers. For example:

1. Set an `NMAMapView` delegate:

```
self.mapView.delegate = self;
```

2. Implement `mapView:didSelectObjects:` to get the map click event.

```
- (void)mapView:(NMAMapView *)mapView didSelectObjects:(NSArray *)objects
{
 for (NMAClusterViewObject *object in objects) {
  if ([object isKindOfClass:[NMAClusterViewObject class]]) {
   NSLog(@"Cluster of %d markers", object.markers.count);
  }
 }
}
```

Working with Clusters

HERE SDK also provides a few other ways for you to interact with marker clusters. You can get all markers inside one specific cluster by using `NMAClusterViewObject`, which is a proxy object class representing a cluster. For example:

```
NMAGeoCoordinates *coordinates =
```

```
 [NMAGeoCoordinates geoCoordinatesWithLatitude:0 longitude:0]; // Your coordinates
NSArray *objects =
 [self.mapView objectsAtPoint:[self.mapView pointFromGeoCoordinates:coordinates]];

for (NMAClusterViewObject *object in objects) {
  if ([object isKindOfClass:[NMAClusterViewObject class]]) {
   NSArray *clusterMarkers = object.markers; // Your clustered markers
  }
 }
```

You can also retrieve the bounding box around all markers that are in a cluster marker by calling the following:

```
NMAClusterViewObject *cluster;
NMAGeoBoundingBox *boundingBox = cluster.boundingBox;
```

Traffic Information

HERE iOS SDK offers real-time traffic flow and congestion overlays. Traffic information can be displayed on the `NMAMapView` (where available) by setting its `trafficInfoVisible` property to `YES`. While HERE SDK requires network data connection to download real time traffic information, the visualization may continue to be displayed even if a connection is lost — until the traffic events expire, or the visibility is toggled.

Note: If you set `trafficInfoVisible` property to `YES` in `NMAMapView`, you need to switch the map view to one of the following schemes.

- `NMAMapSchemeNormalDayWithTraffic`
- `NMAMapSchemeNormalNightWithTraffic`
- `NMAMapSchemeHybridDayWithTraffic`
- `NMAMapSchemeHybridNightWithTraffic`
- `NMAMapSchemeCarNavigationDayWithTraffic`
- `NMAMapSchemeCarNavigationNightWithTraffic`
- `NMAMapSchemeHybridCarNavigationDayWithTraffic`

For more information on map schemes see *Map Schemes* on page 39

Traffic visualization is refreshed when one of the following happens:

1. The map is moved by a significant distance
2. The map is not moved for 1 minute. This duration can be set using `setRefreshInterval` in `NMATrafficManager`

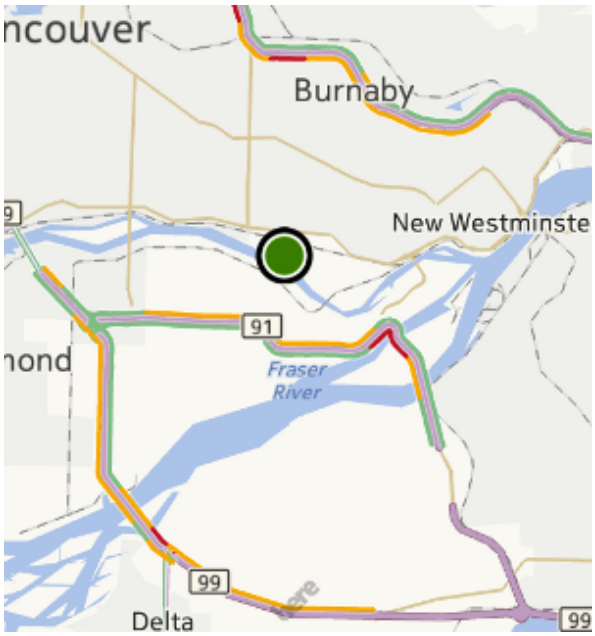
Traffic Flow

Traffic flow lines are color-coded as follows:

- Green - Normal
- Amber - High
- Red - Very High
- Black - Blocking

The following figure provides an example of traffic visualization:

Figure 32: Traffic information with color-coded lines



You can control the display of traffic flow lines via `trafficDisplayFilter` property on `NMAMapView`.

For example, you can set the map to only display traffic flow lines that are "very high" (red) or "blocking" (black) by performing the following:

```
// set the minimum displayed traffic level
mapView.trafficDisplayFilter = NMATrafficSeverityVeryHigh;
```

Traffic Flow Example on GitHub

You can find an example that demonstrates this feature at *https://github.com/heremaps/* (Obj-C) and *https://github.com/heremaps/* (Swift).

Traffic Incidents

The traffic information updates contain live traffic event information; these events are represented by various icons on the map. The following figures show examples of different types of traffic events:

Figure 33: NMATrafficObject example: Roadwork

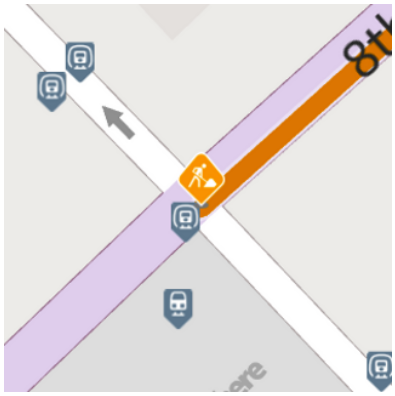


Figure 34: NMATrafficObject example: Accident

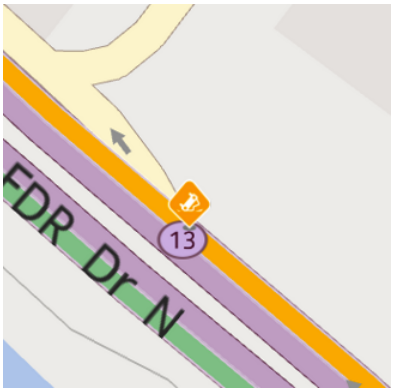
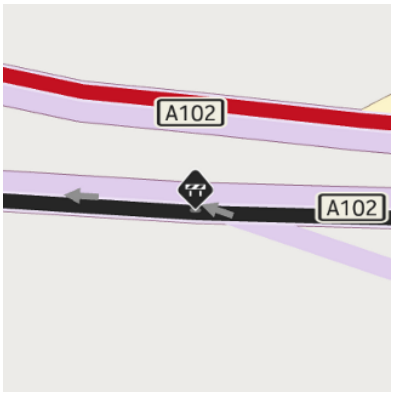


Figure 35: NMATrafficObject example: Road Closed



Traffic Objects and Events

Traffic events are represented on the map using instances of the `NMATrafficObject` *proxy object class*. These objects may be selected either by tapping on the map (which returns the objects from `-mapView:didSelectObjects:` method in the `NMAMapViewDelegate` protocol) or by calling `-objectsAtPoint:` method in `NMAMapView`. The underlying event is represented by an instance of `NMATrafficEvent`, which may be accessed via `trafficEvent` property in `NMATrafficObject`. The properties in `NMATrafficEvent` contain information about the event including type, description, and affected streets.

Preloading Map Data

Applications developed with HERE SDK have the ability to download map data. This enables the use of maps, search, routing, and other features without an active data connection. There are two approaches to download map data for offline use. These approaches are described in the next sections.

Note: Offline public transit routing requires map data covering the entire transit system to be downloaded.

Map Download by Specifying a Bounding Box or Route

Map data can be fetched into a persistent cache (with default size of 256 MB) to enable offline map capabilities by specifying a bounding box or radius around a route. This can be done through the `NMAMapDataPrefetcher` and associated classes. The `NMAMapDataPrefetcher` provides a set of methods to estimate the size of fetches (in KB), initiate fetches, cancel fetches, and clear the map data cache.

NMAMapDataPrefetcher and NMAMapDataPrefetcherListener

`NMAMapDataPrefetcher` is a singleton which must be obtained using `sharedMapDataPrefetcher` method. The next code examples show basic `NMAMapDataPrefetcher` use.

To check for the size of data in advance, use `estimateMapDataSizeForBoundingBox:error:` or `estimateMapDataSizeForRoute:radius:error:` methods:

```
NMAPrefetchRequestError error = NMAPrefetchRequestErrorNone;
NSInteger requestId =
 [[NMAMapDataPrefetcher sharedMapDataPrefetcher]
  estimateMapDataSizeForBoundingBox:bbox error:&error];
```

To start `NMAMapDataPrefetcher` fetch, use `fetchMapDataForBoundingBox:error:` or `fetchMapDataForRoute:radius:error:`.

```
NMAGeoBoundingBox* bbox = …
NMAPrefetchRequestError error = NMAPrefetchRequestErrorNone;
NSInteger requestId = [[NMAMapDataPrefetcher sharedMapDataPrefetcher]
 fetchMapDataForBoundingBox:bbox error:&error];
...
```

To cancel `NMAMapDataPrefetcher` fetch, call `cancel:` method.

```
NSInteger requestId = ...
[[NMAMapDataPrefetcher sharedMapDataPrefetcher] cancel:requestId];
...
```

`NMAMapDataPrefetcher` operations are performed asynchronously. When available, the results of these operations are passed on to `NMAMapDataPrefetcher` listeners. Listeners must implement the `NMAMapDataPrefetcherListener` protocol. The callbacks of `NMAMapDataPrefetcherListener` protocol are:

- `prefetcher:didUpdateProgress:forRequestId:` - Called during map data fetches to indicate the progress completed in the range of [0...1] for a request ID.
- `prefetcher:didUpdateStatus:forRequestId:` - Called during map data fetches to indicate status for a request ID.
- `prefetcher:didPurgeCache:` - Callback to indicate that the cache has been cleared of any unused map data.

NMAPrefetchStatus Enum

`NMAPrefetchStatus` is an enum data type that represents the status of `NMAMapDataPrefetcher` fetch operations. This status is returned through the `NMAMapDataPrefetcherListener` protocol.

- `NMAPrefetchStatusInProgress` - `NMAMapDataPrefetcher` fetch operation is still in progress
- `NMAPrefetchStatusSuccess` - `NMAMapDataPrefetcher` fetch completed successfully
- `NMAPrefetchStatusFailure` - `NMAMapDataPrefetcher` fetch operation failed to complete
- `NMAPrefetchStatusCancelled` - `NMAMapDataPrefetcher` fetch operation was cancelled

NMAPrefetchRequestError Enum

`NMAPrefetchRequestError` is an enum data type representing any errors encountered when initiating an `NMAMapDataPrefetcher` fetch operation.

- `NMAPrefetchRequestErrorNone` – No errors dispatching the request.

- `NMAPrefetchRequestErrorUnkown` – An unknown error has occurred during request submission.
- `NMAPrefetchRequestErrorBusy` – The number of requests is at max capacity.
- `NMAPrefetchRequestErrorInvalidParameters` – The request was invalid due to invalid parameters.
- `NMAPrefetchRequestErrorOperationNotAllowed` – ODML prefetching permission is missing.

Check the network connectivity at the application level to ensure `NMAMapDataPrefetcher` operations are only performed when the device has a connection.

Incremental Updates

Map data downloaded by specifying a bounding box or a route can be updated to the latest version using `NMAMapLoader` APIs. Map data version is consistent for all map data across the entire system, whether the map data is downloaded or not. It is not possible to have some data from one map version and some from another version simultaneously in the cache. Therefore, it is important to keep the map version of the system up to date.

You can perform incremental updates if you are updating to the latest map data release from the two previous releases. Incremental updates are typically small downloads as only the changes are downloaded. For example, when updating to the Q1 2018 map data release from the Q4 2017 or Q3 2017 release an incremental update or patch is used. Where a patch is not available (e.g. updating from Q2 2017 to Q1 2018), using the `NMAMapLoader` APIs to update to the latest version results in the removal of all downloaded map data, whether the data was downloaded on-demand or by specifying a bounding box/route.

Data Groups

Map packages are made up of several groups, each of which contains a different type of map data. Some of these groups may be selected or deselected before map packages are downloaded for offline use, depending on the needs of the application. The optional data groups are given in `NMAMapDataGroup` enum. To select or deselect a data group for download, pass the appropriate enum value to `NMAMapLoader selectDataGroup:` or `deselectDataGroup:` method. `isDataGroupSelected:` method can be used to query the current selection state of a data group.

Map Package Download

The second method of getting offline maps capabilities is enabled through the use of `NMAMapLoader` and its associated objects. The `NMAMapLoader` class provides a set of APIs that allow manipulation of the map data stored on the device. Operations include:

- `getMapPackages` - To retrieve the current state of the map data on the device through a delegate callback. After this asynchronous operation is complete, `rootPackage` property is also populated with the root item of package hierarchy
- `getMapPackageAtGeoCoordinates:` - To find the smallest `NMAMapPackage` containing a given set of geocoordinates
- `installMapPackages:` - To request the specified list of `NMAMapPackage` to be installed
- `uninstallMapPackages:` - To request the specified list of `NMAMapPackage` to be uninstalled
- `checkForMapDataUpdate` - To check whether a new map data version is available
- `getMapDataUpdateSize` - To get the size of a pending map data version update
- `performMapDataUpdate` - To perform a map data version update, if available
- `cancelCurrentOperation` - To cancel the running `MapLoader` operation

NMAMapLoader is a singleton which must be obtained using sharedMapLoader method. The following code example shows basic NMAMapLoader use:

- Initiate NMAMaploader

```
- (void) initMapLoader
{
 NMAMapLoader *mapLoader = [NMAMapLoader sharedMapLoader];
 mapLoader.delegate = self;
 ...
 [[NMAMapLoader sharedMapLoader] getMapPackages];
 ...
 }
```

- Install a map package

```
- (void) startInstall
{
 ...
 NSArray *packageArray = [[NSArray alloc]
  initWithObjects:packageToModify, nil];
 [[NMAMapLoader sharedMapLoader]
  installMapPackages:packageArray];
 ...
}
```

NMAMapLoader operations are performed asynchronously. When available, the results of these operations are passed on to the map loader delegate. The delegate is accessed via NMAMapLoader delegate property, and it must implement the NMAMapLoaderDelegate protocol. The callbacks of NMAMapLoaderDelegate protocol are:

- mapLoader:didUpdateProgress: - Called during NMAMapLoader operations to indicate the current progress of that operation
- mapLoader:didInstallPackagesWithResult: - Callback associated with NMAMapLoader installMapPackages: method call
- mapLoader:didUninstallPackagesWithResult: - Callback associated with NMAMapLoader uninstallMapPackages: method call
- mapLoader:didFindUpdate:fromVersion:toVersion:withResult: - Callback associated with NMAMapLoader checkForMapDataUpdate method call
- mapLoader:didGetUpdateSize:withResult: - Callback associated with NMAMapLoader getMapDataUpdateSize method call
- mapLoader:didUpdateWithResult: - Callback associated with NMAMapLoader performMapDataUpdate method call
- mapLoader:didGetMapPackages:withResult: - Callback associated with NMAMapLoader getMapPackages method call
- mapLoader:didGetMapPackage:atGeoCoordinates:withResult: - Callback associated with NMAMapLoader getMapPackageAtGeoCoordinates: method call
- mapLoaderDidLoseConnection: - Callback sent when the NMAMapLoader loses its server connection during an operation
- mapLoaderDidFindConnection: - Callback sent when the NMAMapLoader reestablishes its connection

Note: If the app is switched to the background while NMAMapLoader is downloading a map package, the download continues to run in the background for a maximum of three minutes. If the download was not completed within this time, then the download resumes when the app becomes active again.

Note: If a map loader operation is interrupted due to a connection loss, it automatically resumes when the connection is restored (assuming that the operation is not explicitly cancelled during this time).

Map Loader Example on GitHub

You can find an example that demonstrates this feature at *https://github.com/heremaps/* (Obj-C) and *https://github.com/heremaps/* (Swift).

NMAMapLoaderResult Enum

`NMAMapLoaderResult` is an enum data type representing the status for `NMAMapLoader` operations. The status is returned through the `NMAMapLoaderDelegate` protocol.

- `NMAMapLoaderResultSuccess` - `NMAMapLoader` operation was completed without errors.
- `NMAMapLoaderResultInvalidParameters` - `NMAMapLoader` operation was called with invalid parameters.
- `NMAMapLoaderResultOperationCancelled` - `NMAMapLoader` operation was cancelled by a call to cancelCurrentOperation.
- `NMAMapLoaderResultConnectionFailed` - a connection to the map data server cannot be made.
- `NMAMapLoaderResultInitializationFailed` - the `NMAMapLoader` could not be initialized.

It is recommended to check the network connectivity at the application level to ensure `NMAMapLoader` operations are only performed when the device has a connection.

NMAMapPackage Interface

The map data packages available for download are represented as a tree structure with the root map package representing the world map. The `NMAMapPackage` interface represents the model through which this tree structure is accessed. The properties of class `NMAMapPackage` are:

- `parent`
- `children`
- `packageId`
- `sizeOnDisk`
- `installed`

The `installed` state of a particular `NMAMapPackage` instance is not updated dynamically to reflect changes to map data on disk. For example, if you retrieve an `NMAMapPackage` instance with `getMapPackages` and then call `installMapPackages` method with this package as the parameter, this package state remains the same even when `didInstallPackagesWithResult` callback occurs. Instead, if you want to retrieve an `NMAMapPackage` with the latest install state, access `NMAMapLoader rootPackage` property. The `rootPackage` property is updated with a new object instance after every install operation.

Incremental Updates

MapLoader exposes the ability to update the map data version to provide the user with the freshest map data available. The map data version applies not only to map data pre-installed using the MapLoader but also to data downloaded on-demand.

Map data version is consistent for all map data across the entire system, whether the map data is downloaded or not. It is not possible to have some data from one map version and other data from another map version concurrent in the disk cache. Therefore, it is important to keep the map version of the system up to date.

You can perform incremental updates if you are updating to the latest map data release from the two previous releases. Incremental updates are typically small downloads as only the changes are downloaded. For example, when updating to the Q1 2018 map data release from the Q4 2017 or Q3 2017 release, an incremental update or patch is used. Where a patch is not available (such as updating from Q2 2017 to Q1 2018), all map data packages are re-downloaded resulting in a much larger download size. Map version updating is exposed through `checkForMapDataUpdate` and `performMapDataUpdate` methods of `NMAMapLoader`.

Data Groups

Map packages are made of several groups each of which contains a different type of map data. Some of these groups may be selected or deselected before map packages are downloaded for offline use depending on the needs of the application. The optional data groups are given in `NMAMapDataGroup` enum. To select or deselect a data group for download, pass the appropriate enum value to `NMAMapLoader selectDataGroup:` or `deselectDataGroup:` method. `isDataGroupSelected:` method can be used to query the current selection state of a data group.

The selected data groups of map loader are used for all future installation operations. However, changes to the data group selection do not affect previously installed packages. To update these packages, call `performMapDataUpdate` after changing the data group selection.

Note: The default data group selection may not be optimal for some applications. To minimize disk space usage, it's recommended that any applications which allow offline map downloads ensure they are only downloading the required data groups.

Custom Raster Tiles

You can use HERE iOS SDK to enhance maps with the custom raster tiles API — `NMAMapTileLayer`.

Custom raster tiles are tile images that you can add to a map for enhancing the map with extra information over a large geographical area. If the application is set to display custom raster tiles, then users see them whenever they view a designated geographical area at a specified zoom level or range of zoom levels.

You can provide tile images in two ways:

1. Store custom raster tile images on a remote server and return URLs via `NMAMapTileLayerDataSource mapTileLayer:urlForTileAtX:y:zoomLevel:` protocol method.
2. Provide raw bitmap data using `NMAMapTileLayerDataSource mapTileLayer:requestDataForTileAtX:y:zoomLevel:tileRequest:` protocol method.

Note: Before enabling raster tiles ensure that the extruded buildings feature is disabled as the appearance of both features does not go well together.

Map Raster Tile Example on GitHub

You can find an example that demonstrates this feature at *https://github.com/heremaps/* (Obj-C) and *https://github.com/heremaps/* (Swift).

Dividing a Map and Using Tile Coordinates

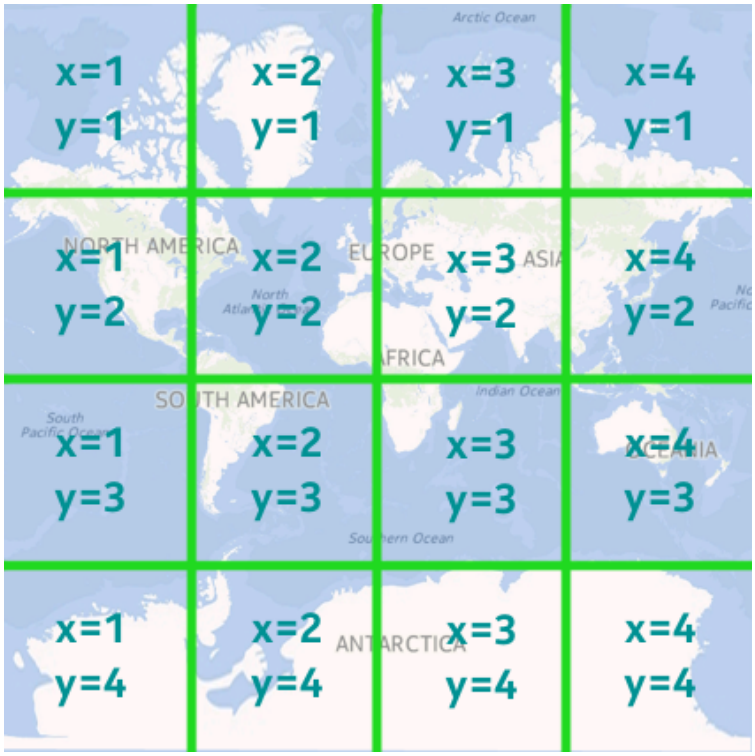
`NMAMapTileLayer` uses a scheme that divides the world map into tiles specified by x, y, and zoom level coordinates. This coordinate system is used by the `NMAMapTileLayerDataSource` protocol when it requests tiles.

At each zoom level it is expected that the world map is rendered on $(2^{zoomlevel})^2$ tiles:

- at level 0: 1 x 1 = 1 tile
- at level 1: 2 x 2 = 4 tiles
- at level 2: 4 x 4 = 16 tiles
- at level 3: 8 x 8 = 64 tiles
- at level 4: 16 x 16 = 256 tiles
- continuing on until zoom level 20

For example, at zoom level 2 the world map would be divided up as follows:

Figure 36: World Map at Zoom Level 2



The x and y parameters indicate which tile is being requested for the given zoom level.

You need to provide enough tile images to cover all the zoom levels you are supporting within a geographical area. You can restrict custom tile rendering to a specific `NMAGeoBoundingBox` using `boundingBox` property of `NMAMapTileLayer`. You can restrict the zoom level using `showAtZoomLevel:` and related methods.

Supplying Tiles from a Web Server

The steps for providing custom tiles from a web server to a map view are as follows:

1. Host the appropriate number of tiles on a server according to the zoom levels and `NMAGeoBoundingBox` you are supporting. The tiles must be in either PNG or JPG format and should be sized at 256 x 256 pixels (this is the default size but can be changed).
2. Create an object that derives from `NMAMapTileLayerDataSource` and implement `mapTileLayer:urlForTileAtX:y:zoomLevel:method` to return a URL pointing to the specified tile on your server.
3. Create an `NMAMapTileLayer` object and set its properties to correspond to the tile data source server. At least, set `boundingBox` and `zoomLevel` properties to reflect the tiles hosted on your server. Set `dataSource` property.
4. Optionally, customize the appearance of the tiles within the map view by setting properties such as `fadingEnabled`.
5. Add the `NMAMapTileLayer` object to the `NMAMapView` by calling `addMapTileLayer:` method.

The following code snippet shows a class that renders the Queen Elizabeth Olympic Park in London. The park is displayed as a tile layer that is added to an `NMAMapView`, and the raster tiles are served from HERE server. To use this class, call `[OlympicParkTileLayer addOlympicParkTileLayerToMapView:myMapView]`.

```
@interface OlympicParkTileLayer : NMAMapTileLayer <NMAMapTileLayerDataSource>
@end

@implementation OlympicParkTileLayer

+(void)addOlympicParkTileLayerToMapView:(NMAMapView*)mapView
{
 OlympicParkTileLayer *tileLayer = [OlympicParkTileLayer new];
 [mapView addMapTileLayer:tileLayer];
 [mapView setGeoCenter:tileLayer.boundingBox.center
   zoomLevel:14.0
   orientation:NMAMapViewPreserveValue
   tilt:NMAMapViewPreserveValue
   withAnimation:NMAMapAnimationNone ];
}

-(id)init
{
 if (self = [super init]) {
  // Set the data source
  self.dataSource = self;

  // Set the bitmap format properties (must be compatible with pngs hosted on the
  // server)
  self.pixelFormat = NMAPixelFormatRGBA;
  self.transparent = NO;

  // Limit the tiles to the bounding box supported by the server
  NMAGeoBoundingBox *olympicParkBoundingBox =
  [NMAGeoBoundingBox geoBoundingBoxWithTopLeft:
    [NMAGeoCoordinates geoCoordinatesWithLatitude:51.557000 longitude:-0.042772]
   bottomRight:
   [NMAGeoCoordinates geoCoordinatesWithLatitude:51.525941 longitude: 0.028296]];
  self.boundingBox = olympicParkBoundingBox;

  // Customize the tile layer
  self.fadingEnabled = YES;
  // covers everything else on the map
  self.mapLayerType = NMAMapLayerTypeForeground;
```

```
  // Enable caching
  self.cacheTimeToLive = 60 * 60 * 24;    // 24 hours
  self.cacheSizeLimit = 1024 * 1024 * 64; // 64MB
  [self setCacheEnabled:YES withIdentifier:@"OlympicParkTileLayer"];
 }
 return self;
}

-(NSString *)mapTileLayer:(NMAMapTileLayer *)mapTileLayer
   urlForTileAtX:(NSUInteger)x
       y:(NSUInteger)y
    zoomLevel:(NSUInteger)zoomLevel
{
 // Return a URL for the specified tile
 // This tile source is hosted by HERE Global B.V. and may be removed at any time
 return [NSString stringWithFormat:
  @"http://api.maps.example.org/maptiles/olympic_park/normal.day/%d/%d/%d.png",
  zoomLevel,
  y,
  x ];
}

@end
```

Supplying Tiles as Bitmaps

You can choose to supply tiles as bitmaps if your app uses bundled static tiles, dynamically generated tiles, or if the server that you are using for tile images requires authentication. In the third case, since `NMAMapTileLayer` only uses simple HTTP GET requests to retrieve tile images, it is up to you to provide code that handles authentication and downloads the tiles. Once the tiles have been downloaded, you can use them as local bitmaps with the `NMAMapTileLayer` class.

The steps for providing custom tiles as local bitmaps to a map view are as follows:

1. Create an object that derives from `NMAMapTileLayerDataSource` and implements `mapTileLayer:requestDataForTileAtX:y:zoomLevel:tileRequest:` method. Retain and use `NMAMapTileRequest` parameter to supply the requested bitmap data.
2. Create an `NMAMapTileLayer` object and set its properties to correspond to the tile data source. Set `dataSource` property.
3. Optionally, customize the appearance of the tiles within the map view by setting properties such as `fadingEnabled`.
4. Add the `NMAMapTileLayer` object to the `NMAMapView` by calling `addMapTileLayer:` method.

The following is an example of a class that renders red and green tiles on an `NMAMapView` using a tile layer. It can be used by calling `[SimpleBitmapTileLayer addSimpleBitmapTileLayerToMapView]`.

```
@interface SimpleBitmapTileLayer : NMAMapTileLayer <NMAMapTileLayerDataSource>
{
 char *_redBitmap;
 char *_greenBitmap;
}
@end

@implementation SimpleBitmapTileLayer

+(void)addSimpleBitmapTileLayerToMapView:(NMAMapView*)mapView
{
```

```
 [mapView addMapTileLayer:[SimpleBitmapTileLayer new]];
}

-(id)init
{
 if (self = [super init]) {

  // Set the data source
  self.dataSource = self;

  // Set the bitmap properties (must match what is returned in
  // requestDataForTileAtX:)
  self.pixelFormat = NMAPixelFormatRGBA;
  self.transparent = NO;

  // The tiles will replace the map background
  self.mapLayerType = NMAMapLayerTypeArea;

  // We want the tiles to span the globe so don't limit with a bounding box or
  // zoom level.

  // We don't want to cache these tiles to disk, as in this simple case
  // it's more efficient to keep bitmaps in memory.

  // Create a red bitmap
  _redBitmap = (char *)malloc(256*256*4);
  for (int i=0; i<256*256*4; i+=4) {
   _redBitmap[i] = 0xFF;
   _redBitmap[i+1] = 0x00;
   _redBitmap[i+2] = 0x00;
   _redBitmap[i+3] = 0xFF;
  }

  // Create a green bitmap
  _greenBitmap = (char *)malloc(256*256*4);
  for (int i=0; i<256*256*4; i+=4) {
   _greenBitmap[i] = 0x00;
   _greenBitmap[i+1] = 0xFF;
   _greenBitmap[i+2] = 0x00;
   _greenBitmap[i+3] = 0xFF;
  }
 }

 return self;
}

-(void)dealloc
{
 free(_redBitmap);
 free(_greenBitmap);
}

-(void)     mapTileLayer:(NMAMapTileLayer *)mapTileLayer
   requestDataForTileAtX:(NSUInteger)x
                       y:(NSUInteger)y
               zoomLevel:(NSUInteger)zoomLevel
    tileRequest:(NMAMapTileRequest *)tileRequest
{
 // Dispatch to a global queue to demonstrate retaining the tile request and
 // completing asynchronously. In this case it is not necessary but if obtaining
 // the bitmap data was a lengthy operation we would not want to block the current
 // (map rendering) thread.
 dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0), ^{
  // Use alternating red and green bitmaps to create checkerboard effect
  void *bitmap = ( (x%2==0 && y%2==1) || (x%2 ==1 && y%2==0) ) ? _redBitmap :
    _greenBitmap;
```

```
    // Copy the bitmap into the bitmap buffer supplied by the tile request
    // Copy exactly tileRequest.bytesLength
    memcpy(tileRequest.bytesPtr, bitmap, tileRequest.bytesLength);

    // Complete the tile request AFTER the copy has completed
    tileRequest.status = NMAMapTileRequestStatusComplete;
  });
}

@end
```

There are a couple of important points to consider when using the `NMAMapTileRequest` object:

1. The bitmap data must be in the format specified by `NMAMapTileLayer pixelFormat` property. The default is `NMAPixelFormatRGBA`.
2. Copy the bitmap bytes before you complete the request and only copy the number of bytes specified by `bytesLength` to avoid memory corruption.
3. Always complete the request precisely once with either `NMAMapTileRequestStatusComplete` or `NMAMapTileRequestStatusFailed`. Uncompleted requests occupy memory until they are completed.
4. Do not block `mapTileLayer:requestDataForTileAtX:y:zoomLevel:tileRequest:` for an extended period of time as this impacts rendering performance.

Changing the Overlay Rendering Order

You can choose to customize the map layer that raster tiles are rendered by using `mapLayerType` property. For example, if you place your tiles at `NMAMapLayerTypeForeground` layer along with the `NMAMapObjects` you have created, you can use `zIndex` to control if the objects are rendered on top of, or beneath, the tiles.

Other Tile Customizations

`NMAMapTileLayer` supports properties to further customize the appearance of the tiles such as:

- tile size
- pixel format such as RGB or BGRA (to reduce memory usage)
- whether tiles fade in as they are drawn
- whether tiles from a different zoom level are rendered in place of a missing tile
- whether the tiles support transparency

Caching Tiles

Tiles can be cached to the disk to improve performance and reduce data traffic in the URL fetching case.

When you enable caching, you must provide a cache identifier. This identifier must be unique for each `NMAMapTileLayer` used within your application. Since the cache persists across app sessions, it is important to use the same identifier across sessions (by defining a constant, for example).

You can optionally limit the cache size and time to live for each cached tile. The cache can be cleared at any time by calling `[NMAMapTileLayer clearCache]`. To be sure the cache is completely cleared, first remove the `NMAMapTileLayer` from the map view before calling `[NMAMapTileLayer clearCache]`.

The following code enables disk caching with a 128MB maximum size and a tile time to live of 7 days:

```
NMAMapTileLayer *tileLayer = [[NMAMapTileLayer alloc]init];
tileLayer.dataSource = self; // Assuming self is a valid data source
[tileLayer setCacheEnabled:YES withIdentifier:@"MyUniqueTileCacheIdenfifier"];
tileLayer.cacheTimeToLive = 60 * 60 * 24 * 7; // 7 days
tileLayer.cacheSizeLimit = 1024 * 1024 * 128; // 128 MB
[mapView addMapTileLayer:tileLayer]; // NOTE: add to map view after setting tile properties
```

Performance Tips

1. IMPORTANT: Set `NMAMapTileLayer` properties before adding the tile layer to the map view. Most properties ignore attempts to set them after being added to the view.
2. Do not attempt to add a single `NMAMapTileLayer` instance to multiple maps.
3. Ensure the properties you set on the tile layer match the data you are supplying via the `NMAMapTileLayerDataSource` protocol. For example, do not enable transparency unless your tile data supports it.
4. Do not block `NMAMapTileLayerDataSource` methods for extended periods of time. For example, if it takes a while to generate tiles on the fly, move the processing to a separate GCD queue.
5. Ensure all `NMAMapTileRequest` objects received via `mapTileLayer:requestDataForTileAtX:y:zoomLevel:tileRequest:` are completed to avoid memory leaks.
6. Write directly to `NMAMapTileRequest bytesPtr` property to avoid memory copies.
7. If requesting a specific tile is constantly failing, consider implementing `mapTileLayer:hasTileAtX:y:zoomLevel:` returning `NO`.
8. Use the provided disk caching mechanism.

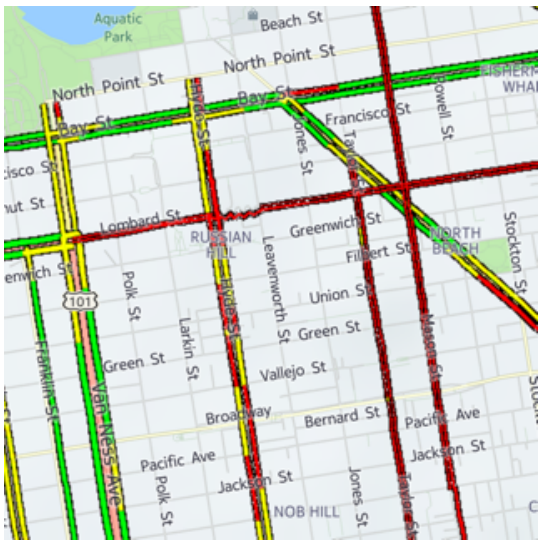
Traffic History

Traffic History allows the user to obtain map tiles that show the typical traffic pattern for a specific time point during the week. To display Traffic History tiles, create an instance of `HistoricalTrafficRasterTileSource` by specifying a day of the week and a time. For example, to show the traffic tiles for Wednesdays at 5:40pm, add the following:

```
NMAHistoricalTrafficTileLayer *historyTileLayer =
 [[NMAHistoricalTrafficTileLayer alloc]initForWeekDay:4 hour:17 minute:40];
```

```
[mapView addMapTileLayer:historyTileLayer];
```

Figure 37: Traffic History in San Francisco



Mobile Asset Management

The Mobile Asset Management (MAM) features provide useful information for logistics companies to manage their fleet vehicles. You can enable them using `showFleetFeature:`, whereas `hideFleetFeature:NMAMapFleetFeatureTypeAll` disables all enabled fleet features.

Fleet Vehicle Map

The fleet vehicle map scheme is a scheme optimized for fleet management. These schemes can be used to show road networks as well as truck toll and highway exits.

To display fleet maps, pick one of the following truck map schemes:

- `NMAMapSchemeTruckDay`
- `NMAMapSchemeTruckNight`
- `NMAMapSchemeHybridTruckDay`
- `NMAMapSchemeHybridTruckNight`

For example:

```
[self.mapView setMapScheme:NMAMapSchemeTruckDay];
```

The screenshot below shows highways with truck toll highlighted in purple and highway exit signs in Berlin.

Figure 38: Fleet Map of Berlin

For information about other map schemes see *Map Schemes* on page 39.

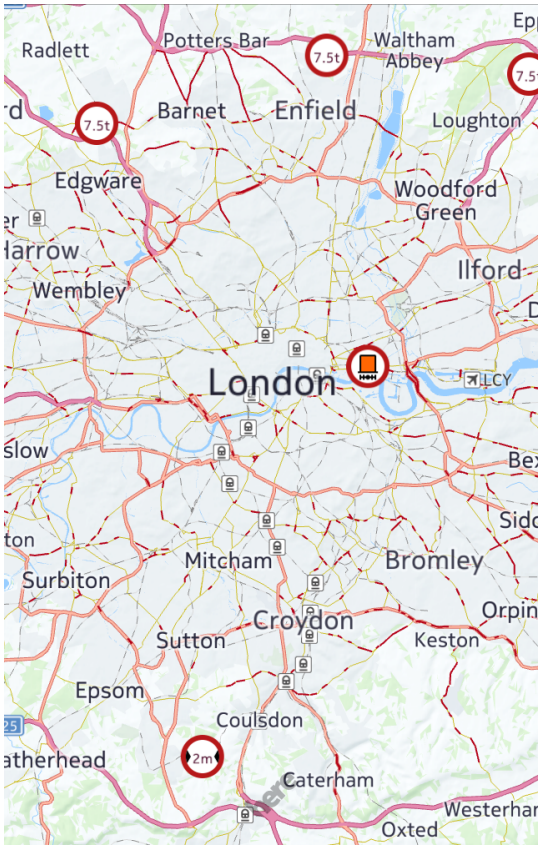
Truck Restrictions

This fleet feature contains information about heavy vehicle route restrictions such as height, weight, or environmental restrictions. For example, trucks carrying flammable materials may not travel on certain roads.

To display truck restrictions, add `NMAMapFleetFeatureTypeTruckRestrictions` to your fleet features:

```
[self.mapView showFleetFeature:NMAMapFleetFeatureTypeTruckRestrictions]:
```

Figure 39: Truck Restrictions in San Francisco



Congestion and Environmental Zones

This fleet feature highlights congestion and environmental zones. Congestion zones are areas where certain classes of vehicles must pay a toll to enter. Environmental Zone areas only admit certain kinds of vehicles depending on their emission class.

To display congestion and environmental zones, add `NMAMapFleetFeatureTypeCongestionZones` and `NMAMapFleetFeatureTypeEnvironmentalZones`:

```
[self.mapView showFleetFeature:NMAMapFleetFeatureTypeEnvironmentalZones];
[self.mapView showFleetFeature:NMAMapFleetFeatureTypeCongestionZones];
```

The screenshots below show congestion and environmental zones in London.

Figure 40: London Congestion Zones

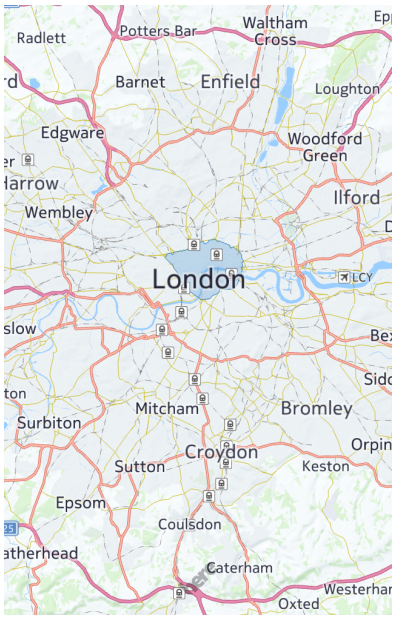


Figure 41: London Environmental Zones

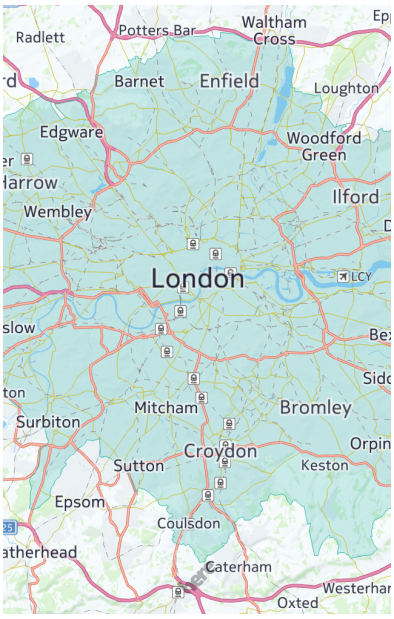
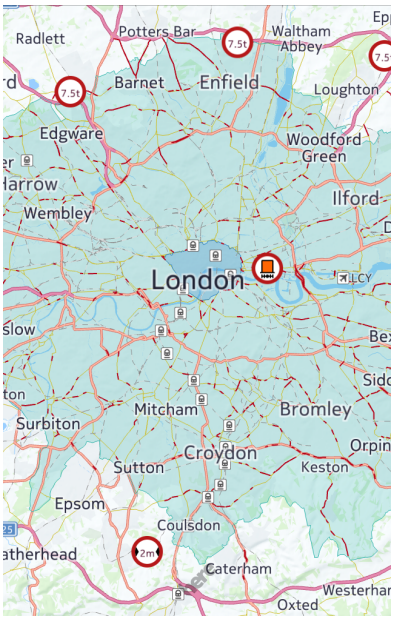


Figure 42: London Congestion and Environmental Zones (with Truck Restrictions)



Fleet Connectivity

The Fleet Connectivity features allow applications that use HERE iOS SDK to receive job dispatches from a fleet dispatcher. Each job dispatch contains custom information such as a geocoordinate to a destination that can be used by your application for navigation or other purposes.

Your application, which is known as an *asset* in a fleet connectivity context, can also use this feature to indicate its job status to the dispatcher. These job statuses include whether it is available to receive a job, whether the job was accepted, and whether the job is finished.

Note: HERE iOS SDK contains APIs for you to implement a fleet connectivity client. For instructions on how to use the fleet dispatcher features see the Fleet Connectivity Extension Developer's Guide at *developer.here.com*.

NMAFleetConnectivityService

This singleton class holds information such as the fleet asset ID, the fleet dispatcher to connect to, the running job, and job event polling interval.

Note: At least, you must set the fleet asset and dispatcher IDs before starting the fleet connectivity service.

NMAFleetConnectivityEvent and NMAFleetConnectivityServiceDelegate

You can set an `NMAFleetConnectivityServiceDelegate` to the service to listen for dispatcher events. Dispatcher events are highly customizable, with the only mandatory information being a `jobID`. Otherwise, you can define any type of information in its `content` payload such as a set of geocoordinates, a Place ID, or an address string.

There are two methods that need to be implemented in `NMAFleetConnectivityServiceDelegate`:

- `fleetConnectivityMessageReceived:`
- `fleetConnectivityEventAcknowledged:error:`

Using the Fleet Connectivity Feature

The basic flow of how to use the Fleet Connectivity feature is as follows:

1. Start the fleet connectivity service and begin listening for fleet connectivity events.

```
NMAFleetConnectivityService *service =
 [NMAFleetConnectivityService sharedFleetConnectivityService];
service.delegate = self;
service.dispatcherId = @"Driver-123";
service.assetId = @"Truck-321";
if ([service start]) {
 // service started
 self.serviceRunning = YES;
} else {
 // service failed to start
}
```

2. Implement the delegate. Once a message is received, check if it is a dispatch job.

```
- (void)fleetConnectivityMessageReceived:(NMAFleetConnectivityMessage *)message
{
 if (message.message.length > 0) {
  // display the optional message
 }
 if ([message isKindOfClass:[NMAFleetConnectivityJobMessage class]]) {
  // This message represents a job

  NMAFleetConnectivityJobMessage *newDestinationMessage =
   (NMAFleetConnectivityJobMessage *)message;
  // Get job ID from newDestinationMessage.jobId
  // Get location (in your preferred format) from newDestinationMessage.message
  // Get threshold from newDestinationMessage.etaThreshold
  // (this threshold controls when ETA updates are sent back to dispatcher)
 } else if ([message isKindOfClass:[NMAFleetConnectivityCustomMessage class]]) {
  // This message does not represent a job

  NMAFleetConnectivityCustomMessage *customMessage =
   (NMAFleetConnectivityCustomMessage *)message;
  // get job ID from customMessage.jobId
  // get messages from customMessage.content
 }
}
```

3. Send an event to the dispatcher that the job has been accepted.

```
NMAFleetConnectivityJobStartedEvent *event =
 [[NMAFleetConnectivityJobStartedEvent alloc] init];
// populate event.jobId
// populate event.etaThreshold
if ([[NMAFleetConnectivityService sharedFleetConnectivityService] sendEvent:event]) {
 // job is running
 // for example, your application can begin navigating to the job destination
} else {
 // job has failed to start
}
```

Alternatively, you can also reject the job by sending a rejection event.

```
NMAFleetConnectivityJobRejectedEvent *event =
 [[NMAFleetConnectivityJobRejectedEvent alloc] init];
// populate event.jobId
if ([[NMAFleetConnectivityService sharedFleetConnectivityService] sendEvent:event]) {
 // job is rejected
}
```

4. While a job is in the accepted state, you can tell the dispatcher that the job is canceled.

```
NMAFleetConnectivityJobCancelledEvent *event =
 [[NMAFleetConnectivityJobCancelledEvent alloc] init];
if (![[NMAFleetConnectivityService sharedFleetConnectivityService] sendEvent:event]) {
 // failed to cancel job
}
```

5. Upon job completion notify the server that the job is finished. For example, you can choose to send this event when your application has successfully finished the navigation session.

```
NMAFleetConnectivityJobFinishedEvent *event =
 [[NMAFleetConnectivityJobFinishedEvent alloc] init];
if (![[NMAFleetConnectivityService sharedFleetConnectivityService] sendEvent:event]) {
 // the job has failed to be finished
}
```

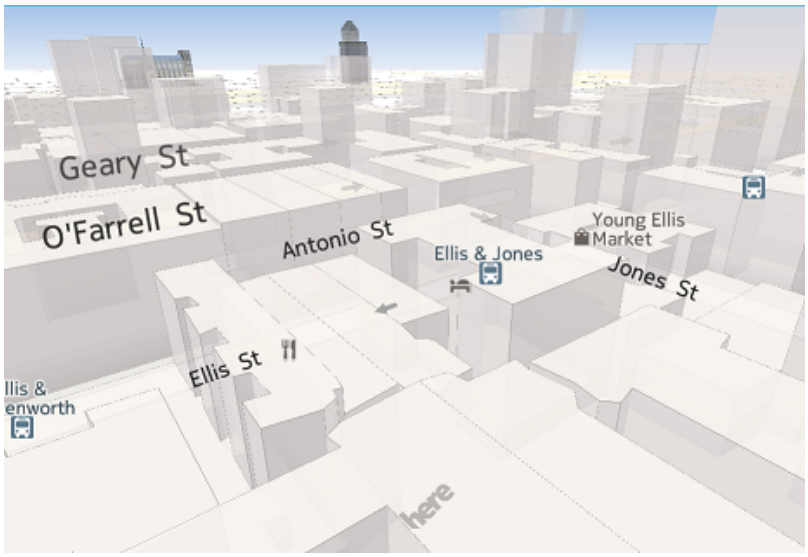
6. In the previous steps after sending an event you will receive an acknowledgment from the dispatching server through `fleetConnectivityEventAcknowledged:error:` callback.

```
- (void)fleetConnectivityEventAcknowledged:(NMAFleetConnectivityEvent *)event error:
(NMAFleetConnectivityError *)error
{
 if ([event isKindOfClass:[NMAFleetConnectivityJobStartedEvent class]]) {
  // the job start event is acknowledged
 } else if ([event isKindOfClass:[NMAFleetConnectivityJobRejectedEvent class]]) {
  // the job rejection event is acknowledged
 } else if ([event isKindOfClass:[NMAFleetConnectivityJobFinishedEvent class]]) {
  // the job completion event is acknowledged
 } else if ([event isKindOfClass:[NMAFleetConnectivityJobCancelledEvent class]]) {
  // the job cancellation event is acknowledged
 } else if ([event isKindOfClass:[NMAFleetConnectivityCustomEvent class]]) {
  // the custom event is acknowledged
 }
}
```

7. Stop the service by calling `[[NMAFleetConnectivityService sharedFleetConnectivityService] stop];`

Extruded Buildings

Figure 43: Extruded Buildings on a Map of San Francisco



HERE iOS SDK supports 3D representations of buildings and structures. This feature is called extruded buildings, and you can display them by setting `extrudedBuildingsVisible` property to `YES` in `NMAMapView`.

Note: The Extruded Buildings feature should not be used together with Raster Tiles. While enabling `extrudedBuildingsVisible` property ensure that you have also disabled Raster Tiles. For more information on these features please see *Custom Raster Tiles* on page 62 and *3D Venues* on page 173.

Transit Information

Your application can use API calls from HERE iOS SDK to display transit information for users.

Note: The transit map schemes (`NMAMapSchemeNormalDayTransit`, `NMAMapSchemeNormalNightTransit`, and `NMAMapSchemeHybridDayTransit`) are specifically designed for displaying transit information. You can optionally use one of these map schemes when your app displays transit information.

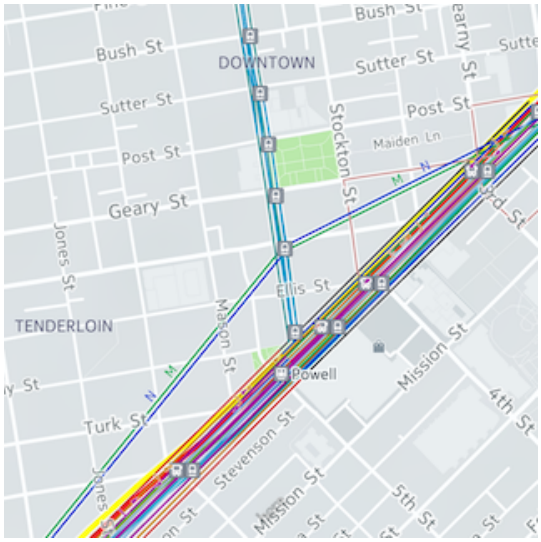
NMAMapTransitDisplayMode

Map transit data is displayed as a layer over a map area. To customize this transit layer, set `transitDisplayMode` property available in the `NMAMapView` class. For example, to show all transit information available:

```
// Assumes map is instantiated
```

```
mapView.transitDisplayMode = NMAMapTransitDisplayModeEverything;
```

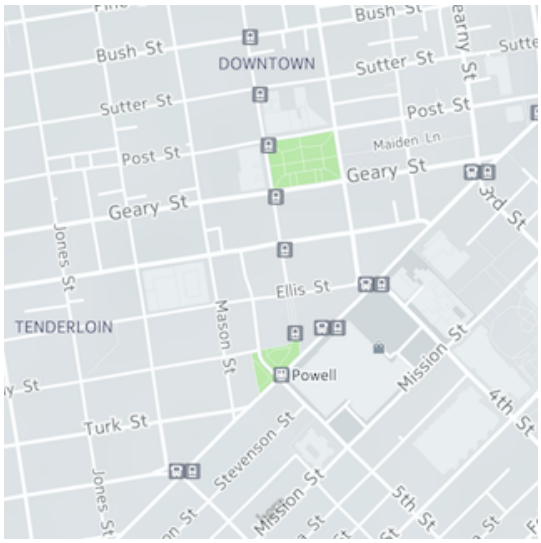
Figure 44: MapTransitLayer set to show everything



To show only transit stops and accesses, call:

```
// Assumes map is instantiated
mapView.transitDisplayMode = NMAMapTransitDisplayModeStopAndAccess;
```

Figure 45: MapTransitLayer set to show only transit stops and accesses



To hide all transit information call:

```
// Assumes map is instantiated
mapView.transitDisplayMode = NMAMapTransitDisplayModeNothing;
```

Note: `transitDisplayMode` settings may be affected when you change map schemes. For example, changing the map scheme to `NMAMapSchemeNormalDayTransit` enables "everything" mode. It is advisable that map scheme changes occur before changes in `transitDisplayMode`.

Map Transit Mode Example on GitHub

You can find an example that demonstrates this feature at *https://github.com/heremaps/* (Obj-C) and *https://github.com/heremaps/* (Swift).

Highlighting Transit Objects

The following four types of transit data objects are available:

- Transit Stop data - represented by `NMATransitStop`
- Transit Line data - represented by `NMATransitLine`
- Transit Access data - represented by `NMATransitAccess`
- Transit System Info data - represented by `NMATransitSystem`

These types of data are represented by `NMATransitObject` presentation objects, which are child instances of `NMAProxyObject`. They can be selected through tap gestures and passed to `NMATransitManager` to request for the appropriate data object.

The following is an example of using an `NMATransitObject` to retrieve a data object. If the tapped object was a transit line, then your application receives `transitManager:didGetLineInfo:forId:` callback with a populated `NMATransitLine`. Note that the retrieved transit data object has the same `uniqueId` as `transitObject`.

```
// assuming that this class adopts the NMATransitManagerDelegate protocol
[NMATransitManager sharedTransitManager].delegate = self;
// the transitObject is an NMATransitObject
[[NMATransitManager sharedTransitManager] requestInfoForObject:transitObject];
```

Depending on the use case there are several ways of getting a single one or a list of Identifier objects from a transit line.

- Use `uniqueId` property when the user has selected an `NMATransitLine`
- Use `NMATransitStop.lineIds` property when the user has selected an `NMATransitStop`. It returns a list of `Identifier` of the lines connected to this selected transit stop.

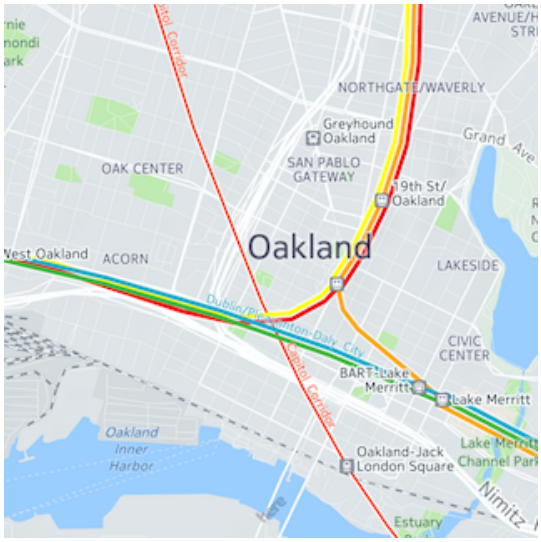
For details of handling tappable `NMAProxyObject` see *Objects and Interaction* on page 43.

With a single one or a list of unique identifiers you can call the following API to highlight the lines:

```
// Assumes mapView is instantiated and uniqueId is a valid identifier
```

```
[mapView.mapTransitLayer highlightTransitLinesFromUniqueIds:@[uniqueId]];
```

Figure 46: MapTransitLayer highlighting transit lines connected to the selected transit stop



NMATransitStop

An `NMATransitStop` object contains information about a transit stop. The following figures show the different types of transit stops:

Figure 47: NMATransitStop: A metro station

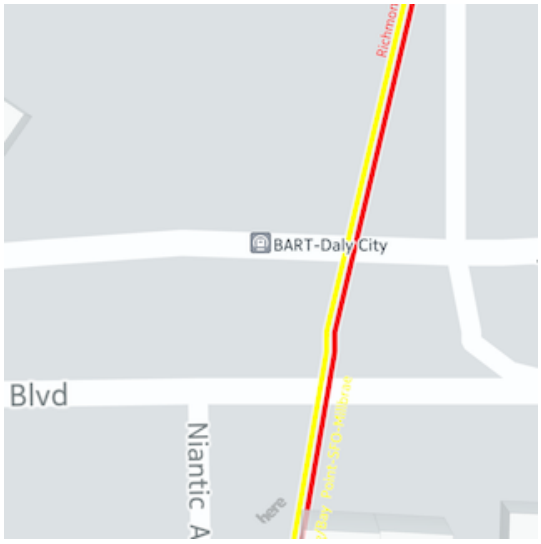


Figure 48: NMATransitStop: A ferry station

To acquire the `TransitStopObject`, implement `transitManager:didGetStopInfo:forId:` method in the `NMATransitManagerDelegate` protocol and perform a request from `NMATransitManager`.

The `TransitStopObject` class provides the following properties for getting information about the transit stop:

- `location` - gets the location coordinates of the transit stop
- `officialName` - gets the official name of the transit stop
- `informalName` - gets the informal name of the transit stop

- `uniqueId` - gets the identifier of the transit stop
- `systemIds` - gets the transit system this transit stop belongs to, it can be more than one
- `lineIds` - gets a list of `Identifier` objects for transit lines connected to this transit stop

You can also use `hasTransitType:type` method to see whether this stop supports a transit type. For example, a transit stop may support both public bus and a metro railway.

NMATransitLine

An `NMATransitLine` object contains information about a transit line. The following figure shows some examples of different types of transit lines:

Figure 49: Three types of transit lines: Metro, Train, and Water

To acquire the `NMATransitLine`, implement `transitManager:didGetLineInfo:forId:` method in the `NMATransitManagerDelegate` protocol and perform a request from `NMATransitManager`.

`NMATransitLine` provides a unique identifier as a class property. This identifier can be submitted to the `NMAMapTransitLayer` to highlight this line on the map (see *NMAMapTransitDisplayMode*).

The `NMATransitLine` class contains transit line information in the following properties:

- `officialName` - gets the official name of the transit line
- `informalName` - gets the informal name of the transit line
- `shortName` - gets the short name of the transit line
- `transitType` - gets the transit types (`NMATransitType`) this transit line belongs to
- `color` - gets the color associated with the line, if available
- `systemId` - gets the transit system this transit line belongs to
- `uniqueId` - gets the identifier of the transit line

NMATransitAccess

An `NMATransitAccess` object contains information about a transit access. A transit access is an entrance/exit to a transit stop. There can be multiple transit accesses to a transit stop.

Transit access is presented as a smaller transit stop, with a downward triangle attached to the bottom, that is only visible in higher zoom levels. The icons presenting the transit stops and accesses vary between different countries and companies. The following images show two examples:

Figure 50: Transit Stop and Access: Metro Station with Single Access

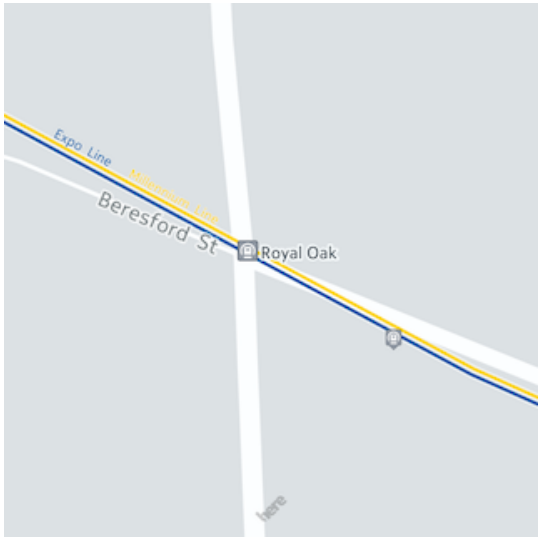


Figure 51: Transit Stop and Access: Metro Station with Multiple Accesses

To acquire the `NMATransitAccess`, implement `transitManager:didGetAccessInfo:forId:` method in the `NMATransitManagerDelegate` protocol and perform a request from `NMATransitManager`.

The `NMATransitAccess` provides the following properties for getting information about the transit access:

- `location` - gets the location coordinates of this transit access
- `name` - gets the name of this transit access
- `stopId` - gets a unique identifier of the transit stop this transit access leads to
- `uniqueId` - gets the identifier of the transit line

NMATransitSystem

The `NMATransitSystem` class contains information about a public transit system that can be accessed by calling one or more of the following properties:

- `officialName` - gets the official name of the transit system
- `website` - gets the website URL of the transit system
- `companyOfficialName` - gets the official transit system company name
- `companyWebsite` - gets the website URL of the transit system company
- `companyRoutePlannerWebsite` - gets the route planner URL of the transit system company
- `companyScheduleWebsite` - gets the schedule url of the transit system company
- `companyTelephoneNumber` - gets the phone number of the transit system company

To acquire the `NMATransitSystem`, implement `transitManager:didGetSystemInfo:forId:` method in the `NMATransitManagerDelegate` protocol and perform a request from `NMATransitManager`.

NMATransitManager

The `NMATransitManager` class is responsible for querying transit information of various types from a unique identifier with an `NMATransitManagerDelegate` for monitoring query results and triggering appropriate callback methods upon completion. Applications can call `[NMATransitManager sharedTransitManager]` class method to retrieve an `NMATransitManager` for querying transit information.

`NMATransitManagerDelegate` can be used to monitor query results from the `NMATransitManager`. It is required to be implemented within the application and submitted as part of the asynchronous query request.

The `NMATransitManagerDelegate` protocol contains the following callbacks:

- `transitManager:didGetLineInfo:forId:` - provides an `NMATransitLine` object
- `transitManager:didGetStopInfo:forId:` - provides an `NMATransitStop` object
- `transitManager:didGetAccessInfo:forId:` - provides an `NMATransitAccess` object
- `transitManager:didGetSystemInfo:forId:` - provides an `NMATransitSystem` object
- `transitManager:didCompleteWithError:` - signifies the asynchronous query request has completed. Please note that `NMATransitManager` rejects all subsequent requests unless it has completed the current request. An `NMATransitManagerErrorBusy` results if the `NMATransitManager` is busy.

An asynchronous request is submitted to the `NMATransitManager` along with `OnGetTransitInfoListener`. Note that the `NMATransitManager` instance is created by simply calling `NMATransitManager` constructor.

Transit-related enumerations

- `NMATransitObjectType` enum - represents values describing different transit object types: `NMATransitObjectTypeAccess`, `NMATransitObjectTypeLine` or `NMATransitObjectTypeStop`
- `NMATransitType` enum - represents values describing possible types of transit such as `NMATransitTypePublicBus`, `NMATransitTypeLightRail` or `NMATransitTypeWater`

Map Customization

Whether you want to optimize your map for a certain display size, use case, branding, or highlight objects which are important to your users, HERE SDK map customization feature allows you to have a fine level of control of your map view rendering characteristics.

This section presents the components and concepts you need to create your own map look-and-feel.

Map Schemes

To customize the map, the first step is to obtain an `NMACustomizableScheme` object from the map view. With this object you can then get and set properties to modify the map. You can change color, width, length, and other properties of map objects such as buildings, land features, roads, and so on. If the scheme you are customizing is currently at the affected zoom level and it is active, then the changes to properties are visible.

A custom scheme you create will not be permanently saved but will live as long as the map view object is in memory.

The class hierarchy is as follows:

1. `NMAMapView`
2. `NMACustomizableScheme`
3. `NMACustomizableVariable`

First you create or get a custom scheme from the map view, then get the respective property, `NMACustomizableVariable`, and finally modify its properties using its accessor methods.

Map Customization Example on GitHub

You can find an example that demonstrates this feature at *https://github.com/heremaps/* (Obj-C) and *https://github.com/heremaps/* (Swift).

Scheme Customization

Map customization starts by selecting one of the predefined schemes (such as `NMANormalDayScheme` and `NMANormalNightScheme`) to serve as a starting point. These predefined schemes are not customizable themselves but provide the initial values from which your custom scheme is derived. It is necessary to have the proper permission to access this base scheme as well as permission to customize.

Example of pre-defined schemes that we are already familiar with are (respectively, Normal Day, Normal Night):

Figure 52: Normal Day scheme

Figure 53: Normal Night scheme

Creating Your First Map Scheme Customization

In this example we will change the float property `CountryBoundary_Width` which causes the following rendering effect:

Figure 54: Normal Country Boundary

Figure 55: Adjusted Country Boundary

Let's learn how to implement this simple change. To begin, once you decided which scheme to base on, create an `NMACustomizableScheme` object with the map view method:

```
NMACustomizableScheme * customScheme;
customScheme = [self.mapView createCustomizableSchemeWithName:@"myCustomScheme" basedOnScheme:
 NMAMapSchemeNormalDay];
```

Once you created the customizable scheme, you can retrieve it again anytime with the following code as long as the map view object is not destroyed. Custom schemes are created and valid only for the specific Map View from which they were obtained.

```
NMACustomizableScheme * customScheme =
 [self.mapView.getCustomizableSchemeWithName:@"myCustomScheme"];
```

Before setting the attributes you define the zoom range for which the change shall take effect. For this purpose helper class `NMAZoomRange` is provided taking a minimum and maximum zoom level value:

```
NMAZoomRange * myZoomRange = [[NMAZoomRange alloc] initWithMinZoomLevel:0.0f and toZoomLevel:20.0f];
```

You are now ready to read and set values. In the following example we read and change `NMASchemeCountryBoundaryWidth` property:

```
// to read a primitive (float) property value: provide the property name and the zoom level
float returnValue = [customScheme floatForProperty:NMASchemeCountryBoundaryWidth forZoomLevel:
 10.0f];

// to set a float property: provide the property name, new value, and the previously created zoom
 range (NMAZoomRange)
```

```
[customScheme setFloatProperty NMASchemeCountryBoundaryWidth withValue:10.0f forZoomRange:
 myZoomRange];
```

As seen above, for simple types such as Integer and Float an object is not necessary for modification. For types such as color the methods return an object for easier manipulation.

Finally, you can activate the custom scheme in the map:

```
[self.mapView setMapScheme:@"myCustomScheme"];
```

Note: It is perfectly fine to create a custom theme and activate it before making any customization changes. This way when changes are applied, they are immediately rendered and visible on the map view as the properties are modified with their respective setters.

You should now see the map as the following:

Figure 56: Adjusted Country Boundary

Listing the available customizable properties

All available properties can be found in `NMACustomizableVariable.h` header file. For each property type an `NS_ENUM` exists, which is passed as an identifier of the property to its get and set methods. You can refer to the header file to find out its type.

One set of accessor is available for each type of property, for example: `integerForProperty:`, `floatForProperty:`, and `colorForProperty:`. The same goes for setter methods.

Note: The class `NMACustomizableVariable` also has the static method `allAvailableProperties` which returns an `NSDictionary` with all the properties HERE SDK supports for customization (containing all `NMACustomizableVariable` objects, with its name as the key). This is especially useful if one wants to programmatically iterate through all the properties. It is always possible to find out the property type by reading `propertyType` from the `NMACustomizableVariable` object.

In the iOS SDK the following helper classes exist:

- `NMACustomizableVariable` - Base class, used for primitive types such as integer and float
- `NMACustomizableColor` - Extends `NMACustomizableVariable` with methods to handle color properties

Changing Color Properties

In the following example you can see how to modify `NMASchemeBuildingColor` property:

Figure 57: Normal Building Color

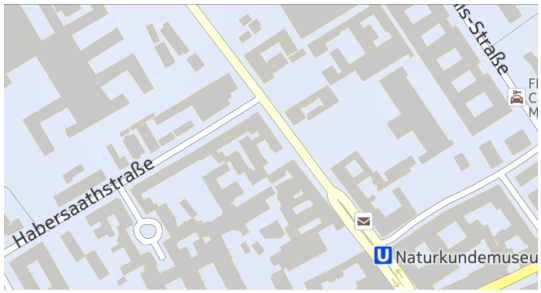
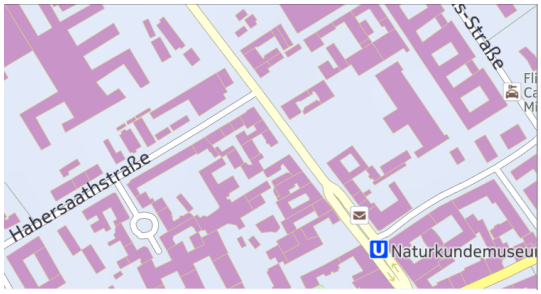


Figure 58: Adjusted Building Color



To modify color map attributes, we first obtain the custom color object `NMACustomizableColor`, then modify it:

```
NMACustomizableColor *buildingColor = [customScheme colorForProperty:NMASchemeBuildingColor
 forZoomLevel:2.0f];
[buildingColor setRed:100.0f];
[buildingColor setGreen:100.0f];
[buildingColor setBlue:133.0f];

// now apply the changes using the previously created zoom range
[customScheme setColorProperty:buildingColor forZoomRange:myZoomRange];
```

Positioning

HERE iOS SDK allows applications to choose from two different location information sources:

- Basic positioning is described in *Basic Positioning* on page 86 and provides a simple interface to the location information provided by the iOS platform. This location source is always at the developer's disposal regardless of HERE iOS SDK license.
- Advanced positioning is described in *Advanced Positioning by HERE* on page 90. Advanced positioning is accessed via the same interface as the basic positioning but the underlying location source is the HERE Positioning that provides developers, amongst other things, advanced offline network positioning as well as indoor positioning.

Note: To use the advanced HERE Positioning, one or more HERE Positioning features need to be enabled in HERE iOS SDK license.

Basic Positioning

An application created using HERE iOS SDK can use information from positioning capabilities of a user's device to display its position and, optionally, provide real-time updates. Getting the current position requires an application to make use of the `NMAPositioningManager` interface from HERE SDK. To receive position updates or position-lost notifications, an application should use `NSNotificationCenter addObserver` with the notification names found in `NMAPositioningManager.h`:

- `NMAPositioningManagerDidUpdatePositionNotification`
- `NMAPositioningManagerDidLosePositionNotification`

The user's current position can be easily displayed on the map using the `NMAPositionIndicator` class. Each instance of `NMAMapView` owns an instance of this class accessed via `positionIndicator` property.

NMAPositioningManager

The `NMAPositioningManager` class provides information related to the device geographical location such as the current position and the average speed. `NMAPositioningManager` is a singleton class and thus should only be accessed through the `sharedPositioningManager` class method.

To receive position updates, an object must subscribe to the `NMAPositioningManager` notifications. The notification update frequency can be controlled according to the data source which is set in `dataSource` property. By default, `dataSource` is an `NMADevicePositionSource` instance with one of the following `NMADevicePositioningMethod` convenience options to set the frequency of the device updates.

- `NMADevicePositioningMethodGPS` - sends standard position updates as the location changes. These updates are provided by the iOS API `CLLocationManager startUpdatingLocation`.
- `NMADevicePositioningMethodSignificantChanges` - sends position updates only when the user moves a significant distance. These updates are provided by the iOS API `CLLocationManager startMonitoringSignificantLocationChanges`. Use this option if you would like to save power.

Note: Add `NSLocationWhenInUseUsageDescription` to your project `Info.plist` so that `CLLocationManager` can properly access the user's location. The value of this key is displayed to the user when the system requests for permission to use location services. See *App Submission Requirements* on page 26 for recommended strings.

Note: In addition to setting the `dataSource` to an `NMADevicePositionSource` you can also set it to a custom data source that conforms to the `NMAPositionDataSource` protocol. For more information on `NMAPositionDataSource` consult the API Reference.

To start receiving real time positioning updates, the application needs to call `NMAPositioningManager startPositioning` which uses `NMADevicePositioningMethodGPS` as the default update mechanism. This method returns a `BOOL` value indicating whether or not positioning was successfully started.

While position updates are being received, the application can retrieve the current position of the client device through `NMAPositioningManager currentPosition` property. This current position is equal to either `rawPosition` or `mapMatchedPosition` property, depending on which seems more likely to be correct under the current circumstances. `rawPosition` is a position value from the current data source that has not been modified by HERE SDK engine, while `mapMatchedPosition` is a position data value that is

matched to the nearest car or pedestrian road, depending on `mapMatchMode` property. If the positioning manager is not active, or it has an invalid position, then `currentPosition` method returns `nil`.

Note: Map matching is disabled by default. It can be enabled automatically through the use of any HERE SDK feature which requires map matching, such as navigation, or it can be manually enabled by setting `mapMatchingEnabled` to `YES`. When map matching is disabled, `mapMatchedPosition` returns `nil`, and `currentPosition` returns the raw position.

When the application no longer requires position updates, it should notify the `NMAPositioningManager` by calling `stopPositioning`. Position updates are then stopped, provided that no other SDK services (such as `NMAPositionIndicator`) that require position updates are in use.

NMAPositioningManager Notifications

The `NMAPositioningManager` notifications can be used to track position updates of a client device as determined by its positioning mechanism (for example, its GPS). To register or unregister for these notifications, use the following methods:

```
[[NSNotificationCenter defaultCenter] addObserver:self
  selector:@selector(methodName)
      name:NMAPositioningManagerDidUpdatePositionNotification
    object:[NMAPositioningManager sharedPositioningManager]];
```

```
[[NSNotificationCenter defaultCenter] removeObserver:self
    name:NMAPositioningManagerDidUpdatePositionNotification
  object:[NMAPositioningManager sharedPositioningManager]];
```

Applications can register for two types of notifications:

- `NMAPositioningManagerDidUpdatePositionNotification`
- `NMAPositioningManagerDidLosePositionNotification`

Note: `NSNotificationCenter` does not limit how many times an object can register to the same notification. You should be careful not to register the same object more than once to a notification. Otherwise, the object receives duplicate notifications.

The following is an example of registering and handling these notifications in a `UIViewController`:

```
// Start positioning and register for position update notifications
- (void)viewDidLoad
{
 ...
 if ([[NMAPositioningManager sharedPositioningManager] startPositioning]) {
 // Register to positioning manager notifications
 [[NSNotificationCenter defaultCenter] addObserver:self
  selector:@selector(positionDidUpdate) name:NMAPositioningManagerDidUpdatePositionNotification
  object:[NMAPositioningManager sharedPositioningManager]];

 [[NSNotificationCenter defaultCenter] addObserver:self
  selector:@selector(didLosePosition) name: NMAPositioningManagerDidLosePositionNotification
  object:[NMAPositioningManager sharedPositioningManager]];
 }

 ...
}
// Handle NMAPositioningManagerDidUpdatePositionNotification
- (void)positionDidUpdate
{
```

```
 NMAGeoPosition *position = [[NMAPositioningManager sharedPositioningManager] currentPosition];
 [_mapView setGeoCenter:position.coordinates
    withAnimation:NMAMapAnimationLinear];
}
// Handle NMAPositioningManagerDidLosePositionNotification
- (void)didLosePosition
{
 ...
}
```

To avoid unnecessary position updates while the application is in the background, you can stop positioning and restart it when the application returns to the foreground using `UIApplicationDelegate` protocol callbacks.

The following code snippet demonstrates how to stop positioning and unregister from the notifications:

```
- (void)viewWillDisappear:(BOOL)animated
{
 [[NMAPositioningManager sharedPositioningManager] stopPositioning];
 [[NSNotificationCenter defaultCenter] removeObserver:self
  name:NMAPositioningManagerDidUpdatePositionNotification
  object:[NMAPositioningManager sharedPositioningManager]];
 [[NSNotificationCenter defaultCenter] removeObserver:self
  name:NMAPositioningManagerDidLosePositionNotification
  object:[NMAPositioningManager sharedPositioningManager]];
}
```

Note: Even if background location updates are enabled in your Xcode project, the `NMAPositioningManager` does not provide updates when your application is in the background unless there is an active navigation session. To override this behavior, call `setBackgroundUpdatesEnabled:YES` on `NMAPositionDataSource`.

Position Simulation

HERE SDK provides two classes, `NMALoggedPositionSource` and `NMARoutePositionSource`, which can be used to simulate position updates within an application. These classes implement the `NMAPositionDataSource` protocol; to use them, instances should be created, configured, and assigned to `dataSource` property of `NMAPositioningManager`. Only one position data source may be used at a time.

The `NMALoggedPositionSource` class provides positioning data from a log file. Currently, the GPX file format is supported. The positions listed in the log file are processed one by one until the end of the log is reached. The spacing of the updates is controlled with `updateInterval` property. `updateStyle` property can be used to modify how updates are generated. `stationary` property simulates halting of movement along the logged path, and `positionLost` property simulates the loss of the position data.

The `NMARoutePositionSource` class provides positioning data from a calculated `NMARoute`. Updates are generated from the beginning to the end of the route with the frequency controlled by `updateInterval` property. The simulated travel speed is set using `movementSpeed` property. Stopping and losing position are simulated with `stationary` and `positionLost` properties respectively.

Note: `NMALoggedPositionSource` does not support indoor positioning.

Creating Position Logs

You can also use HERE SDK to create GPX logs that can be replayed by `NMALoggedPositionSource`. To do this, set `logType` property in `NMAPositionManager` to the value `NMAPositionLogTypeDataSource`

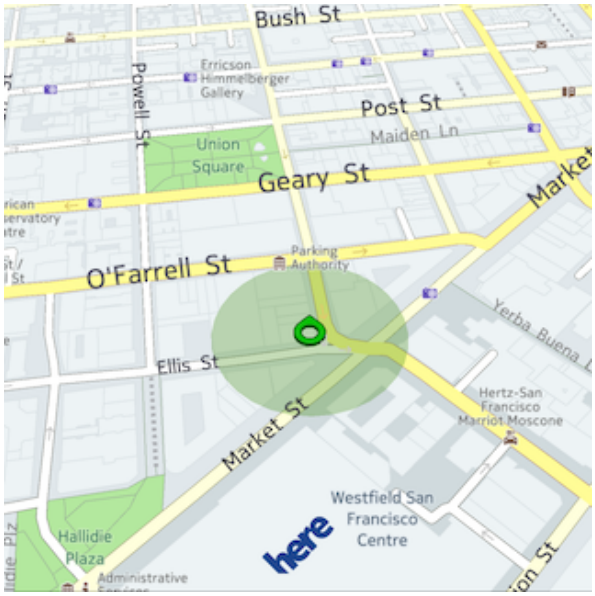
indicating that the position received from the current data source should be logged. GPX logs are created in `Documents` folder of your application. To disable position logging, set `NMAPositionLogType` to `NMAPositionLogTypeNone`.

Note: This feature is only intended for debugging purposes. Do not use Position Logging in a production application.

NMAPositionIndicator

The `NMAPositionIndicator` class provides a convenient way to add a map object that marks the user's current location as reported by the `NMAPositioningManager`. The position indicator can be set to display the raw, map-matched, or current position (a position that is automatically selected between raw or map-matched). The position indicator is rendered as a circular object within a translucent circle, the diameter of which illustrates the accuracy of the indicated position. The types of map objects can be used to customize `NMAPositionIndicator` are `NMAMapMarker` , `NMAMapLocalModel`, and `NMAMapCircle`.

Figure 59: An NMAPositionIndicator



Each `NMAMapView` instance has an `NMAPositionIndicator` instance which can be accessed from `NMAMapView positionIndicator` property. The map object displayed by `NMAPositionIndicator` can be changed with its `displayObject` property, and the indicator can be shown or hidden with its `visible` property.

Note: You can use `tracksCourse` property to control whether the position indicator is automatically oriented to the current direction of movement. When enabled, HERE SDK rotates the `positionIndicator` to line up with the direction the vehicle is travelling in. In other words, if the vehicle is idle, there are no changes. If you need the position indicator to point to the device compass direction instead, use *CLLocationManagerDelegate*.

Note that `orientationOffset` is also only applicable when the positionIndicator yaw has changed based on the vehicle movement.

You can customize the accuracy circle color and whether it is visible by using `accuracyIndicatorColor` and `accuracyIndicatorVisible` properties.

```
// Display position indicator
```

```
mapView.positionIndicator.visible = YES;
```

Note: Setting `NMAPositionIndicator` to visible automatically enables `NMAPositioningManager` updates.

For the position indicator to stay in the center of the map and illustrate real-time updates of the device position, it is necessary to update the map center whenever a new location update is received. Please note that frequently re-drawing the map in this manner consumes device battery life. You should be aware of battery power implications while performing real-time updates. The following code can be used to update the map location when a position update is received:

```
- (void)positionDidUpdate
{
 NMAGeoPosition *position = [[NMAPositioningManager sharedPositioningManager] currentPosition];
 [_mapView setGeoCenter:position.coordinates
    withAnimation:NMAMapAnimationLinear];
}
```

For more information about the classes introduced and demonstrated in this section refer to the *API reference documentation*.

Advanced Positioning by HERE

In addition to the basic positioning services HERE iOS SDK provides advanced HERE Positioning with the following key features:

- High accuracy indoor positioning with building and floor detection using **Bluetooth**™ radio
- Automatic, on-demand download of radio positioning data (*radiomaps*) for positioning without network connection (offline)
- Automatic positioning method switching between satellite-based (Global Navigation Satellite System, GNSS) and Bluetooth positioning, providing the best possible position information using the available position methods
- HERE Indoor Positioning supports both private and public data. You can have a private venue that is mapped through HERE *Private Venues* on page 179 or your own custom indoor map, with the indoor location information being only available for your applications. In contrast, indoor location information for public venues is available for all HERE SDK users with an appropriate license
- Global data hosting infrastructure for the optimal availability, reliability, and user experience

HERE Indoor Positioning information

To use HERE Indoor Positioning, indoor positioning support must be deployed to a venue. To do this, please visit *indoor.here.com*. All the required information to setup HERE Indoor Positioning is available there. Especially familiarize yourself with *HERE Indoor Positioning Installation Guide*.

After the deployment HERE Indoor Positioning can use Bluetooth radio for indoor positioning. Due to the differences in the iOS device capabilities, HERE Indoor Positioning performance may differ between the device models and iOS versions.

Wi-Fi -based indoor positioning is not supported in iOS due to the platform and operating system limitations.

HERE Positioning Feature Groups

HERE Positioning is split into features groups. Depending on your business plan, you have an access to one or more of the following feature groups:

- **Private Indoor** - This feature provides high accuracy indoor positioning for *private* venues from which indoor Bluetooth radio data has been collected using *HERE Indoor Radio Mapper*. The feature provides indoor positioning capability that is accessible only by the owner of the positioning data. You can find further details in *HERE Indoor Positioning Installation Guide*.
- **Public Indoor** - This feature provides high accuracy indoor positioning for *public* venues from which indoor Bluetooth radio data has been collected using *HERE Indoor Radio Mapper*. The feature uses HERE Indoor Positioning community radiomap that is accessible by all the HERE iOS SDK users having access to this feature. Further details can be found in *HERE Indoor Positioning Installation Guide*.

HERE Indoor Positioning (private and public) works by utilizing downloaded positioning assistance data, such as *radiomaps*, for position estimation within the device. Network connectivity is only required for the radiomap download after which no connectivity is needed unless further radiomap tiles are needed.

Radiomap download is handled by the *on-demand downloader* which automatically downloads radiomap tiles in the vicinity of the device. The downloader also maintains radiomaps on the device by updating the radiomap tiles at regular intervals as needed and by removing the oldest radiomap tiles when the radiomap tiles exceed the storage quota. This quota is configured as speficied in the table below. The table also shows how often HERE iOS SDK attempts to update the radiomaps.

Technology	Quota	Update interval	Description
Indoor Bluetooth	32MB	7 days	Contains high accuracy radiomaps for indoor Bluetooth positioning

Using HERE Positioning

To start using HERE Positioning in your iOS applications, perform the following steps:

1. Find **Capabilities** tab in the Xcode project settings and enable **Background Modes** for *Location Updates* and *Uses Bluetooth LE accessories*.
2. Declare *Private Data Access* and *App Transport Security* exceptions by updating `Info.plist` of the app. See *App Submission Requirements* on page 26 for more details.
3. After completing the previous configuration steps you can change the positioning manager data source to HERE Positioning with the following code:

```
NMAPositioningManager.sharedPositioningManager.dataSource = [[NMAHEREPositionSource alloc]
init];
```

Running HERE Positioning in background

To make HERE Positioning run in background, add the following line to the app delegate function `applicationDidBecomeActive`:

```
                    [NMAPositioningManager.sharedPositioningManager.dataSource
setBackgroundUpdatesEnabled:true];
```

Moreover, ensure that your `Info.plist` has the following attributes:

```
<key>NSLocationAlwaysUsageDescription</key>
<string>This application uses location information</string>

<key>NSLocationWhenInUseUsageDescription</key>
<string>This application uses location information</string>

<key>NSLocationAlwaysAndWhenInUseUsageDescription</key>
<string>This application uses location information</string>

<key>NSMotionUsageDescription</key>
<string>This application uses motion sensors for indoor positioning</string>

<key>NSBluetoothPeripheralUsageDescription</key>
<string>This application uses Bluetooth for indoor positioning</string>

<key>NSBluetoothAlwaysUsageDescription</key>
<string>This application uses Bluetooth for indoor positioning</string>

<key>UIBackgroundModes</key>
<array>
   <string>bluetooth-central</string>
   <string>location</string>
</array>
```

Indoor Positioning Mode

The *indoor positioning mode* refers to the set of radiomaps used by the indoor positioning engine. For detailed discussion on the different radiomaps please refer to *HERE Indoor Positioning Installation Guide*.

Indoor positioning can be configured to run in four different modes which are defined by `NMAIndoorPositioningMode` enumeration. Use the property `indoorPositioningMode` on the `NMAHEREPositionSource` interface to change the indoor positioning mode. You can use the same property to check the current mode.

The following table introduces the four indoor positioning modes.

Table 2: Indoor Positioning Modes Supported by HERE Positioning

Mode	Description
`AUTOMATIC`	HERE Positioning automatically chooses which mode to apply. Requires that the SDK license has both public and private indoor positioning features enabled. If both features are enabled in the SDK license, then this positioning mode is set as the default.
`COMMUNITY`	Community indoor radiomap is used by the indoor positioning engine. Requires an SDK license with the public indoor positioning feature enabled. If only the public indoor positioning feature is enabled in the SDK license, then this indoor positioning mode is set as the default.
`PRIVATE`	A private indoor radiomap is used by the indoor positioning engine. Requires an SDK license with the private indoor positioning feature enabled. If only the private indoor positioning feature is enabled in the SDK license, then this indoor positioning mode is set as the default.

Indoor Positioning-specific Properties

HERE Positioning has three indoor positioning-specific properties in `NMAGeoPosition`:

- `NSString buildingName`

Contains the building name, if known. If the building name is not present or unknown, it is set to `nil`.

For HERE Venue Maps the building name is assigned by HERE when creating the Venue Map.

For custom indoor maps the building name is the one given when the indoor map was imported using HERE Indoor Radio Mapper. However, illegal characters are removed from the name.

- `NSString buildingId`

  Contains the building ID, if known. If the building ID is not present or unknown, it is set to `nil`.

  For HERE Venue Maps the building ID is a globally unique ID assigned by HERE when creating the Venue Map. An example of a HERE Venue Map building ID is `DM_8213`.

  For custom indoor maps the building ID is a user-defined `buildingName` prefixed with "`BM_`".

  If the position estimate is outdoors but indoor positioning is used for producing the location estimate, the building ID string is populated as `OUTDOOR`.

- `NSNumber floorId`

  Contains the floor level information. If the floor ID is unknown or not present, it is set to `nil`.

  For HERE Venue Maps the floor levels are returned as defined for the particular HERE Venue Map.

  For the custom indoor maps the floor level follows the floor levels specified in HERE Indoor Radio Mapper when the indoor maps were imported.

Hints for HERE Positioning API usage

- When using HERE Positioning for indoor positioning, the positioning API will return non-indoor location for the first few seconds even when in a venue with indoor positioning coverage. In such a case one option is to discard the non-indoor locations for the first few seconds in the application start-up.
- iOS tutorial projects are available through GitHub at *https://github.com/heremaps/here-ios-sdk-examples/tree/master/here-positioning-ios* (Obj-C) and *https://github.com/heremaps/here-ios-sdk-examples/tree/master/here-positioning-ios-swift* (Swift). The project helps you to get your first HERE Positioning application up and running smoothly.

Troubleshooting HERE Positioning

If your application does not receive any position updates while using HERE Positioning, try the following:

- Check that the application has *Capabilities* set in Xcode as defined in this document. See *Using HERE Positioning* on page 91.
- Check that the application has *Private User Data Access* declarations set in Xcode as defined in this document. See *Using HERE Positioning* on page 91.
- Make sure that the relevant network settings are enabled:
  - Go to the device **Settings** and check that Bluetooth is enabled
  - Go to **Settings** and check from the application settings that *Location Access* is set to *Always* and that *Motion and Fitness and Mobile Data* access is enabled.
- Make sure that the device has a network connection.
- Make sure the *indoor positioning mode* is set correctly.

If your application receives inaccurate position estimates while using HERE Positioning, try the following:

- Make sure that the location estimates originate from the indoor positioning engine by checking the position source through the source property in `NMAGeoPosition`.

Map Matching

HERE iOS SDK performs Map Matching automatically when it needs to match a raw position to the road network, e.g. during drive guidance when there may be inaccuracies in the road rendering or GPS data. Map matching also supports cases when the GPS signal is lost while entering a road tunnel. The position is extrapolated and updated based on the driver's speed and knowledge of the tunnel layout.

Automotive Map Matching

HERE iOS SDK supports high-accuracy map matching through the `NMAPositionDataSourceAutomotive` class. As a requirement to use this class, you should have positioning data input from a GNSS module that supports dead reckoning. This ensures a continuous and reliable stream of position updates even in cases when GPS becomes unavailable (for example, when the user is driving in a tunnel). It is strongly recommended that position updates are provided at a constant rate of 10 Hz together with standard deviations of the following:

- Horizontal radial error (large component)
- Horizontal radial error (small component)
- Course
- Speed
- Elevation

Note:

- Automotive Map Matching is currently offered as a beta feature. APIs may change without notice.
- Automotive Map Matching does not support tunnel extrapolation.

Custom Data Sources

In general, you can also use any custom positioning data by implementing `NMAPositionDataSource` class and setting it to `dataSource` property in `NMAPositioningManager` before starting the positioning manager.

Directions

Car and Pedestrian Routing

HERE iOS SDK supports route calculation with multiple waypoints optimized for walking or driving.

A route describes a path between at least two waypoints, the starting point and the destination, with optional intermediate waypoints in between. Applications can provide route information to users in two ways:

- A line rendered on a map that displays a connecting path between all waypoints
- Turn-by-turn directions in text format

Routing Example on GitHub

You can find an example that demonstrates this feature at *https://github.com/heremaps/* (Obj-C) and *https://github.com/heremaps/* (Swift).

NMACoreRouter

`NMACoreRouter` class is responsible for calculating an `NMARoute` using a list of stops and an `NMARoutingMode`. It also provides an `NMACalculateResultBlock` for monitoring calculation progress and triggering appropriate callback methods upon completion. To calculate a route, call `calculateRouteWithStops:routingMode:completionBlock:` method. You can launch parallel routing requests by using separate `NMACoreRouter` instances to launch calculation requests.

The `NMACoreRouter` can also calculate routes with traffic taken into account by setting `dynamicPenalty` property before launching a route calculation. For more information on the `NMADynamicPenalty` class see *Dynamic Routing Penalty* on page 101.

Note: `NMACoreRouter` variables must be strongly referenced and retained (e.g. made into a class member). Otherwise, the object may go out of scope and be garbage collected.

NMAWaypoint

The `NMACoreRouter` supports waypoints as `NMAPlace`, `NMAPlaceLocation`, or `NMAWaypoint` objects.

You can use `NMAWaypoint` to add more waypoint details to a car route calculation. These details include whether a waypoint is a deliberate stopover or a pass-through point that the route must pass through. This affects routing, as routes containing stopovers or pass-through waypoints may be different. For example, a calculated route may suggest a U-turn maneuver after a stopover, while a route containing the same location as a pass-through waypoint suggests continuing on the same street. The pass-through waypoint type is only supported in car routes and it is not supported in other route types.

NMARoutingMode

The `NMARoutingMode` class is a model of the parameters required to calculate an `NMARoute` such as:

- `routingType` - the routing type, such as Fastest or Shortest
- `transportMode` - the mode of transportation
- `routingOptions` - the routing options (represented by `NMARoutingOption` enums) applicable for this route
- `startDirection` - the direction of travel that the route should start in. By default this is set to "any direction" to achieve the fastest possible route
- `departureTime` - the departure time for the route
- `resultLimit` - the maximum number of alternate routes to calculate (the actual number of results may be fewer than this limit)

Note: Routing type describes different optimizations that can be applied during the route calculation:

- `Fastest` - route calculation from start to destination optimized by travel time. In some cases the route returned by the fastest mode may not be the route with the shortest possible travel time. For example, it may favor a route that remains on a highway even if a shorter travel time can be achieved by taking a detour or shortcut through a side road.

- `Shortest` - route calculation from start to destination disregarding any speed information. In this mode the distance of the route is minimized while keeping the route sensible. This includes, for example, penalizing turns. Because of that the resulting route will not necessarily be the one with minimal distance.
- `Balanced` - route calculation from start to destination optimizing based on combination of travel time and distance.

Note: HERE SDK allows for more than one route to be returned from a route calculation between two waypoints. You can use the `NMARoutingMode` class to set the desired number of routes, and HERE SDK then returns different routes according to this limit. Note that the first element of the returned array is the best result based on the routing options, and the rest of the returned routes are not listed in any specific order.

NMARoute

The `NMARoute` class represents a distinct calculated path connecting two or more waypoints and consists of a list of maneuvers and route links. A call to `calculateRouteWithStops:routingMode:completionBlock:` method of `NMACoreRouter` triggers a route calculation while the returned `NSProgress` object and `NMACalculateResultBlock` block can be used to monitor the operation and process the resulting `NMARoute` objects.

An `NMARoute` object contains route information that can be accessed by calling one or more of the following methods:

- `routingMode` - the `NMARoutingMode` for the route
- `waypoints` - the array of all waypoints for the route
- `start` - the starting waypoint for the route
- `destination` - the destination waypoint for the route
- `maneuvers` - the array of maneuvers for the route
- `length` - the length of the route, in meters
- `ttaIncludingTrafficForSubleg` - the `NMARouteTta` indicating the estimated time to arrival taking into account traffic conditions at the time of the route calculation
- `ttaUsingDownloadedTrafficForSubleg` - the `NMARouteTta` indicating the estimated time to arrival with traffic conditions while the traffic conditions are taken from traffic data downloaded to the device
- `ttaExcludingTrafficForSubleg` - the `NMARouteTta` indicating the estimated time to arrival without considering traffic conditions
- `boundingBox` - gets the smallest `NMAGeoBoundingBox` that contains the entire route
- `routeGeometry` - gets the array of all `NMAGeoCoordinates` along the route
- `mapPolyline` - gets the `NMAMapPolyline` representation of the route

NMARouteTta

The `NMARouteTta` ("time-to-arrival") class provides useful information about the route such as duration and route details that impact travel duration including car pool restrictions, turn restrictions, and blocked roads. For example, to retrieve a duration for a calculated route, use `tta` property. For example,

```
NSTimeInterval duration = [route ttaExcludingTrafficForSubleg:NMARouteSublegWhole].duration;
```

You can also retrieve the duration for a subleg from a route by using `ttaForSubleg:` method. For example,

```
if (myRoute.sublegCount > 0)
{
 NSTimeInterval duration = [myRoute ttaExcludingTrafficForSubleg:0].duration;
}
```

Traffic-Aware Routing and NMARouteTta

The `NMARoute` class also provides `ttaIncludingTrafficForSubleg` and `ttaUsingDownloadedTrafficForSubleg` methods which provide a different set of `NMARouteTta` results that take traffic into account. If the route was originally calculated with setting a traffic penalty mode, `ttaIncludingTrafficForSubleg` method will return the estimated time to arrival with traffic conditions at the time of the route calculation. The value is calculated once at the time of route calculation and does not change with time. `ttaUsingDownloadedTrafficForSubleg` should be used in cases when the time elapsed since a route was returned is large enough to make the time to arrival (as obtained using `ttaIncludingTrafficForSubleg`) no longer accurate, given changing traffic conditions. `ttaUsingDownloadedTrafficForSubleg` will provide a more up-to-date time to arrival estimate provided traffic data has been constantly refreshed using the `NMATrafficManager` class.

Code snippet on how to use `ttaIncludingTrafficForSubleg` to get time to arrival considering traffic at the time the route was calculated.

```
...
NMADynamicPenalty *penalty = [[NMADynamicPenalty alloc] init];
penalty.trafficPenaltyMode = NMATrafficPenaltyModeOptimal;
coreRouter.dynamicPenalty = penalty;
[coreRouter calculateRouteWithStops:stops routingMode:routingMode
 completionBlock:^(NMARouteResult *routeResult, NMARoutingError error) { ... }];
NSTimeInterval duration = [route ttaIncludingTrafficForSubleg:NMARouteSublegWhole].duration;
```

If you have to calculate duration based on downloaded traffic data, you can also perform the following to request for traffic data to be populated:

```
[[NMATrafficManager sharedTrafficManager] requestTrafficOnRoute:route];
// Wait for the callback that the traffic data has been downloaded: trafficDataDidFinish
NSTimeInterval durationWithTraffic = [route
 ttaUsingDownloadedTrafficForSubleg:NMARouteSublegWhole].duration;
```

Note: A traffic-aware route calculation may return more violated options. For more information see *Routing-related enumerations* and the API Reference.

NMAManeuver

The `NMAManeuver` class represents the action required to go from one segment to the next within a calculated `NMARoute`. Each `NMAManeuver` object provides information such as:

- location of the maneuver
- action required to complete the maneuver
- distance between maneuvers
- current road
- next road
- estimated time of the maneuver
- highway signpost (if any) indicating entrance, exit, or merge information
- a list of route elements representing portions of this maneuver

For more information please consult the API Reference.

NMARouteElement and NMARoadElement

`NMARouteElement` and `NMARoadElement` represent portions within a maneuver. For example, a maneuver may ask the driver to turn left and then remain on a street but this street may be comprised of multiple sections including a tunnel, a dirt road, and a toll road. In this situation the maneuver contains multiple `NMARouteElement` objects, with each element containing an `NMARoadElement` property that can provide your application with information about the individual section of the road.

NMAMapRoute

The `NMAMapRoute` class is a type of `NMAMapObject` that displays a calculated route on a map. Typically, an application creates an `NMAMapRoute` after a route calculation and adds the `NMAMapRoute` to the map by calling `NMAMapView addMapObject:`.

You can customize map route colors by using `renderType` property and `NMAMapRoute` APIs. There are three supported render type values: `NMAMapRouteRenderTypePrimary`, `NMAMapRouteRenderTypeSecondary`, and `NMAMapRouteRenderTypeUserDefined`. By default, an `NMAMapRoute` is set to the Primary set of pre-defined colors, and the secondary render type is designed to denote alternate routes. By using

`NMAMapRouteRenderTypeUserDefined`, you can use `color` and `traveledColor` properties and set the route colors to any ARGB color.

Figure 60: Using Primary RenderType

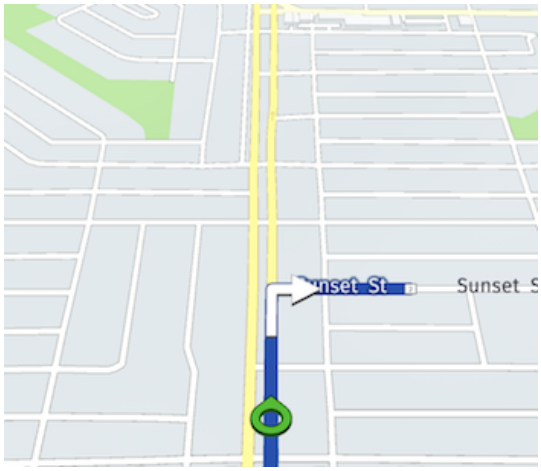


Figure 61: Using Secondary RenderType

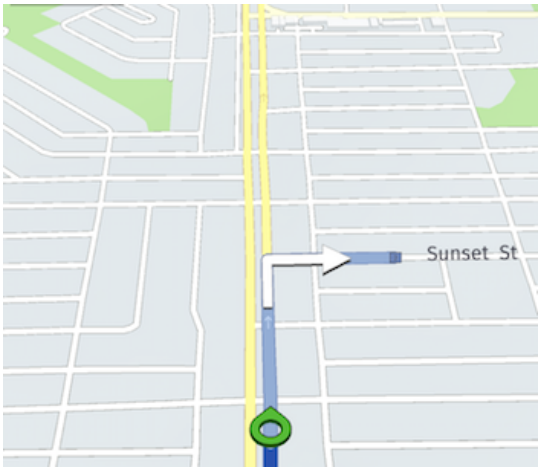
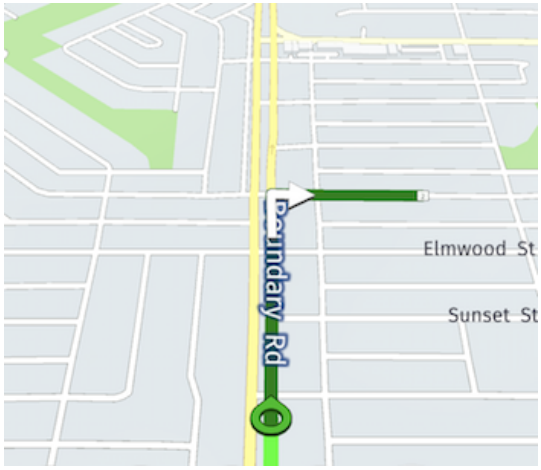
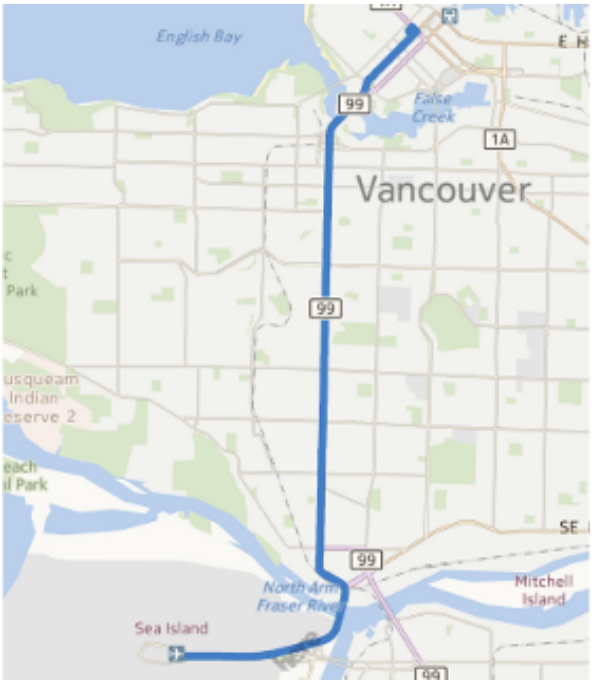


Figure 62: User-Defined Route Color



For example, if you want to render a route that connects two waypoints (start and destination), you can add the following application logic:

Figure 63: An `NMAMapRoute` added to an `NMAMapView`



1. Create an `NMACoreRouter`

```
// Create a NMACoreRouter.
NMACoreRouter* coreRouter = [[NMACoreRouter alloc] init];
```

2. Create an `NSMutableArray` and add two `NMAGeoCoordinates` stops

```
NMAGeoCoordinates* geoCoord1 = [[NMAGeoCoordinates alloc] initWithLatitude:49.1966286
 longitude:-123.0053635];
NMAGeoCoordinates* geoCoord2 = [[NMAGeoCoordinates alloc] initWithLatitude:49.1947289
 longitude:-123.1762924];

NMAWaypoint* waypoint1 = [[NMAWaypoint alloc] initWithGeoCoordinates:geoCoord1];
NMAWaypoint* waypoint2 = [[NMAWaypoint alloc] initWithGeoCoordinates:geoCoord2];

NSMutableArray* stops = [[NSMutableArray alloc] initWithCapacity:4];
[stops addObject:waypoint1];
[stops addObject:waypoint2];
```

3. Create an `NMARoutingMode` and set its `NMATransportMode`, `NMARoutingType`, and `NMARoutingOption` values

```
NMARoutingMode* routingMode = [[NMARoutingMode alloc]
        initWithRoutingType:NMARoutingTypeFastest
        transportMode:NMATransportModeCar
        routingOptions:0];
```

4. If you want to calculate the route while taking traffic conditions into account, set the following flag:

```
NMADynamicPenalty *penalty = [[NMADynamicPenalty alloc] init];
penalty.trafficPenaltyMode = NMATrafficPenaltyModeOptimal;
coreRouter.dynamicPenalty = penalty;
```

5. Calculate the route and receive the results of the route calculation.

```
[coreRouter calculateRouteWithStops:stops routingMode:routingMode
  completionBlock:^(NMARouteResult *routeResult, NMARoutingError error) {

   // If the route was calculated successfully
   if (!error && routeResult && routeResult.routes.count > 0)
   {
    NMARoute* route = [routeResult.routes objectAtIndex:0];
    // Render the route on the map
    NMAMapRoute* mapRoute = [NMAMapRoute mapRouteWithRoute:route];
    [mapView addMapObject:mapRoute];

    // To see the entire route, we orient the map view accordingly
    [mapView setBoundingBox:route.boundingBox
        withAnimation:NMAMapAnimationLinear];
   }
   else if (error)
   {
    // Display a message indicating route calculation failure
   }
  }];
```

Note: Routes are returned even if you receive `NMARoutingErrorViolatesOptions` error. It is up to you to handle these route results that violate routing options.

Dynamic Routing Penalty

You can use `NMADynamicPenalty` to create a policy of roads and area restriction factors applied during routing calculations. For example, you can use this class to indicate that the travel speed in an area is 50 percent slower than usual. The `NMADynamicPenalty` class also allows you to set the mode used for handling traffic events in a route calculation through `trafficPenaltyMode` property.

You can change the active policy by setting `dynamicPenalty` property in `NMACoreRouter`. The policy must be set before a route calculation for the restrictions to be taken into account.

Retrieving Traffic Event Objects

You can use the `NMATrafficManager` class to retrieve traffic events such as road closures or congestions. This is useful if your app needs to display a list of current traffic events for a given route.

Using `NMATrafficManager` is a two-step process. First, retrieve the events on the desired route using `requestTrafficOnRoute:` method:

```
NSNumber *requestID = [[NMATrafficManager sharedTrafficManager] requestTrafficOnRoute:myRoute];
```

`requestTrafficOnRoute:` method launches an asynchronous request to download the current traffic data to your device.

Note: Traffic data is also periodically downloaded when `NMAMapView` is set to display traffic. However, calling `requestTrafficOnRoute:` explicitly is recommended before retrieving traffic events using `NMATrafficManager`.

Upon the successful `trafficDataDidFinish` callback, you can retrieve a list of traffic events that affect the given route or route element by using one of the following methods:

- `getTrafficEventsOnRoute:withCompletion:`
- `getTrafficEventsOnRouteElements:withCompletion:`

Offline Routing

Even without an active data connection applications developed with HERE iOS SDK are able to request routing information to assist travelling from one location to another.

Your application users do not need to maintain an active data connection to calculate routes and render them on a map. It is possible to pre-download updated maps and database information for performing routing requests while offline. For example, if a user has downloaded offline maps of California and Oregon states, a route from San Diego to Portland can be created without any data connection.

For more information about downloading offline maps refer to *Preloading Map Data* on page 57.

Note: In HERE SDK v3.4 we have updated the behavior of routing connectivity modes (`NMARequestConnectivity`) to be more consistent with other parts of the SDK.

- If you launch a request using the "default" connectivity mode, the request is performed according to `NMAApplicationContext` connectivity setting. If the device is offline while `NMAApplicationContext` is set to online mode, the request fails.
- If you launch a request using the "online" connectivity mode, an online request is performed regardless of `NMAApplicationContext` connectivity setting.
- If you launch a request using the "offline" connectivity mode, an offline request is performed regardless of `NMAApplicationContext` connectivity setting.

In all cases if the request fails, no fallback action is automatically performed.

Routing-Related Enumerations

Route calculations make use of HERE SDK enumerations that include:

- `NMARoutingType` enum - represents values describing different routing types such as `NMARoutingTypeFastest` or `NMARoutingTypeShortest`
- `NMATransportMode` enum - represents values describing different transport modes such as `NMATransportModeCar` or `NMATransportModePedestrian`
- `NMARoutingOption` enum - represents values describing special conditions for route calculation such as `NMARoutingOptionAvoidBoatFerry` or `NMARoutingOptionAvoidTollRoad`. Values from this enum are also returned after a route calculation to indicate the options that a route result violates
- `NMARouteViolatedOption` enum - represents values describing special conditions that are violated in a route calculation in addition to `NMARoutingOption`. This enum contains values for blocked roads and turn restrictions. For example, after specifying a route calculation that avoids tolls and ferries, you may get an `NMARoute` with `NMARouteViolatedOptionBlockedRoad` violated option. This indicates that although a route was found, this route goes through at least one blocked road — violating a condition of your route request

  Note: The absence of `NMARouteViolatedOptionBlockedRoad` does not necessarily indicate that no roads are blocked on a route. This is especially true when traffic information is unavailable or not enabled in the route calculation.
- `NMARoutingError` enum - represents values describing possible route calculation errors such as `NMARoutingErrorNone` or `NMARoutingErrorViolatesOptions`

Transit Routing

Transit routes are routes calculated using `NMACoreRouter`, with `NMATransportMode` set to `NMATransportModePublicTransport` in `NMARoutingMode`. With the transit routing feature you can calculate transit routes by using known online timetable information.

You can add a variety of restrictions, such as Avoid Boats, by using `transitRoutingOptions` property. You can also limit the desired number of transit vehicle changes by using `maximumChanges` property.

When a transit route is found, `NMACoreRouter` returns the calculation result block and route result via `calculateRouteWithStops:routingMode:completionBlock:` method. An `NMARoute` contains one or more maneuvers represented by the `NMAManeuver` class. Each of these maneuvers is either a pedestrian or a public transit maneuver. If a maneuver is a public transit maneuver, information specific for transit can be accessed by downcasting the `NMAManeuver` to `NMATransitManeuver`. An `NMATransitManeuver` contains one or more `NMATransitRouteElement` objects. Each `NMATransitRouteElement` object contains the departure station, the arrival station, and travel time of that transit maneuver.

Note: Transit directions are only available in certain areas of the world.

The following is an example of a transit route using `NMACoreRouter`:

```
NMACoreRouter* coreRouter = [[NMACoreRouter alloc] init];

NMAGeoCoordinates* geoCoord1 =
 [[NMAGeoCoordinates alloc] initWithLatitude:49.1966286 longitude:-123.0053635];
NMAGeoCoordinates* geoCoord2 =
 [[NMAGeoCoordinates alloc] initWithLatitude:49.1947289 longitude:-123.1762924];

NMAWaypoint* waypoint1 = [[NMAWaypoint alloc] initWithGeoCoordinates:geoCoord1];
NMAWaypoint* waypoint2 = [[NMAWaypoint alloc] initWithGeoCoordinates:geoCoord2];

NSMutableArray* stops = [[NSMutableArray alloc] initWithCapacity:4];
[stops addObject:waypoint1];
[stops addObject:waypoint2];

NMARoutingMode* routingMode = [[NMARoutingMode alloc]
 initWithRoutingType:NMARoutingTypeFastest
 transportMode:NMATransportModePublicTransport
 routingOptions:0];

[coreRouter calculateRouteWithStops:stops routingMode:routingMode
 completionBlock:^(NMARouteResult *routeResult, NMARoutingError error) {

  // If the route was calculated successfully
  if (!error && routeResult && routeResult.routes.count > 0)
  {
   NMARoute* route = [routeResult.routes objectAtIndex:0];
   // Render the route on the map
   NMAMapRoute* mapRoute = [NMAMapRoute mapRouteWithRoute:route];
   [mapView addMapObject:mapRoute];

   // In order to see the entire route, we orientate the map view
   // accordingly
   [mapView setBoundingBox:route.boundingBox
     withAnimation:NMAMapAnimationLinear];
  }
  else if (error)
  {
   // Display a message indicating route calculation failure
  }
```

```
}];
```

Before displaying transit routes, set the map scheme to include transit so that the `NMAMapRoute` shows the color of the transit lines. You should also use `setBoundingBox:withAnimation:` method to pan and zoom the view to display the entire route.

```
// sets the map scheme to include transit
[mapView setMapScheme:NMAMapSchemeNormalDayTransit];
// zoom to display the entire route
[mapView setBoundingBox:route.boundingBox withAnimation:NMAMapAnimationBow];
```

Bicycle Routing

The bicycle routing feature provides route calculation using car and pedestrian roads with bicycle-specific speed estimations. This type of routing can be performed online or offline, with elevation data being available in an online request.

Note:

- Bicycle routing is currently offered as a beta feature. APIs may change without notice.
- Bike-specific roadways are not yet supported.

Bicycle routing includes pedestrian-only roads and road segments that require traversing a one-way road opposite the allowed direction of travel. When a road is not open for driving in the travel direction, the routing algorithm assumes that the user must walk the bicycle, and therefore it uses the pedestrian walking speed for such segments. As a special exception to this rule, pedestrian segments located in parks are assumed to be open for bicycles, so full bicycle speed is used there. Generally, such walk-only segments are used in bicycle routing only when they provide a big shortcut, or when a waypoint is located on such a segment.

Performing a Bicycle Routing Request

You can perform bicycle routing by using the `NMACoreRouter` class and `NMATransportModeBike` as shown in the following example:

```
// Create an NMACoreRouter
NMACoreRouter* coreRouter = [[NMACoreRouter alloc] init];
NMAGeoCoordinates* geoCoord1 =
    [[NMAGeoCoordinates alloc] initWithLatitude:49.276271 longitude:-123.113224];
NMAGeoCoordinates* geoCoord2 =
    [[NMAGeoCoordinates alloc] initWithLatitude:49.1947289 longitude:-123.1762924];
NMAWaypoint* waypoint1 = [[NMAWaypoint alloc] initWithGeoCoordinates:geoCoord1];
NMAWaypoint* waypoint2 = [[NMAWaypoint alloc] initWithGeoCoordinates:geoCoord2];
NSMutableArray* stops = [[NSMutableArray alloc] initWithCapacity:4];
[stops addObject:waypoint1];
[stops addObject:waypoint2];

NMARoutingMode* routingMode = [[NMARoutingMode alloc]
    initWithRoutingType:NMARoutingTypeFastest
    transportMode:NMATransportModeBike
    routingOptions:0];

[coreRouter calculateRouteWithStops:stops routingMode:routingMode
    completionBlock:^(NMARouteResult *routeResult, NMARoutingError error) {

       // If the route was calculated successfully
      if (!error && routeResult && routeResult.routes.count > 0)
```

```
    {
      NMARoute* bikeRoute = [routeResult.routes objectAtIndex:0];
      // ...
    }
    else if (error)
    {
      // Display a message indicating route calculation failure
    }
  }];
```

Route Elevation

In an online bicycle routing session HERE SDK considers elevation changes when determining what speed should be used on the given road. When going uphill, speed decreases, possibly down to the pedestrian speed. When going downhill, speed increases.

Note: Elevation-based speed estimations may change in the future.

You can also create an elevation profile of a route, similar to the following screenshot, by using altitude data of the points on a calculated route.

Figure 64: Plotted Chart of Elevations

To retrieve this elevation data, use `geometryWithElevationData` property on an `NMARoute` object and inspect `altitude` on each returned `NMAGeoCoordinates` object. If the altitude is not known at that location, the `altitude` property returns `NMAGeoCoordinatesUnknownAltitudeValue`.

Note: Route altitude data is available in online calculated routes for cars, pedestrians, trucks, and bicycles.

Scooter Routing

Scooter routing provides route calculation using car roads with scooter-specific speed estimations. This type of routing can be performed online or offline.

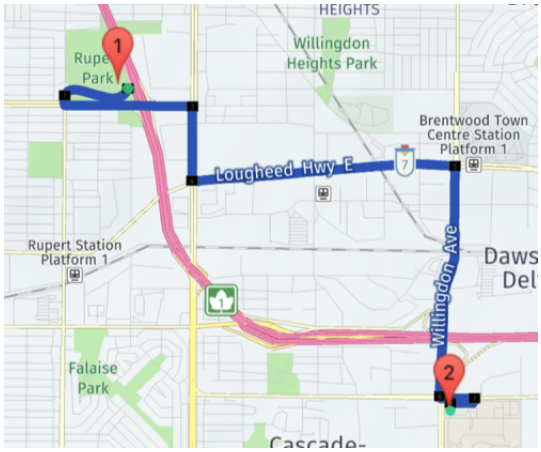
Note: Scooter routing is supported in the following countries only:

- India
- Indonesia
- Singapore
- Taiwan
- Thailand
- Vietnam

In order to perform offline scooter routing, you have to download or prefetch optional data group `NMAMapDataGroupScooterAttributes` such as in the example below. For more information about Data Groups refer to *Map Package Download* on page 59.

```
// Select additional data group needed for offline scooter routing
[[NMAMapLoader sharedMapLoader] selectDataGroup:NMAMapDataGroupScooterAttributes];
// Whether download map package(s)
[[NMAMapLoader sharedMapLoader] installMapPackages:@[package]];
// Or prefetch
[[NMAMapDataPrefetcher sharedMapDataPrefetcher] fetchMapDataForBoundingBox:boudingBox error:nil];
```

Figure 65: Scooter Route



Scooter routing includes car-only roads. Since scooters are not permitted on highways, highways are avoided for this type of route calculation. In cases when forbidden, car-specific roads or highways cannot be avoided, the scooter routing computation fails.

Note: Scooter routing only supports fastest routes.

This feature supports the speed limitation for scooters, which is 45 km/h. Calculated routes take into account the fact that the possible travel speed for scooters is slower than car travel, such as on roads where car travel speed is greater than 45 km/h. In addition, heavy traffic affects scooter travel time less than car travel time.

To calculate a scooter route, use similar steps as for other transport mode types, such as the following example:

```
// Create a NMACoreRouter
NMACoreRouter* coreRouter = [[NMACoreRouter alloc] init];
NMAGeoCoordinates* geoCoord1 =
 [[NMAGeoCoordinates alloc] initWithLatitude:49.276271 longitude:-123.113224];
NMAGeoCoordinates* geoCoord2 =
 [[NMAGeoCoordinates alloc] initWithLatitude:49.1947289 longitude:-123.1762924];

NMAWaypoint* waypoint1 = [[NMAWaypoint alloc] initWithGeoCoordinates:geoCoord1];
NMAWaypoint* waypoint2 = [[NMAWaypoint alloc] initWithGeoCoordinates:geoCoord2];
NSMutableArray* stops = [[NSMutableArray alloc] initWithCapacity:4];
[stops addObject:waypoint1];
[stops addObject:waypoint2];

NMARoutingMode* routingMode = [[NMARoutingMode alloc]
 initWithRoutingType:NMARoutingTypeFastest
 transportMode:NMATransportModeScooter
 routingOptions:0];
```

```
[coreRouter calculateRouteWithStops:stops routingMode:routingMode
 completionBlock:^(NMARouteResult *routeResult, NMARoutingError error) {
  // If the route was calculated successfully
  if (!error && routeResult && routeResult.routes.count > 0) {
   NMARoute* scooterRoute = [routeResult.routes objectAtIndex:0];
   // ...
  } else if (error) {
   // Display a message indicating route calculation failure
  }
 }];
```

Truck Routing

The Truck Routing feature in HERE SDK allows users to calculate routes that can be specifically travelled by trucks and commercial vehicles. Commercial vehicles typically have different regulations for their transportation routes. For example, a government may have laws that restrict trucks carrying flammable materials from travelling in a residential area. By using the Truck Routing feature, you can launch a route calculation that specifically adheres to these restrictions.

In order to perform offline truck routing, you have to download or prefetch optional data group `NMAMapDataGroupTruckAttributes` such as in the example below. For more information about Data Groups refer to *Map Package Download* on page 59.

```
// Select additional data group needed for offline truck routing
[[NMAMapLoader sharedMapLoader] selectDataGroup:NMAMapDataGroupTruckAttributes];
// Whether download map package(s)
[[NMAMapLoader sharedMapLoader] installMapPackages:@[package]];
// Or prefetch map data
[[NMAMapDataPrefetcher sharedMapDataPrefetcher] fetchMapDataForRoute:route radius:radius
 error:&error];
```

Truck Routing and the NMARoutingMode class

The `NMARoutingMode` class contains truck-specific properties that you should set to perform a route calculation. You need to set the route transportation mode to `NMATransportModeTruck` and then optionally set the following route properties before launching the calculation:

- The number of truck trailers
- The truck height
- The truck length
- The truck width
- The maximum allowed truck weight
- The category of tunnels that the truck cannot travel on
- The truck weight per axle
- Hazardous goods that are transported by the truck
- Difficult turns

Note: Truck routing only supports `NMARoutingTypeFastest`. Other routing types are not supported.

A Route Calculation Example

1. As with the previous routing example, create the `NMACoreRouter`, then create an `NSArray` and set its waypoints.
2. Initialize `NMARoutingMode` and set the transport mode to `NMATransportModeTruck`.

```
NMARoutingMode* routingMode = [[NMARoutingMode alloc] initWithRoutingType:NMARoutingTypeFastest
 transportMode:NMATransportModeTruck routingOptions:0];
```

3. Set other truck routing properties.

```
routingMode.vehicleLength = 25.25f;
routingMode.vehicleHeight = 2.6f;
routingMode.trailersCount = 1;
```

4. Calculate the route by calling `calculateRouteWithStops:routingMode:completionBlock:`.

```
[coreRouter calculateRouteWithStops:stops routingMode:routingMode
  completionBlock:^(NMARouteResult *routeResult, NMARoutingError error) {
      // If the route was calculated successfully
      if (!error && routeResult && routeResult.routes.count > 0)
      {
          NMARoute* route = [routes objectAtIndex:0];
          // Render the route on the map
          mapRoute = [NMAMapRoute mapRouteWithRoute:route];
          [mapView addMapObject:mapRoute];
      }
      else if(error)
      {
          // Display a message indicating route calculation failure
      }
  }];
```

Indoor Venue Routing

For more information on 3D venues and indoor routing see *3D Venues* on page 173 and *Venue Routing* on page 181.

Offline Routing

Even without an active data connection the applications developed with HERE iOS SDK are able to request routing information to assist travelling from one location to another.

Your application users do not need to maintain active data connections to calculate routes and render them on a map. It is possible to pre-download updated maps and database information for initiating routing requests while offline. For example, if a user has downloaded offline maps of California and Oregon, a route from San Diego to Portland can be created without any data connection.

For more information about downloading offline maps refer to *Preloading Map Data* on page 57.

Force Online or Offline

You can launch online or offline routing without changing the device or HERE SDK connectivity by using `connectivity` property on an `NMACoreRouter` instance. The `connectivity` property can be set to three possible values:

- If you launch a request using `NMACoreRouterConnectivityDefault` connectivity mode, the request is performed according to `NMAApplicationContext` connectivity setting. If the device is offline while `NMAApplicationContext` is set to online mode, the request fails.
- If you launch a request using `NMACoreRouterConnectivityOnline` connectivity mode, an online request is performed regardless of `NMAApplicationContext` connectivity setting.
- If you launch a request using `NMACoreRouterConnectivityOffline` connectivity mode, an offline request is performed using cached data regardless of `NMAApplicationContext` connectivity setting.

In all cases if the request fails, no fallback action is automatically performed.

To ensure that the connectivity mode is applied, set the `connectivity` property before launching an `NMACoreRouter` calculation request. If an `NMACoreRouterConnectivityOnline` route calculation request fails due to connection issues, HERE SDK returns `NMARoutingErrorNetworkCommunication` error code. If an `NMACoreRouterConnectivityOffline` route calculation request fails due to not enough cached data, HERE SDK returns `NMARoutingErrorGraphDisconnected` error code.

Note: This feature is only applicable to car, bicycle, truck, and pedestrian routing through the `NMACoreRouter` class.

Note: There is no guarantee that online and offline routes will be the same as different algorithms are used for online and offline route calculation. Online route calculation is performed on high performance servers, therefore more computationally intensive algorithms are used online, which cannot be used offline. Online route calculation should be preferred and offline routes are expected to be used as backup especially when there is no connectivity.

Route Consumption

Route consumption calculation is a beta feature which allows the consumption of fuel to be calculated for a given, already calculated route.

Vehicles have a limited range depending on the amount of fuel remaining and how much fuel the vehicle consumes per kilometre under different driving conditions. This is especially important for electric vehicles, which can typically travel less distance on a full charge than a gasoline powered vehicle could travel on a full tank.

To ensure that a vehicle is able to complete a calculated route, HERE iOS SDK can calculate the consumption for each element in that route given certain consumption parameters. This allows the developer to, firstly, determine how much fuel the vehicle will have left at the end of the route and, secondly, check where vehicle would run out of fuel in case if the final destination is not reachable.

Since different vehicles consume different types of fuel (such as gasoline and electricity) and also have different rates of consumption, the consumption calculation must be configured with consumption parameters for a given vehicle before the calculation is performed.

An important concept for consumption calculation is the capacity unit. The capacity unit is how fuel is measured for a given vehicle. Because consumption calculation is general, the unit is effectively defined by the developer. For an electric vehicle the capacity unit is generally kilowatt hours (kWh), for a gasoline powered vehicle it could be liters of gasoline.

A vehicle will have a maximum capacity and current capacity, which indicates how much fuel it can hold in total and the current amount respectively.

In the following description we will assume that we are calculating consumption for an electric vehicle since this is the most common use case. The capacity unit used will be kilowatt hours (kWh).

The consumption parameters are grouped onto the `NMARouteConsumptionParameters` class. The API documentation provides descriptions of each of the individual values that can be modified. Here we will describe the more complex and important values. To begin with, we'll create an empty consumption parameters object.

```
NMARouteConsumptionParamters *consumptionParams = [[NMARouteConsumptionParameters alloc] init];
```

The speed consumption table is the basis for most of the consumption calculation. It is stored in `consumptionSpeed` property. The table represents consumption of the vehicle per metre in kilowatt hours (the capacity unit) for ranges of speeds.

Consider a very simple example with three ranges of speeds and a consumption for each.

Table 3:

Speed range (km/h)	Consumption per metre (kWh)
0 - 30	38.82
31 - 90	18.20
> 90	27.41

From this table we see that our vehicle consumes 38.82 kWh of capacity per metre when travelling between 0 and 30 km/h, 18.20 kWh of capacity per metre when travelling between 31 and 90 km/h, and 27.41 kWh of capacity per metre when travelling at 90 km/h or faster. Of course, this example is very simple, a real world speed consumption table would have finer grained values.

To use this table in our consumption parameters, we set `consumptionSpeed` property using the upper bound of each range as the key and the consumption per metre as the value. The final range (for > 90 km/h) should use the key 250.

```
consumptionParams.consumptionSpeed = @{ @(30):  @(38.82),
     @(90):  @(18.20),
     @(250): @(27.41) };
```

The speed consumption table is enough to calculate simplistic consumption for a route, so let's look at how to do that. Let's assume that we already have an `NMARoute` object called `route` which contains a calculated route. Now let's calculate the consumption for this route.

```
NMARouteConsumption *consumption =
 [route consumptionWithParameters:consumptionParams
     dynamicPenalty:nil];
```

Once we have the route consumption, we can determine the last reachable position on the same route given current capacity of our vehicle. In this example we'll suppose that our vehicle has 100,000 kWh of remaining capacity at the beginning of the route.

```
NMAGeoCoordinates *lastPosition =
 [route lastReachablePositionWithConsumption:consumption
        currentCapacity:100000];
```

The variable `lastPosition` will now contain the last point that our vehicle will be able to reach before the capacity reaches zero. In case if the final destination of the route is reachable, `lastPosition` will be `nil`.

We can also use route consumption to obtain the consumption for each route element that comprise the route itself. The following code snippet prints out the starting and ending geo-coordinates for each route element as well as the consumption for that route element.

```
[route.routeElements enumerateObjectsUsingBlock:^(NMARouteElement *element, NSUInteger idx, BOOL
 *stop) {
        if (idx >= [consumption firstAvailableConsumptionIndex] && [element.geometry count] > 0) {
            NSInteger elementConsumption = [consumption getConsumptionWithIndex:idx];
            NSLog(@"Route element (%.4f, %.4f) --> (%.4f, %.4f) consumption: %zd",
                  element.geometry.firstObject.latitude, element.geometry.firstObject.longitude,
                  element.geometry.lastObject.latitude, element.geometry.lastObject.longitude,
                  consumption);
        }
    }];
```

As mentioned above, setting only `consumptionSpeed` property is sufficient for basic consumption calculation. There are more specialised values which can be set to more closely model the consumption of a real vehicle across different road and traffic conditions. For more details on these properties please see the API documentation.

Route Serialization

Route serialization is a feature that allows users to use `NMARoute` methods to serialize a route into `NSData`, which can be saved as a file. You can also use `NMARoute` to generate a route from a previously serialized route without going through the route calculation process. This is useful when a user wants to recover from a crash during navigation or when a user wants to transfer a route from another device.

Note: Route serialization is currently offered as a beta feature. APIs may change without notice.

Route serialization currently only supports car, bike, truck, and pedestrian routes. Public Transit and Indoor Venue routes cannot be serialized. Route serialization also does not work when the map version from which a route is serialized does not match the current map version. Route serialization also fails if the binary data containing the serialized route is tampered with or corrupted. In these cases a specific `NMARouteSerializationError` error code is returned.

Both asynchronous and synchronous operations are supported. To asynchronously serialize a route, perform the following:

```
NMARoute* route;
// assume that route calculation was already performed

// asynchronous serialization
[NMARoute serializedRouteWithCompletionBlock^(NSData *data, NSError *error) {
 // do something with the data
}];
```

To asynchronously deserialize a route:

```
NSData* data;
// assume 'data' is pointing to a previously serialized route

[NMARoute routeFromSerializedRoute:data withCompletion:^(NMARoute *restoredRoute, NSError *error) {
 // do something with the restored route
}];
```

Import Route

Import Route is a feature which provides an opportunity to create a custom route object from the route shape, particularly from a list of geo-coordinates (lat, lon).

Your application users will be able to import custom routes from external vendors into the SDK and provide route guidance for these routes. As an example, a route is calculated based on a list of geo-coordinates from GPX file previously saved on the device. In any case, the route will be calculated based on our current map data.

Route calculation is based on the given list of points and navigation mode. The navigation mode is used for route calculation between points and for route recalculation during guidance.

A Route Calculation Example

1. First create a new array with the desired points.

```
// Create an array of points
NSMutableArray<NMAGeoCoordinates *> *points = [[NSMutableArray alloc] init];
[points addObject:[[NMAGeoCoordinates alloc] initWithLatitude:52.4992 longitude:13.3956]];
[points addObject:[[NMAGeoCoordinates alloc] initWithLatitude:52.4999 longitude:13.3952]];
[points addObject:[[NMAGeoCoordinates alloc] initWithLatitude:52.5007 longitude:13.3948]];
[points addObject:[[NMAGeoCoordinates alloc] initWithLatitude:52.5015 longitude:13.3945]];
[points addObject:[[NMAGeoCoordinates alloc] initWithLatitude:52.5023 longitude:13.3947]];
[points addObject:[[NMAGeoCoordinates alloc] initWithLatitude:52.5031 longitude:13.3953]];
[points addObject:[[NMAGeoCoordinates alloc] initWithLatitude:52.5030 longitude:13.3961]];
[points addObject:[[NMAGeoCoordinates alloc] initWithLatitude:52.5027 longitude:13.3972]];
[points addObject:[[NMAGeoCoordinates alloc] initWithLatitude:52.5025 longitude:13.3980]];
[points addObject:[[NMAGeoCoordinates alloc] initWithLatitude:52.5020 longitude:13.3980]];
[points addObject:[[NMAGeoCoordinates alloc] initWithLatitude:52.5014 longitude:13.3977]];
```

2. Create a new `NMARoutingMode` object.

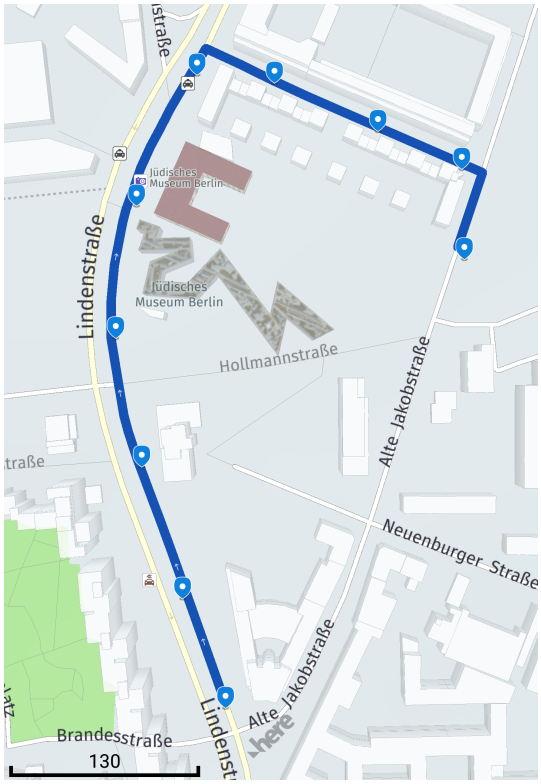
```
// Create the NMARoutingMode and set its transport mode & routing type
NMARoutingMode* routingMode = [[NMARoutingMode alloc] initWithRoutingType:NMARoutingTypeFastest
 transportMode:NMATransportModeCar routingOptions:0];
```

3. Calculate the route by calling `calculateRouteWithPoints:routingMode:completionBlock:`.

```
// Calculate route
[coreRouter calculateRouteWithPoints:points routingMode:routingMode
  completionBlock:^(NMARouteResult *routeResult, NMARoutingError error) {
      // If the route was calculated successfully
      if (!error && routeResult && routeResult.routes.count > 0)
      {
          NMARoute* route = [routes objectAtIndex:0];
          // Render the route on the map
          mapRoute = [NMAMapRoute mapRouteWithRoute:route];
          [mapView addMapObject:mapRoute];
      }
      else if(error)
      {
          // Display a message indicating route calculation failure
      }
```

```
    }];
```

Figure 66: Imported route



Calculating Isoline routing requests

An isoline describes potential routes user can take and provides a polygon indicating the maximum reach in all directions around the given start position by time or distance.

An isoline may consist of several isolated components. This support use cases where the reachable area is disconnected by ferry links.

All islands are returned as separate polygons (components). Ferry links between these polygons are also returned as separate polyline geometries (connections).

The range of reachable using ferries can be displayed as multiple components (with or without connections) or even as a single component.

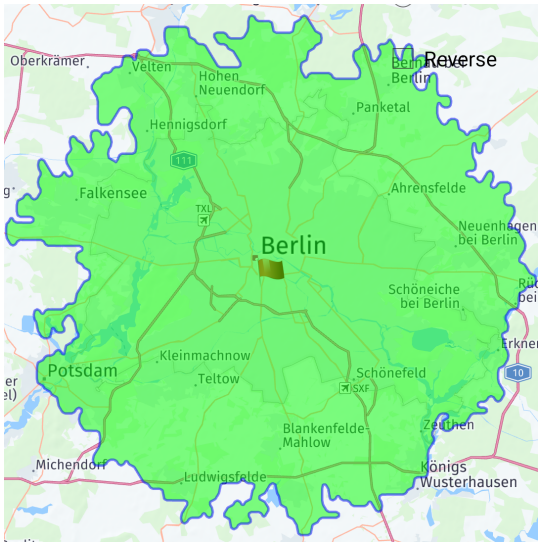
Requesting a distance-based isoline

To calculate a distance-based isoline, specify distance as range type. The range will then be specified in meters. The mode can be either shortest or fastest. A different area can be reached by always driving the fastest possible route, then always driving the shortest possible route. The request below will calculate a 30km isoline around the center of Berlin with the shortest mode. This answers the question: what area can I reach by traveling no more than 30km?

```
NMARoutingMode *routingMode = [[NMARoutingMode alloc] initWithRoutingType:NMARoutingTypeShortest
                                                                transportMode:NMATransportModeCar
                                                                routingOptions:0];
NSMutableArray *ranges = [[NSMutableArray alloc] init];
```

```
[ranges addObject:[NSNumber numberWithInt:30000]];
NMAIsolineOptions *options = [NMAIsolineOptions alloc];
NMAGeoCoordinates *center = [[NMAGeoCoordinates alloc] initWithLatitude:52.4927 longitude:13.4006];
void (^isolineCompletionBlock)(NSArray<NMAIsoline *> * _Nullable, NMAIsolineError)
= ^void(NSArray<NMAIsoline *> * _Nullable isolines, NMAIsolineError error) {
    if (error == NMAIsolineErrorNone) {
    }
};
NMAIsolineRouter *router = [[NMAIsolineRouter alloc] init];
[router calculateFrom:center
          routingMode:routingMode
       isolineOptions:options
            rangeType:NMAIsolineRangeTypeDistance
               ranges:ranges
      completionBlock:isolineCompletionBlock];
```

Figure 67: Example isoline representing a travel distance of 30km from the center of Berlin.

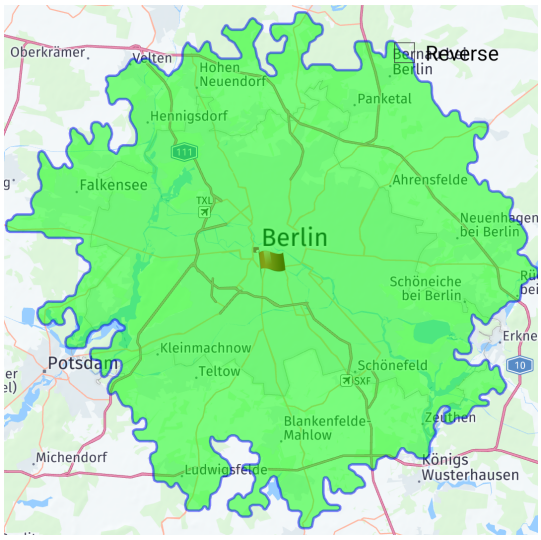


Drivers normally want to drive the fastest possible route. It is possible to calculate an isoline which simulates this behavior while limiting it to a given distance. The request below will calculate a 30km isoline around the center of Berlin with the fastest mode. This answers the question: what area can I reach by traveling no more than 30km while driving the fastest possible route?

```
NMARoutingMode *routingMode = [[NMARoutingMode alloc] initWithRoutingType:NMARoutingTypeFastest
                                                                transportMode:NMATransportModeCar
                                                                routingOptions:0];
NSMutableArray *ranges = [[NSMutableArray alloc] init];
[ranges addObject:[NSNumber numberWithInt:30000]];
NMAIsolineOptions *options = [NMAIsolineOptions alloc];
NMAGeoCoordinates *center = [[NMAGeoCoordinates alloc] initWithLatitude:52.4927 longitude:13.4006];
void (^isolineCompletionBlock)(NSArray<NMAIsoline *> * _Nullable, NMAIsolineError)
= ^void(NSArray<NMAIsoline *> * _Nullable isolines, NMAIsolineError error) {
    if (error == NMAIsolineErrorNone) {
    }
};
NMAIsolineRouter *router = [[NMAIsolineRouter alloc] init];
[router calculateFrom:center
          routingMode:routingMode
       isolineOptions:options
            rangeType:NMAIsolineRangeTypeDistance
               ranges:ranges
```

```
        completionBlock:isolineCompletionBlock];
```

Figure 68: Example isoline representing a travel distance of 30km from the center of Berlin while driving the fastest possible route.



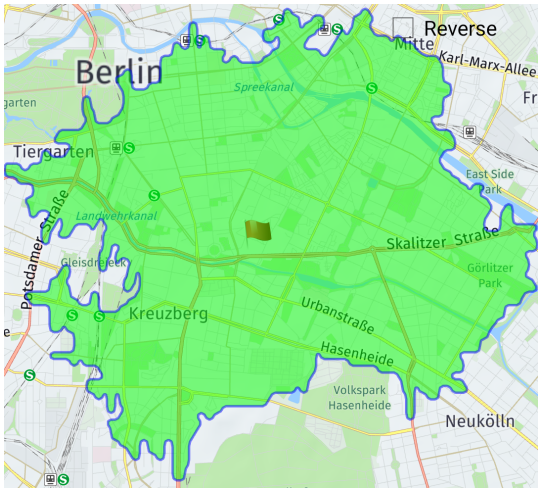
Requesting a time based isoline

To calculate a time-based isoline, specify the time as range type. The range will be specified in seconds. The request below will calculate a 1-hour isoline around the center of Berlin. This answers the question: what area I can reach in one hour from a given position?

```
NMARoutingMode *routingMode = [[NMARoutingMode alloc] initWithRoutingType:NMARoutingTypeFastest
                                                                transportMode:NMATransportModeCar
                                                                routingOptions:0];
NSMutableArray *ranges = [[NSMutableArray alloc] init];
[ranges addObject:[NSNumber numberWithInt:3600]];
NMAIsolineOptions *options = [NMAIsolineOptions alloc];
NMAGeoCoordinates *center = [[NMAGeoCoordinates alloc] initWithLatitude:52.4927 longitude:13.4006];
void (^isolineCompletionBlock)(NSArray<NMAIsoline *> * _Nullable, NMAIsolineError)
= ^void(NSArray<NMAIsoline *> * _Nullable isolines, NMAIsolineError error) {
    if (error == NMAIsolineErrorNone) {
    }
};
NMAIsolineRouter *router = [[NMAIsolineRouter alloc] init];
[router calculateFrom:center
          routingMode:routingMode
       isolineOptions:options
            rangeType:NMAIsolineRangeTypeTime
               ranges:ranges
```

```
        completionBlock:isolineCompletionBlock];
```

Figure 69: Example isoline representing a 1 hour travel time from the center of Berlin.



Setting the level of detail

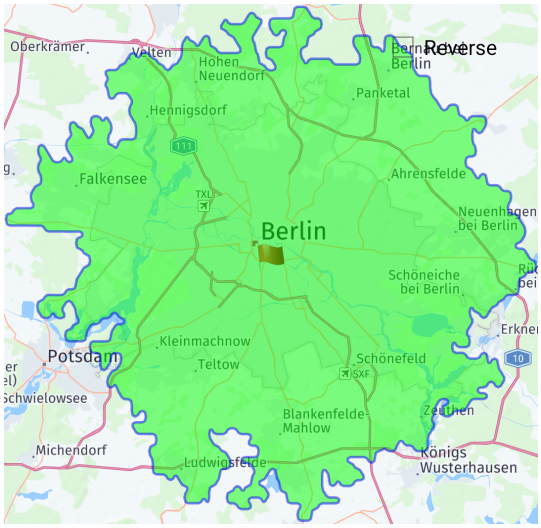
The isoline polygon can contain a lot of points which may be a problem for visualizing it on higher zoom levels.

Below is an example of the 30km isoline calculated in the center of Berlin.

```
NMARoutingMode *routingMode = [[NMARoutingMode alloc] initWithRoutingType:NMARoutingTypeFastest
                                                                   transportMode:NMATransportModeCar
                                                                  routingOptions:0];
NSMutableArray *ranges = [[NSMutableArray alloc] init];
[ranges addObject:[NSNumber numberWithInt:3600]];
NMAIsolineOptions *options = [NMAIsolineOptions alloc];
options.quality = NMAIsolineQualityBest;
NMAGeoCoordinates *center = [[NMAGeoCoordinates alloc] initWithLatitude:52.4927 longitude:13.4006];
void (^isolineCompletionBlock)(NSArray<NMAIsoline *> * _Nullable, NMAIsolineError)
= ^void(NSArray<NMAIsoline *> * _Nullable isolines, NMAIsolineError error) {
    if (error == NMAIsolineErrorNone) {
    }
};
NMAIsolineRouter *router = [[NMAIsolineRouter alloc] init];
[router calculateFrom:center
          routingMode:routingMode
       isolineOptions:options
            rangeType:NMAIsolineRangeTypeDistance
               ranges:ranges
```

```
        completionBlock:isolineCompletionBlock];
```

Figure 70: Example of isoline with many points.



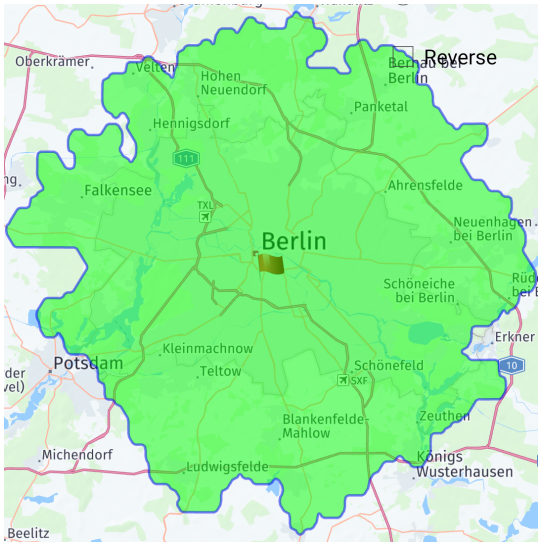
There are two ways to control the level of detail of the isoline polygon. The first one is view resolution. The view resolution allows the client to specify the level of detail needed for visualizing the polygon. It is specified in meters per pixel and reflects the client's zoom level and screen resolution. The isoline service will use this value in order to find the balanced level of detail.

Below is an example of the same isoline request, with view resolution parameter:

```
NMARoutingMode *routingMode = [[NMARoutingMode alloc] initWithRoutingType:NMARoutingTypeFastest
                                                                   transportMode:NMATransportModeCar
                                                                  routingOptions:0];
NSMutableArray *ranges = [[NSMutableArray alloc] init];
[ranges addObject:[NSNumber numberWithInt:3600]];
NMAIsolineOptions *options = [NMAIsolineOptions alloc];
options.quality = NMAIsolineQualityBalanced;
NMAGeoCoordinates *center = [[NMAGeoCoordinates alloc] initWithLatitude:52.4927 longitude:13.4006];
void (^isolineCompletionBlock)(NSArray<NMAIsoline *> * _Nullable, NMAIsolineError)
= ^void(NSArray<NMAIsoline *> * _Nullable isolines, NMAIsolineError error) {
    if (error == NMAIsolineErrorNone) {
    }
};
NMAIsolineRouter *router = [[NMAIsolineRouter alloc] init];
[router calculateFrom:center
          routingMode:routingMode
       isolineOptions:options
            rangeType:NMAIsolineRangeTypeDistance
               ranges:ranges
```

```
        completionBlock:isolineCompletionBlock];
```

Figure 71: Example of isoline with view resolution applied.



The second way is to set the maximum number of points allowed in the isoline. Once a number is provided in the request the service will automatically adjust the level of detail and perform shape simplification in order to fit within the limit.

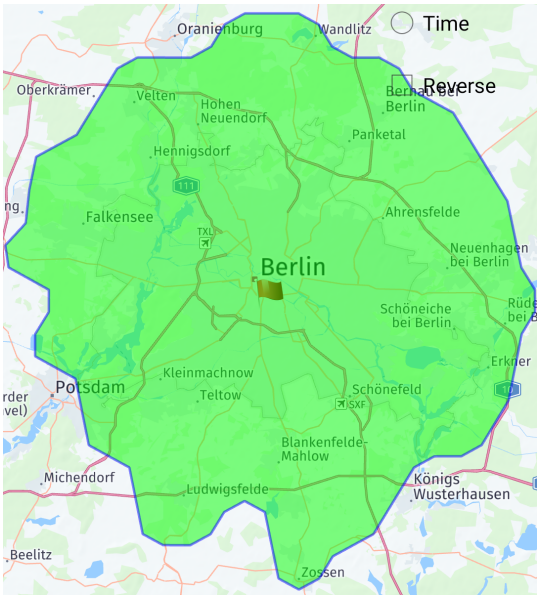
Note: This option should be used if a client has some physical limitation on the number of points it is able to process. Please avoid using this option to control zoom level, use view resolution instead. View resolution and max points can be used at the same time.

Below is an example of the same isoline request, with max points parameter:

```
NMARoutingMode *routingMode = [[NMARoutingMode alloc] initWithRoutingType:NMARoutingTypeFastest
                                                                transportMode:NMATransportModeCar
                                                                routingOptions:0];
NSMutableArray *ranges = [[NSMutableArray alloc] init];
[ranges addObject:[NSNumber numberWithInt:3600]];
NMAIsolineOptions *options = [NMAIsolineOptions alloc];
options.quality = NMAIsolineQualityPerformance;
NMAGeoCoordinates *center = [[NMAGeoCoordinates alloc] initWithLatitude:52.4927 longitude:13.4006];
void (^isolineCompletionBlock)(NSArray<NMAIsoline *> * _Nullable, NMAIsolineError)
= ^void(NSArray<NMAIsoline *> * _Nullable isolines, NMAIsolineError error) {
    if (error == NMAIsolineErrorNone) {
    }
};
NMAIsolineRouter *router = [[NMAIsolineRouter alloc] init];
[router calculateFrom:center
          routingMode:routingMode
       isolineOptions:options
            rangeType:NMAIsolineRangeTypeDistance
               ranges:ranges
```

```
        completionBlock:isolineCompletionBlock];
```

Figure 72: Example of isoline with max points applied.



Electronic Horizon

Electronic Horizon uses the map as a sensor to provide a continuous forecast of the upcoming road network by ingesting the map topography which is currently out of the driver's sight. Together with the vehicle position and other relevant road attributes, the HERE Electronic Horizon API creates a simplified model of the road ahead. By using techniques like adaptive cruise control (ACC) or curve speed warning, you can help the driver to save fuel and to make driving safer and less stressful.

Types of data the HERE Electronic Horizon API can predict include:

- Most probable path
- Alternative side paths
- Speed limits
- Slopes
- Curvature coordinates
- Road classes
- Road types

The HERE Electronic Horizon API fully supports the offline use case and adheres to the vehicle's advanced driver assistance systems (ADAS) standard being ADASISv2 compliant.

Before You Start

Please make sure to use the correct flavor of HERE SDK. Electronic Horizon is included in HERE iOS SDK since version 3.6.0.

Note: Electronic Horizon supports online and offline road network prediction. Although the API is capable of accessing map data from the cloud, it is strongly recommended to download the expected map areas beforehand. All data that would be available online is fully available in

offline mode. For offline usage (via ODML) make sure to enable `NMAMapDataGroupADAS` and `NMAMapDataGroupLinkGDBIdPVId` data groups before downloading the desired map data.

Since the Electronic Horizon requires the user's position, it is mandatory to require `ACCESS_FINE_LOCATION` permission. Although the API can also work in offline mode, it is recommended to request at least the following permissions from the user:

```
<key>NSLocationAlwaysUsageDescription</key>
<string>This is needed to determine your current location</string>
<key>NSLocationWhenInUseUsageDescription</key>
<string>This is needed to determine your current location</string>
```

How to Retrieve Your First Electronic Horizon Events

Before you can access the Electronic Horizon, make sure to start `NMAPositioningManager`:

```
NMAPositioningManager* positioningManager = [NMAPositioningManager sharedPositioningManager];
if ([positioningManager startPositioning]) {
    positioningManager.mapMatchingEnabled = YES;
    positioningManager.mapMatchMode = NMAMapMatchModeCar;
}
```

Then you can create the Electronic Horizon instance and set the delegate to receive events. Make sure your delegate conforms to the `NMAElectronicHorizonDelegate` protocol. The protocol defines several callbacks which are marked as optional. Therefore you are only required to implement the callbacks you really need.

```
self.electronicHorizon = [[NMAElectronicHorizon alloc] init];
self.electronicHorizon.delegate = self;

- (void)electronicHorizon:(nonnull NMAElectronicHorizon *)electronicHorizon
    didReceiveNewPosition:(nonnull NMAEHPosition *)position {
    // handle event here
}
```

Now you are able to tell the Electronic Horizon engine to calculate the road network ahead – based on the current position of your device. You do this by simply calling:

```
[self.electronicHorizon update];
```

Note: Calling `update` method is resource intensive. It is advised not to call this function too frequently. `NMAPositioningManager` usually provides a new position each second. Instead you may want to update the Electronic Horizon only after:

- A certain distance
- A certain amount of time
- A certain event

Since calling `update` can take a while, the `NMAElectronicHorizonDelegate` protocol provides several callbacks to get notified as soon as the predicted path has changed:

- `electronicHorizon:didReceiveNewPosition:` - Always called last after the other callbacks. May contain an unchanged `NMAEHPathTree` if the position is the same as for the previous call. Always contains the full path.
- `electronicHorizonDidReceiveTreeReset:` - As soon as the current position is beyond the main `NMAEHPathTree` look ahead or trailing distance, `electronicHorizonDidReceiveTreeReset` gets

called and a new NMAEHPathTree object is constructed. Otherwise the user is still travelling on the expected paths and the main NMAEHPathTree is instead extended.

- electronicHorizon:didReceivelinkAdded:link: - Notifies which link was just added to an NMAEHPathTree. Note that the main NMAEHPathTree does not contain a parent while an NMAEHPathTree may or may not contain children, no matter how deep it is nested in the hierarchy.
- electronicHorizon:didReceiveLinkRemoved:link: - Called as soon as a link was removed from a parent NMAEHPathTree. Note that it is not guaranteed to be called for every link that was once added to a path tree.
- electronicHorizon:didReceivePathAdded: - A new NMAEHPathTree is added to the existing tree. Since each NMAEHPathTree contains parents and children, you could start to traverse the road network from here although only electronicHorizon:didReceiveNewPosition: is guaranteed to provide the full path.
- electronicHorizon:didReceivePathRemoved: - Called before electronicHorizon:didReceiveNewPosition: as soon as an NMAEHPathTree was removed. Usually this is a good place to remove any map objects from the map that belong to this NMAEHPathTree.
- electronicHorizon:didReceiveChildDetached:child: - A path was detached from a parent because the user has left the main path but is still travelling on a side path. The detached child most likely becomes the new main path.

Make sure to call update after a valid position is provided by the NMAPositioningManager, otherwise no events are delivered. Please note that electronicHorizon:didReceiveNewPosition: is guaranteed to be called after each update request even if no map matched position could be found. The only requirement is a valid position which is automatically fetched by the Electronic Horizon after an update.

Note: It is recommended to call update from a background thread as the execution may take some time.

If the user is off-road, Electronic Horizon cannot calculate a predicted path since a map matched position is required to operate successfully. This can be detected with the following code snippet:

```
- (void)electronicHorizon:(nonnull NMAElectronicHorizon *)electronicHorizon
    didReceiveNewPosition:(nonnull NMAEHPosition *)position {
    NMAEHPathTree* pathTree = position.pathTree;
    if (pathTree == nil) {
        // we seem to be off-road
    }
}
```

An NMAEHPathTree represents the road network ahead and by default also behind. It contains nested NMAEHPathTrees as children and parents but only the root path does not contain a parent NMAEHPathTree. This can be used to find out if an NMAEHPathTree represents the root path. The path belonging to the Position is always the main path on which the user is currently travelling. This may be the root path but it can also happen that the user decides to take a turn into a side path which then represents the new main path containing the root path as parent. When the path is detached and the tree gets restructured, the new main path becomes the root path again.

```
NMAEHPathTree* parentPathTree = pathTree.parent;
if (parentPathTree == nil) {
   // pathTree is the main path and the root path
} else {
   // pathTree is the main path, but not the root path
}
```

Each path tree contains at least one `NMAEHLink`, which represents the smallest segment on the road that can be accessed via the API. Usually the main path of an `NMAEHPathTrees` is split into several `NMAEHLink` which can be uniquely identified by an ID. A road leg can be seen as a sequence of several links.

Predicting the Most Probable Path (MPP)

Without giving a route to follow, Electronic Horizon predicts the most probable path (which acts as root or main path) a user may take. Previous positions may be taken into account. If no previous position is available, the API suggests the most likely path solely based on the available road attributes.

The probability of each path can be extracted like shown below:

```
- (void)electronicHorizon:(nonnull NMAElectronicHorizon *)electronicHorizon
    didReceiveNewPosition:(nonnull NMAEHPosition *)position {
        NMAEHPathTree* pathTree = position.pathTree;
        if (pathTree == nil) {
            // off-road
            return;
        }

        NMAEHPathTree* parentPathTree = pathTree.parent;
        if (parentPathTree == nil) {
             // pathTree is the main path
        }

        // Main path is expected to have a probability of 1 (= 100%)
        float probability = pathTree.probability;
        NSLog(@"Probability of main path: %f", probability);

        for (NMAEHPathTree* childPathTree in pathTree.children) {
            NSLog(@"Probability of side path: %f", childPathTree.probability);
        }
    }
```

If a path tree is predicted with a probability of 0, then it is highly unlikely that a driver may take this turn. Nevertheless Electronic Horizon is including such roads. In some cases an application may want to present such information to the user, such as in the case of one-way streets. Note that all probabilities except for the main path are below 0.5 (= 50%) as a driver is expected to follow the main path with a higher probability.

Setting a Route

Typically you want Electronic Horizon to follow a route which then becomes the MPP. You can directly set a route to the electronic horizon instance:

```
NMARoute* myRoute = [self getRoute];
[self.electronicHorizon setRoute: myRoute];
```

There is no option to unset a route as setting a nil route would lead to an `NSError`. However, as soon as a user might leave the path of the route, the MPP is recalculated based on the new positions.

Note: Only routes with transport mode `NMATransportModeCar` and `NMATransportModeTruck` are supported. Other types throw an `NSError`.

Customize the Electronic Horizon Ahead and Behind

Typically you want to define the kilometres ahead that Electronic Horizon can predict. You can do this by setting an `NSArray` to define up to n nested side paths.

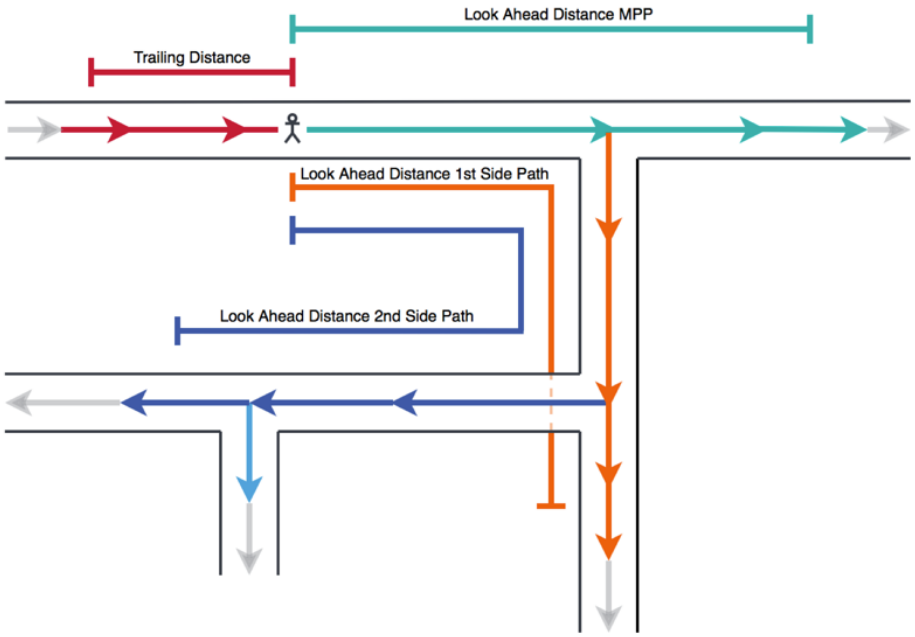
```
int trailingDistanceInMeters = 200;
[self.electronicHorizon setTrailingDistanceInCentimeters: trailingDistanceInMeters * 100];

int mainPathDistanceInMeters = 1000;
int sidePathDistanceInMeters = 500;
NSArray *distances = [NSArray arrayWithObjects:
    [NSNumber numberWithInt: mainPathDistanceInMeters * 100],
    [NSNumber numberWithInt: sidePathDistanceInMeters * 100], nil];
[self.electronicHorizon setLookAheadDistancesInCentimeters: distances];
```

Old links and path tree children are removed only when they are fully behind the defined trailing distance from the current position. If a link is longer than the trailing distance, the link is not removed until its start and end nodes are outside the defined threshold. Therefore it may appear that setting a trailing distance might not have an immediate effect, which is especially valid for longer links. Note that the minimum trailing distance is 1 cm.

The main path has at least one child while the full path tree always contains n + 1 side paths. Although the driver is expected to follow the main path, at each junction alternative paths can be taken. Any road that is branching from the main path is called a 1st level side path. Since side paths can further branch into side paths, all side paths branching from 1st level side paths are called 2nd level side paths, and so on.

As you can see from the illustration below, the distance of each side path starts from the vehicle position. When three look-ahead distances are set (MPP, 1st side path, 2nd side path), then a side path of the 2nd side path is as long as one link. See the light blue arrow below as an example.

Figure 73: Trailing Distance and Look Ahead Distance



In order to understand how the root path tree and its children are generated, it may be helpful to visualize the paths programmatically on a map view as shown below for two look-ahead distances. The full path tree can be retrieved from `electronicHorizon:didReceiveNewPosition:` callback:

```
- (void)electronicHorizon:(nonnull NMAElectronicHorizon *)electronicHorizon
    didReceiveNewPosition:(nonnull NMAEHPosition *)position {
        NMAEHPathTree* pathTree = position.pathTree;
        if (pathTree != nil && [pathTree isEqual: self.previousPathTree]) {
            // same PathTree object, we ignore any added paths
            // and wait for a newly created PathTree
            return;
        }

        self.previousPathTree = pathTree;

        if (pathTree != nil) {
            [self showAllLinksInPathTree: pathTree];
        } else {
            // offroad
        }
}
```

Note: An implementation may want to handle an added path as soon as the existing `NMAEHPathTree` object is extended. In this example we are only interested in a newly created path tree object to visualize the full path tree. Keep in mind that a user is usually travelling along the expected path. This path is then extended instead of being created freshly anew. Same applies for the paths behind the trailing distance which are removed from a parent path tree instead of being reset as part of a full tree reset (see `electronicHorizonDidReceiveTreeReset:`).

Once you can be sure to have a valid path tree, you can start rendering all nested links. The `zIndex` is only used to ensure that deeper paths are rendered on top of the higher ones in the hierarchy:

```
- (void) showAllLinksInPathTree: (NMAEHPathTree *) mainPathTree {
    int zIndex = 0;
    [self addLinksToHereMap: mainPathTree.links
                  withColor: [UIColor redColor]
                   andIndex: zIndex];

    // comment out the below to show only main path
    for (NMAEHPathTree* firstLevelSidePath in mainPathTree.children) {
        [self addLinksToHereMap: firstLevelSidePath.links
                      withColor: [UIColor blueColor]
                       andIndex: ++zIndex];

        for (NMAEHPathTree* secondLevelSidePath in firstLevelSidePath.children) {
            [self addLinksToHereMap: secondLevelSidePath.links
                          withColor: [UIColor greenColor]
                           andIndex: ++zIndex];
            // and so on ...
        }
    }
}

- (void) addLinksToHereMap: (NMAEHLinkRange *) links
                 withColor: (UIColor *) linkColor
                  andIndex: (int) zIndex {
    for (NMAEHLink* link in links) {
        NMAGeoPolyline* geoPolyline = [self.mapAccessor getLinkPolyline: link];
        if (geoPolyline) {
            NMAMapPolyline* mapPolyline = [[NMAMapPolyline alloc] initWithPolyline: geoPolyline];
            [mapPolyline setLineColor: linkColor];
            [mapPolyline setLineWidth: 10];
            [mapPolyline setZIndex: zIndex];
            [self.mapView addMapObject: mapPolyline];
```

```
        }
    }
}
```

By accessing an `NMAGeoPolyline` you can render the path as a map object onto the map as shown above. The path geometry of each link is defined by an `NMAGeoPolyline` you should extract using an `NMAEHMapAccessor`. Since the `NMAGeoPolyline` is not part of the link itself, it must be extracted from the map data. Therefore it is mandatory to check for nil to ensure that the needed map data is available.

At each junction a driver may decide to drive back to where she/he was coming from. These paths have a probability below 50% but are nevertheless a valid choice and are therefore by default not excluded from Electronic Horizon path tree hierarchy. However, in most cases you would like to ignore unlikely turns. For example, with the following code snippet you can detect if a side path is following the parent path in opposite direction:

```
- (bool) isBackTurn: (NMAEHPathTree *) childPathTree {
    NMAEHPathTree* parentPath = childPathTree.parent;
    if (!parentPath) {
        return false;
    }

    long childPathTreeOffset = [childPathTree offsetCentimeters];
    for (NMAEHLink* parentLink in parentPath.links) {
        if ([parentLink endOffsetCentimeters] == childPathTreeOffset) {
            NMAEHLink* firstInChild = childPathTree.links.nextObject;
            return firstInChild.id == parentLink.id &&
                firstInChild.direction != parentLink.direction;
        }
    }
    return false;
}
```

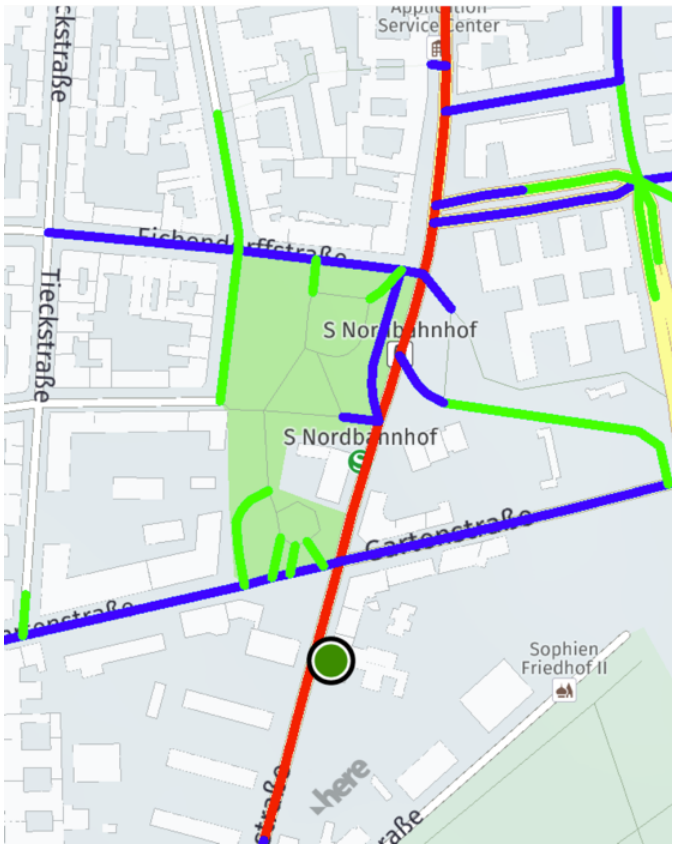
First you compare the end offset distance of the links of the parent path with the offset of the side path (child). Then you can compare the ID of the first link of the children with the ID of the last link of the parent: If the direction is different, then the child must be heading in the opposite direction. Each road segment is represented by a link which is uniquely identified by an ID. Only attributes like direction and / or offsets may be different for the same link. Note that this is just an example which should be further optimized and adapted to your specific needs.

If all goes well, you should see the full path tree rendered as shown below:

- The root path is red
- The 1st level side path is blue

- The 2nd level side path is green

Figure 74: Rendered Root PathTree, 1st and 2nd Side Paths



It is not possible to set a look-ahead distance for the n + 1 child: Therefore the green side path (which is the 2nd level side path) is only as long as one link (the first link on a side path of the blue path). Remember that we have passed only two distance values for the main path (red) and the 1st level side path (blue). By default this results in a 3rd level side path (green) that consists of one link.

Note: The `NMAEHPathTree` you receive with `electronicHorizon:didReceiveNewPosition:` callback always includes the full road network ahead including unlikely paths like, for example, one-way or dead-end streets. Depending on your use case, it may be useful to ignore paths with a low probability or to limit the scope to the main path only - or even to filter out links that follow into the opposite direction like shown above. Make sure not to add too many side paths to the horizon as this would not only increase processing time but also the likelihood of less important links. In most cases it should be sufficient to set only one look-ahead distance as this would by default include first level side paths with a length of one link.

Get Road Attributes

Electronic Horizon offers different road attributes that can be accessed from the available map data via an `NMAEHMapAccessor`. Some attributes like slope data are directly accessible while `NMAEHMetaData` (such as side of driving or unit system) and `NMAEHLinkInformation` (such as form of route) hold multiple attributes. All attributes are valid for the entire link.

Below is an example on how to extract speed limits from `NMAEHLinkInformation`:

```
- (void)logSpeedLimits: (NMAEHPathTree *)pathTree {
    NMAEHMapAccessor* mapAccessor = self.electronicHorizon.mapAccessor;
    for (NMAEHLink* link in pathTree.links) {
```

```
        NMAEHLinkInformation* linkInformation = [mapAccessor getLinkInformation: link];
        if (linkInformation) {
            double speedLimitMetersPerSecond = linkInformation.speedLimitMetersPerSecond;
            NSLog(@"Speed limit on path: %ld Km", lroundl(speedLimitMetersPerSecond * 3.6));
        } else {
           NSLog(@"Map data not available");
        }
    }
}
```

One speed limit is available per link and it is valid for the whole length of the link. In other words: If the speed limit changes in comparison to the previous link, the location of where the speed limit changes equals the start coordinate of the link.

Please refer to the API reference for more road attributes (like functional road classes) which could be extracted in a similar way.

Road attributes that are not part of `NMAEHLinkInformation` must be retrieved directly from `NMAEHMapAccessor`. Below you can find an example on how this could be implemented for road slope data:

```
double lengthInMeters;
NMAEHLinkInformation* linkInformation = [self.mapAccessor getLinkInformation: link];
if (linkInformation) {
    lengthInMeters = linkInformation.lengthMeters;
} else {
    NSLog(@"Map data not available");
    return;
}

NSArray<NMAEHSlopeDataPoint *>* slopeDataPointList = [self.mapAccessor getSlopeDataPoints: link];
if (!slopeDataPointList) {
    NSLog(@"Map data not available");
    return;
}

for (NMAEHSlopeDataPoint* slopeDataPoint in slopeDataPointList) {
    double normalizedPosition = slopeDataPoint.relativePositionOnLink;
    double positionOnLinkInMeters = normalizedPosition * lengthInMeters;
    double percent = slopeDataPoint.slopePercent;
    NSLog(@"Slope data position: %f", positionOnLinkInMeters);

    // if percent is greater/smaller than +/- 100% => Infinity
    if (percent == +INFINITY) {
        NSLog(@"Ascending slope > 45°");
    } else if (percent == -INFINITY) {
        NSLog(@"Downward slope > 45°");
    } else {
        if (percent > 0) {
            NSLog(@"Ascending slope: %f", percent);
        } else {
            NSLog(@"Downward slope: %f", percent);
        }
    }
}
```

While the raw slope data is extracted from `[self.mapAccessor getSlopeDataPoints: link]`, you may also want to retrieve additional data from `NMAEHLinkInformation` such as the total length in metres of the current link. An `NMAEHSlopeDataPoint` holds only the normalized position of the link (for example, 0.0

= Start, 0.5 = Middle, 1.0 = End). In order to find the position of the `NMAEHSlopeDataPoint` relative to the beginning of the link, we should multiply its position by the relative length:

```
double positionOnLinkInMeters = normalizedPosition * lengthInMeters;
```

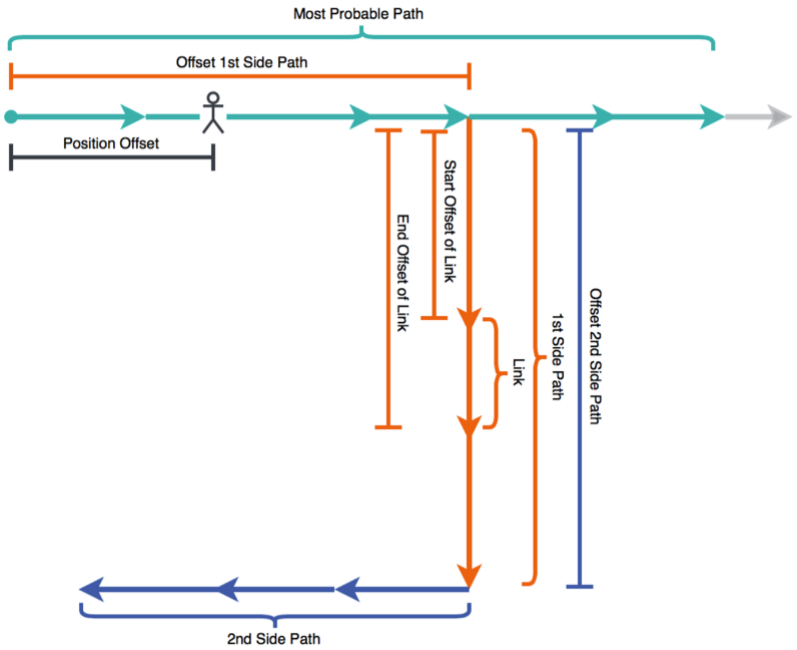
The slope itself is given as percentage value as it would appear on a traffic sign. 100% equals a slope of 45° degrees. Although the world-wide steepest known slope is around 35° degrees, in theory it can happen that a slope is higher than 45° degrees. This is indicated by the Electronic Horizon API as positive or negative infinity as shown above. There can be no percentage value above 100% or below -100%. A positive value indicates an ascending slope while a negative value represents a downward slope. The coordinate of the slope data point can be found by comparing its relative position with the curvature data extracted from the `NMAGeoPolyline` of the link (see above).

Finding the Distance to a Link

Usually an Electronic Horizon implementation may want to inform a driver in advance about upcoming relevant road attributes such as a tunnel ahead. In order to calculate the distance from the current vehicle position to the start of an `NMAEHLink` (for example, when representing such attribute as a Tunnel), the API provides offset values in centimetres.

These offsets are available for each `NMAEHLink`, `NMAEHPathTree`, and the Most Probably Path (MPP). Note that an offset of a side path can be directly retrieved from the side path. It indicates the length from the start of its parent path to the beginning of the side path. Side paths that branch from the MPP contain the length from the start of the MPP to the beginning of the side path. An `NMAEHLink` contains two offsets indicating start and end offset of the link. The current vehicle position is updated with each `update` call and it is indicated by position offset.

Figure 75: Find the Distance to a Link



This can be used to accumulate the offset values to find the distance to any given `NMAEHLink` in relation to the vehicle position. The vehicle position is given by `position.offsetCentimeters`. Note that the

MPP may not be the root path in case the driver has decided to leave the previous MPP. The implementation below iterates through the entire path tree starting from the path that contains the `NMAEHLink` of interest:

```
- (NSUInteger) getDistanceFromVehicleToStartOfLinkInCentimeters: (NMAEHPosition *) position
                    path: (NMAEHPathTree *) pathContainingTheLink
                    link: (NMAEHLink *) link {
    NSUInteger offset = link.startOffsetCentimeters;
    NMAEHPathTree* currentPath = pathContainingTheLink;

    while (currentPath != position.pathTree) {
        offset += currentPath.offsetCentimeters;
        currentPath = currentPath.parent;
    }
    return offset - position.offsetCentimeters;
}
```

If your application wants to inform a driver about upcoming tunnels, you first need to find a link that represents a tunnel. Then you can calculate the distance to that link and inform the user. Note that the distance information is provided with an accuracy down to centimetres, so you may want to convert the distance value to metres by dividing by 100:

```
NMAEHLinkInformation* linkInformation = [self.mapAccessor getLinkInformation: link];
if (linkInformation && linkInformation.isTunnel) {
    NSUInteger distanceFromVehicleToTunnelInCentimeters =
        [self getDistanceFromVehicleToStartOfLinkInCentimeters: self.position path: pathTree link:
 link];
    NSLog(@"Tunnel ahead in: : %ld meters", distanceFromVehicleToTunnelInCentimeters/100);
}
```

Retrieving ADASIS v2 Message Data

The Electronic Horizon API is capable of enriching the vehicle data by providing relevant message data for ADASIS v2 compliant in-car clients. To prepare communication to a client, you first need to provide a message configuration to define the desired types of data. As a next step, you can create an instance of the `NMAAdasisV2Engine` which consumes that configuration.

To receive messages, the delegate class must conform to the `NMAAdasisV2EngineDelegate` protocol which provides `adasisV2Engine:didReceiveAdasisMessage:` callback.

```
- (id)init {
     NMAAdasisV2MessageConfiguration* adasisV2MessageConfiguration =
         NMAAdasisV2MessageConfiguration.createWithAllMessagesEnabled;

     self.adasisV2Engine = [[NMAAdasisV2Engine alloc]
 initWithConfiguration:adasisV2MessageConfiguration];
     self.adasisV2Engine.delegate = self;
 }

 - (void)adasisV2Engine:(nonnull NMAAdasisV2Engine *)adasisV2Engine
didReceiveAdasisMessage:(nonnull NSData *)message {
     // The ADASIS v2 specification defines messages with 8 bytes payload data
     // where the first three bits define the message type followed by the actual content.
     NSLog(@"ADASIS message received. Length: %lu", (unsigned long)message.length);
 }
```

The above creates a configuration which enables all available message types. Alternatively you can create a configuration enabling only default message types by calling:

```
NMAAdasisV2MessageConfiguration* adasisV2MessageConfiguration =
```

```
    NMAAdasisV2MessageConfiguration.createWithDefaultMessagesEnabled;
```

This enables only `POSITION`, `STUB`, `SEGMENT`, and `META-DATA` message types. In total, six different message types are defined by the ADASIS v2 specification:

- `POSITION` message specifies the current position(s) of the vehicle
- `STUB` message indicates the start of a new path that has an origin at an existing one
- `SEGMENT` message specifies the most important attributes of a part of the path
- `PROFILE SHORT` message describes attribute of the path whose value can be expressed in 10 bits
- `PROFILE LONG` message describes attribute of the path whose value can be expressed in 32 bits
- `META-DATA` message contains utility data

The Electronic Horizon API provides getters and setters to dynamically enable or disable each of the messages and its parts containing data (such as slopes) individually like shown below for `SEGMENT` message type:

```
if (adasisV2MessageConfiguration.segmentEnabled) {
    adasisV2MessageConfiguration.segmentEnabled = false;
}
```

You can run a single `NMAAdasisV2Engine` instance along with an `NMAElectronicHorizion` instance but you have to make sure to call `[self.adasisV2Engine update]` in order to trigger a sequence of ADASIS messages based on the current device position. The usage follows the same pattern as described already for the `NMAElectronicHorizion` instance. Therefore it is required to start `NMAPositioningManager` before you are able to receive your first ADASIS v2 compliant messages.

ADASIS v2 assumes a one-way communication between an ADASIS v2 Horizon provider and a client. The Electronic Horizon API ensures that the receiving sequence of messages is compliant to the ADASIS v2 specification, so usually you do not need to parse its content. Note that the ADASIS specification is proprietary. In order to establish a communication to a client, please contact your ADASIS v2 stakeholder. The AdasisV2Engine solely acts as provider and therefore does not receive or parse any information from a contributing client – as this would typically depend on the individual needs of an actual implementation.

Search

Geocoding and Reverse Geocoding

Geocoding and reverse geocoding APIs from HERE iOS SDK allow application developers to offer search functionality for requesting `NMAPlaceLocation` information. Geocoding APIs resolve a free-formatted text query to an `NMAGeoCoordinates`, while reverse geocoding APIs resolve from an `NMAGeoCoordinates` to geographic data such as `NMAAddress`.

`NMAAddress` provides textual address information including house number, street name, city, country, district, and more. It encompasses everything about an address or a point on the map. The `NMAPlaceLocation` class represents an area on the map where additional attributes can be retrieved. These additional attributes include `NMAAddress`, unique identifier, label, location, access locations, and `NMAGeoBoundingBox` for the location.

Geocoding and Reverse Geocoding Example on GitHub

You can find an example that demonstrates this feature at *https://github.com/heremaps/* (Obj-C) and *https://github.com/heremaps/* (Swift).

The NMAGeocoder Interface

The `NMAGeocoder` interface represents a factory class used to instantiate location search requests. Two types of requests are available: `NMAGeocodeRequest` and `NMAReverseGeocodeRequest`.

The NMAGeocodeRequest Interface

The `NMAGeocodeRequest` interface represents an extended `NMARequest`. The `NMAGeocodeRequest` can be created using a combination of a search area and a free text query string. This is known as a "one-box" request. It returns `NMAPlaceLocation` results according to the specified search area and text query. You can specify a search area by providing an `NMAGeoBoundingBox` or a location with a search radius.

The following shows the methods used to create one-box requests:

```
NMAGeocodeRequest* request = [[NMAGeocoder sharedGeocoder] createGeocodeRequestWithQuery:string
      searchArea:geoBoundingBox
      locationContext:geoCoordinates];
```

```
NMAGeocodeRequest* request = [[NMAGeocoder sharedGeocoder] createGeocodeRequestWithQuery:string
      searchRadius:radius
      locationContext:geoCoordinates];
```

The preceding methods return an `NMAGeocodeRequest` object. To perform the request, call its `startWithListener:` method. The parameter of this method is an object which receives the request results; the object must implement the `NMAResultListener` protocol. Once a request is invoked, it can be cancelled using `cancel` method of `NMARequest`, which returns a `BOOL` value indicating whether the result was cancelled successfully. If the `NMAGeocodeRequest` is successful, a list of `NMAGeocodeResult` objects is returned to the listener.

The following code example demonstrates how to use an `NMAGeocodeRequest`:

```
// Implementation of NMAResultListener
@interface NMAGeocodeTest : NSObject<NMAResultListener> {
}
@end
@implementation NMAGeocodeTest

// NMAResultListener protocol callback implementation
- (void)request:(NMARequest*)request
  didCompleteWithData:(id)data
  error:(NSError*)error
{
 if ( ( [request isKindOfClass:[NMAGeocodeRequest class]]) &&
  ( error.code == NMARequestErrorNone ) )
 {
  // Process result NSArray of NMAGeocodeResult objects
  [self processResult:(NSMutableArray *)data];
 }
 else
 {
  // Handle error
```

```
  ...
 }
}

- (void) startSearch
{
 NMAGeoCoordinates *topLeft =
  [[NMAGeoCoordinates alloc]
   initWithLatitude:52.537413 longitude:13.365641];
 NMAGeoCoordinates *bottomRight =
  [[NMAGeoCoordinates alloc]
   initWithLatitude:52.522428 longitude:13.39345];
 NMAGeoBoundingBox *boundingBox =
  [NMAGeoBoundingBox
   geoBoundingBoxWithTopLeft:topLeft bottomRight:bottomRight];

 NMAGeocodeRequest* request = [[NMAGeocoder sharedGeocoder]
 createGeocodeRequestWithQuery:@"100 INVALIDENSTRASSE"
      searchArea:boundingBox
      locationContext:nil];

 // limit the number of results to 10
 request.collectionSize = 10;

 NSError* error = [request startWithListener:self];
 if (error.code != NMARequestErrorNone)
 {
  // Handle request error
  ...
 }
}
@end
```

The NMAReverseGeocodeRequest interface

The `NMAReverseGeocodeRequest` interface represents an extended `NMARequest` used to retrieve `NMAPlaceLocation` data. The request is created using an `NMAGeoCoordinates` as shown below:

```
NMAGeocodeRequest* request = [[NMAGeocoder sharedGeocoder]
 createReverseGeocodeRequestWithGeoCoordinates:geoCoordinates];
```

The above method returns an `NMAReverseGeocodeRequest` object. Reverse geocode requests are used in the same way as regular geocode requests (described in the previous section) but the results are returned as an array of `NMAReverseGeocodeResult` objects.

The following example shows how to create and use an `NMAReverseGeocodeRequest`:

```

// Implementation of NMAResultListener
@interface NMAReverseGeocodeTest : NSObject<NMAResultListener> {
}
@end
@implementation NMAReverseGeocodeTest

// NMAResultListener protocol callback implementation
- (void)request:(NMARequest*)request
 didCompleteWithData:(id)data
 error:(NSError*)error
{
 if ( ( [request isKindOfClass:[NMAReverseGeocodeRequest class]]) &&
  ( error.code == NMARequestErrorNone ) )
 {
  // Process result NSArray of NMAReverseGeocodeResult objects
```

```
  [self processResult:(NSMutableArray *)data];
 }
 else
 {
  // Handle error
  ...
 }
}

- (void) startSearch
{
 // Instantiate an Address object
 NMAGeoCoordinates* vancouver = [[NMAGeoCoordinates alloc] initWithLatitude:49.2849
 longitude:-123.1252];

 NMAReverseGeocodeRequest* request = [[NMAGeocoder sharedGeocoder]
 createReverseGeocodeRequestWithGeoCoordinates:vancouver];

 NSError* error = [request startWithListener:self];
 if (error.code != NMARequestErrorNone)
 {
  // Handle request error
  ...
 }
}

@end
```

By default the reverse geocode request above searches for the closest street address. Alternatively, you can also create a reverse geocoding request in one of the following modes (`NMAReverseGeocodeMode`):

- `NMAReverseGeocodeModeRetrieveAddresses` - Search for the closest street address or addresses (same as above)
- `NMAReverseGeocodeModeRetrieveAreas` - Retrieve the administrative area information for the position provided in the request
- `NMAReverseGeocodeModeRetrieveLandmarks` - Search for landmarks like parks and lakes in the proximity provided in the request
- `NMAReverseGeocodeModeRetrieveAll` - Search for streets, administrative areas, and landmarks. This mode aggregates the results of the previous three modes in one call
- `NMAReverseGeocodeModeTrackPosition` - Retrieve street and address information based on a position and bearing

See the following for an example of how to use this type of request. Note that `bearing` parameter is only used when you use `NMAReverseGeocodeModeTrackPosition`.

```
NMAReverseGeocodeRequest* request = [[NMAGeocoder sharedGeocoder]
 createReverseGeocodeRequestWithGeoCoordinates:geoCoordinates
 mode:NMAReverseGeocodeModeRetrieveLandmarks
 bearing:0];

[request startWithBlock:^(NMARequest *request, id data, NSError *error) {
  if(!error) {
   NSArray *results = (NSArray*)data;
   NMAReverseGeocodeResult *result = nil;
   NSMutableString *address = [NSMutableString new];

   for (NSUInteger i = 0; i < results.count; i ++) {
    result = [results objectAtIndex:i];

    [address appendFormat:@"%ld.\n", i + 1];

    if (result.distance > 0) {
     [address appendFormat:@"distance:%.2f\n", result.distance];
```

```
    }
    if (result.location.address.houseNumber) {
     [address appendFormat:@"houseNo:%@\n", result.location.address.houseNumber];
    }
    if (result.location.address.street) {
     [address appendFormat:@"street:%@\n", result.location.address.street];
    }
    if (result.location.address.city) {
     [address appendFormat:@"city:%@\n", result.location.address.city];
    }
    if (result.location.address.state) {
     [address appendFormat:@"state:%@\n", result.location.address.state];
    }
    if (result.location.address.postalCode) {
     [address appendFormat:@"postalCode:%@\n", result.location.address.postalCode];
    }
    if (result.location.address.countryName) {
     [address appendFormat:@"country:%@\n", result.location.address.countryName];
    }
    if (result.location.timeZone) {
     [address appendFormat:@"timeZone:%@\n\n", result.location.timeZone.name];
    }
   }
  }
  }];
```

Offline Geocoding

Applications developed with HERE iOS SDK can perform offline geocoding, which allows geocode and reverse geocode requests to be performed without an active data connection. This is done automatically when an active data connection is not available as long as the map and database information has been previously downloaded. When a data connection is available, HERE SDK attempts to perform the request online first.

For more information about the APIs introduced and demonstrated in this section refer to the *API Reference documentation*.

Search and Discovery

HERE iOS SDK includes a Places API which provides functionality to search, discover, and obtain more information about places in the real world.

HERE Places helps to determine whether a business meets your needs through reviews and photos from real people. In addition to basic information such as opening hours and contact details, HERE Places can also include editorials from popular guides to help identify the best places for you to visit.

Note that offline search is supported when data connection is unavailable if the data required to perform the search has been previously downloaded.

Steps for Performing a Search

1. Implement the `NMAResultListener` protocol to handle the completion of the search.
2. Create a request using the `NMAPlaces` factory.
3. Invoke the request by calling `NMARequest startWithListener:`.
4. `NMAResultListener request:didCompleteWithData:error:` callback is triggered when the request is finished.

Note: Applications that use the Places API must honor the following prescribed workflow:

1. Search
2. Request for Details
3. Perform Actions

Do not preload results that are linked from a response in order to improve performance as doing so violates HERE guidelines. For more information about usage restrictions consult the *API Implementation Check List* section in the Places RESTful API documentation.

Discovery Requests

HERE Places API supports the following discovery requests. Requests are created through factory methods in `NMAPlaces`.

Request	`NMAPlaces` method	Purpose
Search	`createSearchRequestWithLocation: query:`	Finds places that match user-provided search terms.
Explore	`createExploreRequestWithLocation:searchArea:filters:`	Finds interesting places nearby, or in the map viewport, sorted by popularity. Use this type of request if you are trying to answer the question "What are the interesting places near here?" The results may be optionally restricted to a given set of categories, which acts as a filter in terms of what places get returned.
Here	`createHereRequestWithLocation:filters:`	Helps users identify places at the given location by finding places of interest near a given point, sorted by distance. Use this type of request if you are trying to answer the question "What is near this location?" or "Where am I?" You can use this endpoint to implement features like "check-in" (by identifying places at the user's current position) or "tap to get more information about this place of interest". **Note:** Normally, the closest known places are returned with the Here Discovery request but if the uncertainty in the given position is high, then some nearer places are excluded from the result in favor of more popular places in the area of uncertainty.

Request	NMAPlaces method	Purpose
Around	createAroundRequestWithLocation:searchArea:filters:	Allows users to request places near a given point based on a location precision parameter. The places around that point are returned in order of proximity. This type of request is intended for applications that employ features such as augmented reality where places around the user's location are displayed on a device. It is intended to provide places that are likely to be visible to the user as well as important places that are farther away. The Around request is considered experimental, and its behavior and functionality are still evolving. Check future documentation for updates to this feature.

The following code example demonstrates how to perform a search discovery request. You need to implement the NMAResultListener protocol by implementing request:didCompleteWithData:error callback method, and also initialize the request by calling request startWithListener:.

```
// Sample Search request listener
@interface NMASearchTest : NSObject<NMAResultListener> {
 NMADiscoveryPage* _result;
}
@end
@implementation NMASearchTest

// NMAResultListener protocol callback implementation
- (void)request:(NMARequest*)request
  didCompleteWithData:(id)data
  error:(NSError*)error
{
 if ( ( [request isKindOfClass:[NMADiscoveryRequest class]]) &&
  ( error.code == NMARequestErrorNone ) )
 {
  // Process result NMADiscoveryPage objects
  _result = (NMADiscoveryPage*) data;
 }
 else
 {
  // Handle error
  ...
 }
}
- (void) startSearch
{
 // Create a request to search for restaurants in Vancouver
 NMAGeoCoordinates* vancouver =
 [[NMAGeoCoordinates alloc] initWithLatitude:48.263392
     longitude:-123.12203];

 NMADiscoveryRequest* request =
 [[NMAPlaces sharedPlaces] createSearchRequestWithLocation:vancouver
     query:@"restaurant"];

 // optionally, you can set a bounding box to limit the results within it.
```

```
 NMAGeoCoordinates *boundingTopLeftCoords = [[NMAGeoCoordinates alloc] initWithLatitude:49.277484
 longitude:-123.133693];
 NMAGeoCoordinates *boundingBottomRightCoords = [[NMAGeoCoordinates alloc]
 initWithLatitude:49.257209 longitude:-123.11275];
 NMAGeoBoundingBox *bounding = [[NMAGeoBoundingBox alloc] initWithTopLeft:boundingTopLeftCoords
 bottomRight:boundingBottomRightCoords];

 request.viewport = bounding;

 // limit number of items in each result page to 10
 request.collectionSize = 10;

 NSError* error = [request startWithListener:self];
 if (error.code != NMARequestErrorNone)
 {
  // Handle request error
  ...
 }
}
@end
```

To ensure that your application gets the best search results, you can set a location context to your search request by setting a bounding box to `viewport` property. In the previous example you can also replace the `NMAGeoBoundingBox` with the viewport from `NMAMapView`.

The result of a search or explore discovery request is an `NMADiscoveryPage`. The `NMADiscoveryPage` represents a paginated collection of items from which the following can be retrieved:

- Next page request - an `NMADiscoveryRequest` used to retrieve additional pages of search items
- Items for the current page - an `NSArray` of `NMALink`, either `NMAPlaceLink` or `NMADiscoveryLink`

If `NMADiscoveryPage.nextPageRequest` is nil, no additional results are available.

The following is an example:

```
...
@interface NMANextPageTest : NSObject<NMAResultListener> {
 NMADiscoveryPage* _page;  // valid NMADiscoveryPage instance
}
@end
@implementation NMANextPageTest
- (void)onNextPageAction
{
 NSError* error = [_page.nextPageRequest startWithListener:self];
 if ( error.code == NMARequestErrorNone )
 {
  // More data is available
 }
}

// NMAResultListener protocol callback implementation
- (void)request:(NMARequest*)request
  didCompleteWithData:(id)data
  error:(NSError*)error
{
 if ( ( [request isKindOfClass:[NMADiscoveryRequest class]] ) &&
  ( error.code == NMARequestErrorNone ) )
 {
  // Process NMADiscoveryPage objects
 }
 else
 {
  // Handle error
```

```
  ...
 }
}
...
@end
```

`NMADiscoveryPage discoveryResults` property contains an array of `NMALink` objects. The items are actually a collection of `NMALink` subclasses:

- `NMAPlaceLink` - Represents discovery information about an `NMAPlace`. The `NMAPlaceLink` contains a brief summary about a place. Details about a place are available from the `NMAPlace` that the `NMAPlaceLink` references.
- `NMADiscoveryLink` - Represents a discovery-related API link used to retrieve additional `NMADiscoveryPage` instances. This type of `NMALink` can be a result item in an Explore or Here type of search. The `NMADiscoveryLink` references refine discovery requests resulting in more specific results. For example, the `NMADiscoveryLink` may link to a discovery request to search for 'Eat & Drink', 'Going Out', 'Accommodation', and so on.

It is recommended that each type of `NMADiscoveryPage` be checked before it is used. In the following example it is shown how an `NMAPlace` is retrieved through an `NMAPlaceLink`:

```
@interface NMASearchTest : NSObject<NMAResultListener> {
    NMADiscoveryPage* _result;
}
@end
@implementation NMASearchTest
// Retrieve the place details when the user selects a displayed PlaceLink.
- (void)onPlaceLinkSelected:(NMAPlaceLink*)placeLink
{
 NSError* error = [[placeLink detailsRequest] startWithListener:self];
 if ( error.code == NMARequestErrorNone )
 {
  // More data will available.
  ...
 }
}

// NMAResultListener protocol callback implementation
- (void)request:(NMARequest*)request
  didCompleteWithData:(id)data
  error:(NSError*)error
{
    if ( ( [request isKindOfClass:[NMADiscoveryRequest class]]) &&
         ( error.code == NMARequestErrorNone ) )
 {
  _result = (NMADiscoveryPage*) data;
  NSArray* discoveryResult = _result.discoveryResults;

  for ( NMALink* link in discoveryResult )
  {
   if ( link isKindOfClass:[NMADiscoveryLink class] )
   {
    NMADiscoveryLink* discoveryLink = (NMADiscoveryLink*) link;

    // NMADiscoveryLink can also be presented to the user.
    // When a NMADiscoveryLink is selected, another search request should be
    // performed to retrieve results for a specific category.
    ...
   }
   else if ( link isKindOfClass:[NMAPlaceLink class] )
   {
```

```
    NMAPlaceLink* placeLink = (NMAPlaceLink*) link;

    // NMAPlaceLink should be presented to the user, so the link can be
    // selected in order to retrieve additional details about a place
    // of interest.
    ...
   }
  }
 }
 else  if ( ( [request isKindOfClass:[NMAPlaceRequest class]]) &&
            ( error.code == NMARequestErrorNone ) )
 {
  NMAPlace* place = (NMAPlace*)data;
  // Access to additional details about a place of interest.
 }
 else
 {
  // Handle error
  ...
 }
}
@end
```

Search Example on GitHub

You can find an example that demonstrates this feature at *https://github.com/heremaps/* (Obj-C) and *https://github.com/heremaps/* (Swift).

The NMAPlace Class

The `NMAPlace` class represents a detailed set of data about a physical place acting as a container for various attributes, collections of media about a place, and key-value pairs of related places. An `NMAPlace` object can belong to a specific `NMACategory` and has attributes such as:

- A unique identifier (ID)
- A name
- An `NMAPlaceLocation` object representing the physical location of the place. `NMAPlaceLocation` also contains a street address and a list of the geocoordinate positions to access this place
- An array of `NMACategory` objects that link to the categories assigned to the place
- An `NMALink` object containing a link to the origin of supplied information, typically a website of the supplier
- An `NSString` representing a URL to an icon (optional) In an offline search your application should provide the place icon.
- Optional information such as related places, user ratings, reviews, and other editorial media

Category Filters

A place of interest can be associated with categories such as museum, restaurant, and coffee shop. While creating an Explore or Here discovery request, you can choose to provide category filters to get a more specific set of results. For example, you may want to search for sushi restaurants near Vancouver city hall.

To get a list of categories, call `topLevelCategories` method in `NMAPlaces`. From this list of categories you can then retrieve one or more levels of sub-categories. For example, "Bars/Pubs" under the "Restaurant" category. Once you have the categories, you can then create an `NMACategoryFilter` object and call

`addCategoryFilterFromUniqueId` method. Note that each `NMACategoryFilter` object can represent multiple categories.

```
NSArray* categories = [[NMAPlaces sharedPlaces] topLevelCategories];

for (id category in categories)
{
 if (category.uniqueId == "restaurant" )
 {
  NMACategory* restCategory = category;
  NMAGeoCoordinates* vancouver = [[NMAGeoCoordinates alloc]
          initWithLatitude:47.592229
          longitude:-122.315147];

  NMACategoryFilter *categoryFilter = [NMACategoryFilter new];
  [categoryFilter addCategoryFilterFromUniqueId:restCategory.uniqueId];

  NMADiscoveryRequest* request = [ [NMAPlaces sharedPlaces]
        createHereRequestWithLocation:vancouver
        filters:categoryFilter];
  //...
 }
}
```

Note: The category list is retrieved and saved to cache automatically when the user creates an instance of `NMAPlaces`. However, if the user changes the locale or destroys `NMAPlaces` without a previously cached list and restarts the device in offline mode, then the cache may not automatically update when the device becomes online again.

To remedy this scenario, call `refreshTopLevelCategoriesWithCompletion` method to manually update the category list. You can verify that the cateogry list has been updated if `topLevelCategories` method does not return `nil`.

Text AutoSuggestion Requests

HERE Places Search API also supports text autosuggestion requests. This type of request is used for retrieveing a list of instant results (`NMAAutoSuggestTypePlace`) and refined search links (`NMAAutoSuggestTypeSearch`) that are related to a specified location context and a partial search term. For example, if you make a request with the String "rest" in Berlin, the results then contain search terms such as "Restaurant", "Rest area", and "Restorf, Höhbeck, Germany".

Note: Text AutoSuggestion is currently offered as a beta feature. APIs may change without notice. Offline requests are not supported.

To use text autosuggestions, implement a listener to handle a list of `NMAAutoSuggest` and call `createAutoSuggestionRequestWithLocation:partialTerm:` as follows:

```
// Sample Search request listener
@interface NMATextAutoSuggestionSearchTest : NSObject<NMAResultListener> {

}
@end
@implementation NMATextAutoSuggestionSearchTest

// NMAResultListener protocol callback implementation
- (void)request:(NMARequest*)request
  didCompleteWithData:(id)data
   error:(NSError*)error
{
```

```
 if ( ( [request isKindOfClass:[NMAAutoSuggestionRequest class]]) &&
   ( error.code == NMARequestErrorNone ) )
 {
  // Results are held in an array of NMAAutoSuggest objects
  // You can then check the subclass type using the NMAAutoSuggest.type property
  NSArray* textAutoSuggestionResult = (NSArray*) data;
 }
 else
 {
  // Handle error
  ...
 }
}
- (void) startSearch
{
 NMAGeoCoordinates* vancouver =
   [[NMAGeoCoordinates alloc] initWithLatitude:47.592229
      longitude:-122.315147];

 NMAAutoSuggestionRequest* request =
   [[NMAPlaces sharedPlaces] createAutoSuggestionRequestWithLocation:vancouver
      partialTerm:@"rest"];

 // limit number of items in each result page to 10
 request.collectionSize = 10;

 NSError* error = [request startWithListener:self];
 if (error.code != NMARequestErrorNone)
 {
  // Handle request error
  ...
 }
}
@end
```

You can retrieve the results of `NMAAutoSuggestionRequest` by first checking `NMAAutoSuggest` object type as shown in the following example. If the object is `NMAAutoSuggestTypeSearch`, it contains additional paginated results through its `NMADiscoveryRequest` object. If the object is `NMAAutoSuggestTypePlace`, you can request for more details through its `NMAPlaceRequest`.

```
+(BOOL)checkAutoSuggestResults:(NSArray*)array
{
 for (id item in array) {
  NMAAutoSuggestType type = ((NMAAutoSuggest*)item).type;
  NSString *typeString;
  switch (type){
   caseNMAAutoSuggestTypePlace:
   {
    typeString = @"Place";
   }
    break;
   caseNMAAutoSuggestTypeSearch:
   {
    typeString = @"Search";
   }
    break;
  }

  NSLog(@"Type = %@", typeString);

  NMAAutoSuggest* suggestItem = (NMAAutoSuggest*)item;

  //Retrieve information such as suggestItem.title

  if (type == NMAAutoSuggestTypePlace) {
```

```
    NMAAutoSuggestPlace* suggestPlace = (NMAAutoSuggestPlace*)item;

    //Retrieve information such as suggestPlace.vicinityDescription

    NMAPlaceRequest* detailsRequest = suggestPlace.placeDetailsRequest;

    // Get NMAPlaceResult by calling detailsRequest startWithListener:
    // ...

   } else if (type == NMAAutoSuggestTypeSearch) {

    NMAAutoSuggestSearch* suggestSearch = (NMAAutoSuggestPlace*)item;

    //Retrieve information such as suggestSearch.position

    NMADiscoveryPage* discoveryPage;
    NMADiscoveryRequest* discoveryRequest = suggestSearch.suggestedSearchRequest;

    // Get discoveryPage by calling [discoveryRequest startWithListener:]
    // ...
   }
  }
  return YES;
}
```

External References

A place of interest can contain a reference to a foreign system outside of HERE SDK. For example, an `NMAPlace` representing a restaurant may also contain an external reference to an entry in a restaurant review service. Each external reference in `NMAPlace` is tied to a reference source, and each reference can contain one or multiple identifiers.

The following external reference sources are supported in `NMADiscoveryRequest` and `NMAPlaceRequest`

- `NMAPlacesSourcePVID` - Source for HERE Core Maps POI data
- `"yelp"` - Source for Yelp IDs

An external reference is returned in the form of one or multiple `NSString` identifiers in `NMAPlace`, `NMAPlaceLink`, or `NMAPlaceLocation`. To request for a reference, you need to add a source to the `NMARequest` (such as a discovery request) and then retrieve the reference from the results using the same source name. For example,

```
// Create a request to search for restaurants in Vancouver
NMAGeoCoordinates* vancouver =
[[NMAGeoCoordinates alloc] initWithLatitude:48.263392
    longitude:-123.12203];
NMADiscoveryRequest* request =
[[NMAPlaces sharedPlaces] createSearchRequestWithLocation:vancouver
    query:@"restaurant"];

// We also want to retrieve the Yelp ID external reference
[request addSource:@"yelp"];

request.collectionSize = 10;
NSError* error = [request startWithListener:self];
```

After the request execution is complete, you can retrieve the results by using `referenceIdentifiersForSource:` method as shown in the following:

```
...
if ( resultLink isKindOfClass:[NMAPlaceLink class] )
{
 NMAPlaceLink* placeLink = (NMAPlaceLink*) resultLink;

 NSArray* yelpIds =
  [placeLink referenceIdentifiersForSource:@"yelp"];

 for(NSString* id in yelpIds) {
     NSLog(id); // retrieved Yelp ID
    }
}
...
```

Additional external reference sources are supported through details requests as shown in the next example. For example, you can use `detailsRequest` property on `NMAPlaceLink` to retrieve reference IDs by executing the details request.

Note: You can retrieve the following external references through a details request. Sources marked with [*] cannot be used with `createLookupRequestWithReferenceIdentifier:inSource:` method.

- `NMAPlacesSourcePVID` - Source for HERE Core POI
- `NMAPlacesSourceVenuesAll` - Source for all types of HERE Venue IDs
- `NMAPlacesSourceVenuesVenue` - Source for HERE Venue IDs
- `NMAPlacesSourceVenuesContent` - Source for HERE Venue Content IDs
- `NMAPlacesSourceVenuesDestination` - Source for HERE Venue Destination IDs
- `NMAPlacesSourceBuilding` - Source for HERE Building IDs

```
NMAPlaceRequest* placeDetailsRequest = receivedPlaceLink.detailsRequest;
[placeDetailsRequest addSource:NMAPlacesSourcePVID];
[placeDetailsRequest addSource:NMAPlacesSourceVenuesAll];
[placeDetailsRequest addSource:@"yelp"];
[placeDetailsRequest addSource:@"tripadvisor"];

[placeDetailsRequest startWithBlock:^(NMARequest *request, id data, NSError *error) {
 if (!error && data) {
  NMAPlace *place = (NMAPlace*)data;

  NSArray *pvids = [place referenceIdentifiersForSource:NMAPlacesSourcePVID];
  NSArray *venueIds = [place referenceIdentifiersForSource:NMAPlacesSourceVenuesAll];
  NSArray *yelpIds = [place referenceIdentifiersForSource:@"yelp”];
  NSArray *tripadvisorIds = [place referenceIdentifiersForSource:@"tripadvisor”];
 }
}];
```

You can also use an external PVID or Venues reference in the reverse scenario to retrieve a particular `NMAPlace` by using `createLookupRequestWithReferenceIdentifier:inSource:` method. For example:

```
NMAPlaceRequest* request = [ [NMAPlaces sharedPlaces]
 createLookupRequestWithReferenceIdentifier:@"1126226306"
 inSource:NMAPlacesSourcePVID ];
```

Hybrid Places Search

To support offline search, HERE SDK is preloaded with a database of Places which contains a smaller subset of Places as compared to the online search. When internet connectivity is not available and a place search is performed with the default connectivity mode, only basic Places information is returned for the entries in this database (without rich data such as ratings and reviews). However, once internet connectivity is reestablished, HERE SDK then retrieves online Places results again.

Offline Search

Force Online or Offline

You can launch online or offline search without changing the device or HERE SDK connectivity by using `connectivity` property on an `NMARequest` instance. This property is applicable to all `NMARequest` subclasses except `NMAAutoSuggestionRequest`, which can only be used online.

Note: In HERE SDK v3.4 we have updated the behavior of the routing connectivity modes (`NMARequestConnectivity`) to be more consistent with other parts of the SDK.

`connectivity` property can be set to three possible values:

- If you launch a request using `NMARequestConnectivityDefault` connectivity mode, the request is performed according to `NMAApplicationContext` connectivity setting. If the device is offline while `NMAApplicationContext` is set to online mode, the request fails.
- If you launch a request using `NMARequestConnectivityOnline` connectivity mode, an online request is performed regardless of `NMAApplicationContext` connectivity setting.
- If you launch a request using `NMARequestConnectivityOffline` connectivity mode, an offline request is performed regardless of `NMAApplicationContext` connectivity setting.

In all cases if the request fails, no fallback action is automatically performed.

To ensure that the connectivity mode is applied, set `connectivity` property before executing an `NMARequest`.

Depending on your search request and the connectivity mode you can get a few different errors:

- If an `NMARequestConnectivityOnline` search request fails due to connection issues, HERE SDK returns `NMARequestErrorUnknown` error code.
- If an `NMARequestConnectivityOnline` Geocoding or Reverse Geocoding request fails due to connection issues, HERE SDK returns `NMARequestErrorNetworkCommunication` error code.
- If an `NMARequestConnectivityOffline` search request fails due to not enough cached data, HERE SDK returns zero results.
- If you attempt to start an `NMAAutoSuggestionRequest` with `NMARequestConnectivityOffline` connectivity mode, HERE SDK returns `NMARequestErrorNotSupported` error code since auto suggestions are only supported online.

Custom Locations and Geometries

This section describes the Custom Locations feature. This feature allow user-defined locations and geometries to be retrieved through different search requests.

You can find more information about using Custom Locations, including how to import locations and how to manage location layers, by using the Custom Locations API Developer's Guide and the Custom Location Extension User Guide on *http://developer.here.com*.

Custom Location Extension 2

Custom Location Extension 2 (CLE2) allows you to easily distribute custom geospatial information in your mobile applications. Through CLE2 you can programatically insert spatial content to the local database and upload it to the server for data sharing purposes. You can also also perform online or offline searches. These features effectively turns HERE iOS SDK into a lightweight spatial storage solution that enables insertion and query for geospatial information using optimized algorithms.

The classes that support this feature are located under *NMA Custom Location Extension 2* group. Instead of having specific interfaces for location and geometry requests CLE2 unifies all use cases in one flexible approach: the returned value always contains one of the geometry types (such as `NMACLE2GeometryPoint`), along with a set of 0 to N user-defined attributes that can represent any information. There is no implied structure in these attributes. These attributes are made available as a map of key and attribute values.

Some examples of how you can use these CLE2 features include:

- Show all users' custom Points of Interest (POIs) within a 2km radius.
- Online or offline search for all customer offices within Germany using an area defined by a polygon, then display the offices' reception phone numbers, employee counts, and other details.
- Edit geometry shapes in real time in offline mode and perform queries against them to get notifications when such shapes intersect with other existing fixed shapes and other basic Geofencing solutions. For example, this can be a 'moving platform', e.g. ships near ship docks, where locations are relative to GPS movements.
- Sharing Points of Interest that are not officially available as part of HERE map data, such as city facilities and outdoor basketball courts.
- Persist GPS data that is tied to arbitrary data, e.g. hiking trails with speed, even during offline mode.
- Search for specific types of objects that are near a given route.

Layers and Filtering

All data is organized in the form of layers. When uploading, storing, or searching for information, a layer name string is specified and can be used to better filter relevant information.

Further filtering is possible by checking the geometry attributes. These attributes are user-defined fields that are linked to a geometry, such as `NMACLE2GeometryPoint`, and can be text or number fields.

Note: CLE2 search is restricted per layer by app credentials. To manage the access restriction of a Custom Location layer, contact your Custom Location administrator. If you do not have one, see *Service Support* on page 11 to contact us for more details .

Inserting and Uploading Data

To upload CLE2 data, use the web interface or REST APIs. Refer to the following User Guide for more details: *https://developer.here.com/documentation/versions*.

It is also possible to insert data locally and to the server via HERE iOS SDK. HERE SDK makes it straightforward to generate any location-referenced data even when storing it locally offline and sequentially sharing that information to other devices when a connection is established.

Performing Spatial Searches

To perform a search, choose one of the search types as shown below. A common input parameter to all requests is the searched layer name.

Table 4: Search Classes

Search Type	Description	Class Name
Proximity	Retrieve geometries that are within a given radius from a center.	`NMACLE2ProximityRequest`
Corridor	Retrieve geometries along a route specified by a sequence of coordinates.	`NMACLE2CorridorRequest`
Bounding box	Retrieve geometries within a specified rectangular geographic area.	`NMACLE2BoundingBoxRequest`
Quadkey	Retrieve geometries that fall within a specified QuadKey.	`NMACLE2QuadkeyRequest`
Attribute	Retrieve all geometries that match with a specified query. This type of search is only available online.	`NMACLE2AttributeRequest`

Each of the search request types supports some common properties as listed below.

Table 5: Common Request Arguments

Property	Description	Example Values
`geometryType`	Specifies geometry type to be given in the result (online only), see details below on "Understanding the search results".	• `NMACLE2GeometryFull` • `NMACLE2GeometryLocal` • `NMACLE2GeometryNone`
`cachingEnabled`	Default is False. If enabled, geometries received from such online search request will be stored locally.	
`query`	Currently available for online requests only, this property allows a query filter to be specified on the user's geometry attributes so that only geometries that pass the filter were returned. Accepted strings are free form text with simple equality and comparison operators.	

Once you have a search request object created and set up according to your needs, call its `startWithListener:` or `startWithBlock:` methods. The result of the search will be delivered to the provided block or listener. You can get the geometries that matched search criteria from a `CLE2Result` object by calling `getGeometries()`. This list of geometry results may contain objects of the following types:

Table 6: Geometry Return Types

Class	Geometry Description	Main Properties
`NMACLE2Geometry`	Base class for all other geometry return values containing user-defined attributes.	`NSDictionary<NSString *, NSString *> *attributes`
`NMACLE2GeometryPoint`	Represents a point in coordinates. Relates to a `Point` in WKT.	`NMAGeoCoordinates *coordinates`
`NMACLE2GeometryMultiPoint`	Represents a multi-point as a coordinates array. Relates to a `MultiPoint` in WKT.	`NSArray<NMAGeoCoordinates *> *coordinatesArray`
`NMACLE2GeometryPolyline`	Represents a polyline as an `NMAGeoPolyline`. Relates to a WKT `LineString` object.	`NMAGeoPolyline *geoPolyline`
`NMACLE2GeometryMultiPolyline`	Represents a multi-polyline as an array of `NMAGeoPolyline`. Relates to a WKT `MultiLineString` object.	`NSArray<NMAGeoPolyline *> *multiPolylineArray`
`NMACLE2GeometryPolygon`	Represents a polygon with an `NMAGeoPolygon` for the outer ring and an array of `NMAGeoPolygon` for inner holes. Relates to a WKT polygon object containing all rings of this geometry.	`NMAGeoPolygon *outerRing`, `NSArray<NMAGeoPolygon *> *innerRings`
`NMACLE2GeometryMultiPolygon`	Represents a multi-polygon as an array of `NMACLE2GeometryPolygon`. Relates to a `MultiPolygon` object in WKT.	`NSArray<NMACLE2GeometryPolygon *> *multiPolygonArray`

In OpenGIS (the implementation standard for Geographic Information) and WKT representation formats the concept of a polygon is defined by one outer ring polygon plus zero or more inner hole polygons. This is the reason why the class `NMACLE2GeometryPolygon` contains an `NMAGeoPolygon` and a secondary `NMAGeoPolygon` array.

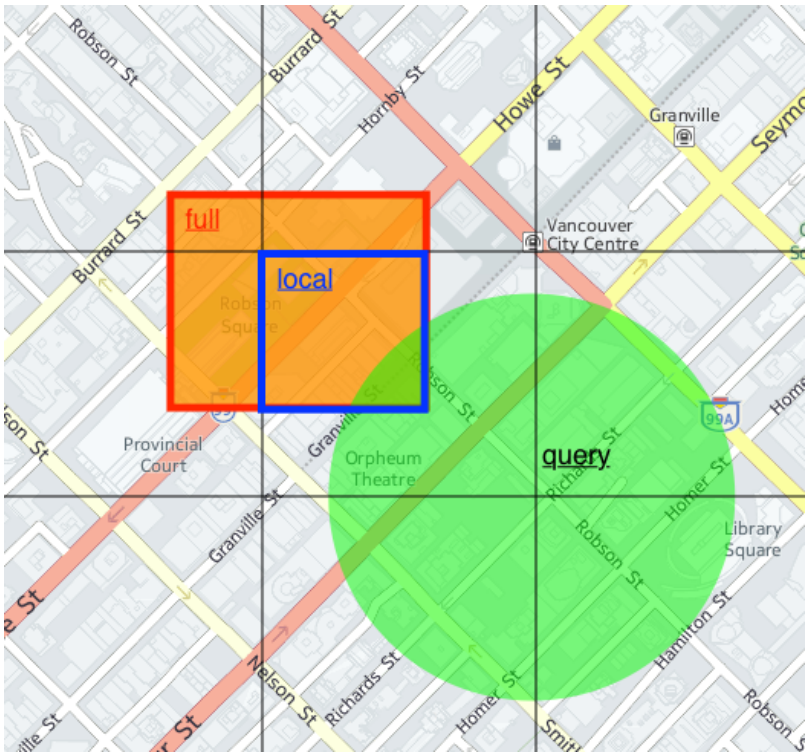
Understanding Local and Full Search Results

While processing user-uploaded data, CLE2 creates a look-up index where geometries are divided by an internal fixed grid. If a geometry spans across several grid tiles, then the search index may contain smaller slices of this uploaded geometry. This behavior allows for better search performance as well as optimized return values since it is possible to only return the relevant part of the originally submitted geometry and thus reduce the response size and processing time.

Before executing a search you can specify if you are only interested in part of the geometry that falls within the tiles around the search area (in other words, the tiled "local" geometry), or if you would like to receive the full geometry as originally uploaded.

Note: By default HERE SDK returns full geometries.

Figure 76: Local and Full Geometry



You can use the following three options to define whether you are requesting for full or local results. These options are available for all search types:

NMACLE2GeometryType	Meaning
`NMACLE2GeometryFull`	The result contains the original geometry (as uploaded). This is the default.
`NMACLE2GeometryLocal`	The result contains the processed geometry that falls within the search area for the tiles in reach.
`NMACLE2GeometryNone`	No geometry is returned at all, only properties/attributes of geometries that match the given search are returned.

You can set the geometry type by using `setGeometryType:` method:

```
NMACLE2ProximityRequest *proximityRequest =
 [[NMACLE2ProximityRequest alloc] initWithLayer:searchLayer center:coordinates radius:radius];
[proximityRequest setGeometryType:NMACLE2GeometryFull];
```

Proximity Search Request Example

To perform a custom location search, you need to create an `NMACLE2ProximityRequest` using `initWithLayer:center:radius` or `initWithLayers:center:radius` methods.

Proximity search returns a list of custom geometries that fall within a specified radius of an `NMAGeoCoordinates` location. For example, the following code shows how to perform search for all

locations in the previously mentioned stores layer that exists within an 8 kilometre radius of Frankfurt Central Station:

```
NMACLE2ProximityRequest * proximityRequest;
proximityRequest= [[NMACLE2ProximityRequest alloc] initWithLayer:@"HERE_SITES"
    center:[NMAGeoCoordinates geoCoordinatesWithLatitude:49.196261
     longitude:-123.004773]
    radius:8000]; // 8km

// Perform the request
[proximityRequest startWithBlock:^(NMACLE2Request *request, NMACLE2Result * result, NSError *error)
 {
  if(!error) {
   // use result.geometriesArray to retrieve list of found NMACLE2Geometry results
  }
 }];
```

The Layer ID parameter represents a set of custom uploaded geometries. For example, `"HERE_SITES"` layer ID represents a sample layer that contains locations of HERE offices in Germany. Since offices are represented by simple points, the returned geometries are of type `NMACLE2GeometryPoint`.

You can also perform proximity search on different layers at the same time:

```
NSArray *layers = @[@"LAYER_1", @"LAYER_2"];
NMACLE2ProximityRequest * proximityRequest;
proximityRequest= [[NMACLE2ProximityRequest alloc] initWithLayers:layers
    center:[NMAGeoCoordinates geoCoordinatesWithLatitude:50.113905
       longitude:8.677608]
    radius:500]; // 500 meters

// Perform the request
[proximityRequest startWithBlock:^(NMACLE2Request *request, NMACLE2Result * result, NSError *error)
 {
  if(!error) {
   // use result.geometriesArray to retrieve list of found NMACLE2Geometry results
  }
 }];
```

After creating a request object you can call `startWithBlock:` method to launch the search request and listen for search results.

You can also add a filter to the request. A filter is a JavaScript-like expression that is evaluated for each location-matching search query. When specified, only geometries for which the expression evaluates to true, e.g. when attributes are matching, are returned. For example, if you want to find geometries that have the custom location parameter of `rating` that is greater than 3 and the attribute "`NAME`" as `"MyPlace23"`, perform the following:

```
NMACLE2ProximityRequest * proximityRequest;
proximityRequest= [[NMACLE2ProximityRequest alloc] initWithLayer:@"HERE_SITES"
    center:[NMAGeoCoordinates geoCoordinatesWithLatitude:49.196261
       longitude:-123.004773]
    radius:8000]; // 8 km

[proximityRequest setQuery:@"CITY == 'Burnaby' && NAME1 != 'MyPlace'"];

// Perform the request
[proximityRequest startWithBlock:^(NMACLE2Request *request, NMACLE2Result * result, NSError *error)
 {
```

```
  if(!error) {
   // use result.geometriesArray to retrieve list of found NMACLE2Geometry results
  }
 }];
```

Iterating Through Results

The `NMACLE2Result` object contains an `NSArray` with all the `NMACLE2Geometry` objects found. Since different objects can be returned, it is recommended to test for type before using the returned geometry. For example, you can perform the following inside the request completion block:

```
for (NMACLE2Geometry *currentLocation in result.geometries) {
 if([currentLocation isKindOfClass:[NMACLE2GeometryMultiPoint class])
 {
  NMACLE2GeometryMultiPoint *multiPoint = (NMACLE2GeometryMultiPoint *)currentLocation;
  // multiPoint.coordinatesArray is an NSArray containing the found NMAGeoCoordinates
  NSLog(@"Found %lu coordinates", (unsigned long)[multiPoint.coordinatesArray count]);
  NSLog(@"Geometry attributes: %@", multiPoint.attributes);
 }
}
```

Each found `NMACLE2GeometryMultiPoint` contains two important properties: a coordinates `NSArray` and an attributes `NSDictionary` with flexible user-defined fields.

Using CLE2 Offline

You can perform search requests online to the server or offline to the local device. To enable offline search against local data, HERE SDK provides different ways for you to pre-fetch data.

Use offline mode as much as possible since it provides the following advantages:

- Resilience to network instability.
- More efficient use of network bandwidth. Instead of sending one request per object you can aggregate requests locally and transmit data in batches.
- Savings in network bandwidth. Your app can cache and update data only near the user's current location or pre-download a layer only when a Wi-Fi network becomes available.
- Potentially making the application more responsive and improving user experience and interface interactions since the data is already available locally on the device.
- Create or modify geometries with HERE SDK and then store them locally effectively using HERE SDK as a data source and a storage of information.

The offline CLE2 feature is designed to be simple to use. Since all database synchronization and geospatial-related complexities are handled by the SDK, you can focus on other parts of app development.

Querying the Local Storage

To search using locally stored data, set the request mode to `NMACLE2Offline` for any of your `NMACLE2Request` and perform the request:

```
// Create any request as normal; In this example we use the proximity request:

NMACLE2ProximityRequest * proximityRequest;
proximityRequest= [[NMACLE2ProximityRequest alloc] initWithLayer:@"HERE_SITES"
    center:[NMAGeoCoordinates geoCoordinatesWithLatitude:50.113905
```

```
     longitude:8.677608]
    radius:500]; // 500 meters

// Set to offline mode:
[proximityRequest setRequestMode: NMACLE2Offline];

// Now perform the request
[proximityRequest startWithBlock:^(NMACLE2Request *request, NMACLE2Result * result, NSError *error)
 {
 if(!error) {
   NSLog(@"Geometries returned in result.geometriesArray came from the device local storage.");
 }
  }];
```

You can configure the search request to hybrid or automatic mode indicating that if during an online request the connection drops or there is a network error, then the request automatically falls back to an offline operation. You can see whether the search was performed online or offline by checking the connectivity mode that was used to perform the search. This can be done by inspecting `requestMode` property on the `NMACLE2Result` object.

```
// Create any request as normal; In this example we use the proximity request:

NMACLE2ProximityRequest * proximityRequest;
proximityRequest= [[NMACLE2ProximityRequest alloc] initWithLayer:@"HERE_SITES"
       center:[NMAGeoCoordinates geoCoordinatesWithLatitude:50.113905
              longitude:8.677608]
       radius:500]; // 500 meters


// Set to automatic mode
[proximityRequest setRequestMode: NMACLE2Automatic];


// now calling a startWithBlock will try an online request, if it fails, the offline storage kicks
 in
[proximityRequest startWithBlock:^(NMACLE2Request *request, NMACLE2Result * result, NSError *error)
 {
    if(!error) {
      NSLog(@"Geometries returned in result.geometriesArray.");
      // to find out whether this response came from the local storage or was
      // obtained from the server (online request), check the requestMode property:
      NMACLE2RequestMode * requestMode = result.requestMode;
    }
  }];
```

Ways to Populate the Local Storage

By default offline features are disabled and the local storage contains no data. There are currently three ways to add geometries to make them available for offline search:

1. Enable caching when performing one or more requests (for example, using `NMACLE2ProximityRequest`).
2. Download one or more layers.
3. Direct insertion of data into the local database.

After populating the database, you can query for the data in offline mode as usual by switching connectivity mode of the respective request to `NMACLE2ConnectivityModeOffline`.

NMACLE2DataManager and NMACLE2Task

The `NMACLE2DataManager` object is the central interaction point with the local storage. With it, it is possible to:

- Download all geometries of a specific layer
- Check how many geometries are currently stored in total, or in a specific layer
- Delete geometries belonging to a specific layer
- Purge the local storage by deleting all items
- Create, update, or delete a local or remote geometry

All the operations relating to data management that `NMACLE2DataManager` exposes make use of an `NMACLE2Task` that represents a unit of work. Since all of the data management operations involve database access, network communication, or both, `NMACLE2Task` runs asynchronously. You can obtain an `NMACLE2Task` object from `NMACLE2DataManager`.

With `NMACLE2Task` you can:

- Pass it to other parts of your code. `NMACLE2Task` is a self-contained unit of work.
- Subscribe for results of the operation. Multiple subscribers are supported and they are called on the main thread.
- Start execution of the task. Tasks are reusable. You can run them repeatedly multiple times, which makes retrying a failed operation very easy.
- Cancel a running task.
- Check if the task is started.
- Check if the task has finished.
- Wait for the task to finish.
- Retrieve the status of a finished operation directly from the task (check for errors).
- Retrieve the result of a successfully finished operation directly from the task.

Storing Data by Caching Search Results

When caching is enabled in an `NMACLE2Request`, any returned geometries are automatically stored locally. To activate it, set `cacheEnabled` property to `YES` before performing the request:

```
// Create any request as normal; In this example we use the proximity request:

NMACLE2ProximityRequest * proximityRequest;
proximityRequest= [[NMACLE2ProximityRequest alloc] initWithLayer:@"HERE_SITES"
    center:[NMAGeoCoordinates geoCoordinatesWithLatitude:50.113905
     longitude:8.677608]
    radius:500]; // 500 meters

// activate caching:
[proximityRequest setCacheEnabled: YES];

// Now perform the request
[proximityRequest startWithBlock:^(NMACLE2Request *request, NMACLE2Result * result, NSError *error)
 {
 if(!error) {
   NSLog(@"Geometries returned in result.geometriesArray are now stored in the local database.");
 }
  }];

// now some geometries are in local storage. At a later point in time if we'd like to make an
 offline search,
```

```
// simply switch the requesMode to offline only in the request:
[proximityRequest setRequestMode: NMACLE2Offline];

// now calling a startWithBlock will operate completely offline:
[proximityRequest startWithBlock:^(NMACLE2Request *request, NMACLE2Result * result, NSError *error)
 {
 if(!error) {
   NSLog(@"Geometries returned in result.geometriesArray are now stored in the local database.");
 }
  }];
```

Storing Data by Downloading Layers

The second option is to use `NMACLE2DataManager` to insert data to the local storage using `downloadLayer` method.

The following is an example of how to use `downloadLayer` method:

```
// Usage examples of NMACLE2DataManager

NMACLE2DataManager * dataManager = [NMACLE2DataManager sharedInstance];

// 1 - Download a layer previously uploaded to the server
[storage downloadLayer:@"LOCAL_SHOPS" completionHandler:^(NSError * _Nullable error) {
   if(error)
   {
    NSLog(@"Unable to download layer. Error: %@", [error description]);
   }
   else
   {
    NSLog(@"Layers downloaded successfully.");
   }
  }];

// 2 - Print the total number of stored geometries (sum of all downloaded layers),
// plus any cached geometries (e.g., from a proximity request with cache enabled)

NSNumber * numberOfGeometries;
[storage numberOfStoredGeometries: &numberOfGeometries];
NSLog(@"Total count: %lli", [numberOfStoredGeometries longLongValue]);

// 3 - Delete all geometries from a specific layer
NSError * error = [[NMACLE2DataManager sharedManager] deleteLayer:@"POKEMONS"];
if(error)
{
  NSLog(@"Unable to delete layer. Error: %@", [error description]);
}
else
{
 NSLog(@"Layers deleted successfully.");
}

// 4 - Completely delete all stored data
[[NMACLE2DataManager sharedManager] deleteAll];
```

Storing Data by Inserting Geometries

You can generate location-based data and persist it locally, remotely, or both, by using the method `geometryTask:onLayer:withGeometries:inStorage:` from the `NMACLE2DataManager` class. This

factory method returns an `NMACLE2Task` object that can be used to start, cancel, or to fetch results of operations at any given time.

```
geometryTask:(NMACLE2Operation)operationType
       onLayer:(NSString*)layerId
withGeometries:(NSArray<NMACLE2Geometry *>*)geometriesArray
     inStorage:(NMACLE2StorageType)storage;
```

- The first parameter in this method describes the operation type which can be one of the following:
  - `NMACLE2OperationCreate`
  - `NMACLE2OperationUpdate`
  - `NMACLE2OperationDelete`

  Note that querying for geometries is accomplished through the respective `NMACLE2Request` specialized classes, so there is no "read" opreation here.
- The second parameter is the layer the operation should be applied to.
- The third parameter is a list with the geometries themselves.
- The last parameter defines whether to operate on local (`NMACLE2StorageTypeLocal`) or remote storage (`NMACLE2StorageTypeRemote`) using HERE CLE2 server.

Note: While this section covers usage of this method for the local option, all operations (create, update, delete) can also be used to change remote layers.

The following is an example on how to create a geometry and store it locally:

```
NSMutableArray <NMACLE2Geometry *>* geometriesToAdd = [NSMutableArray new];

for (int i = 0; i < 100; ++i) {
    NMACLE2GeometryPoint * newPoint = [[NMACLE2GeometryPoint alloc] init];
    NSString * key = [NSString stringWithFormat:@"key_%i", i];
    NSString * valueString = @"hello";
    newPoint.attributes = [@{ key : @"value",
                              @"HERE_SDK" : @"3.4",
                              @"Some_numbers" : valueString,
                              @"New_Feature" : @"Offline functionality \U0001F44C"} mutableCopy];
    newPoint.coordinates = [[NMAGeoCoordinates alloc] initWithLatitude: i % 180 longitude:i % 359];
    [geometriesToAdd addObject:newPoint];
}
NMACLE2Task<NMACLE2OperationResult*> * task;
task = [[NMACLE2DataManager sharedManager] geometryTask:NMACLE2OperationCreate
                                                onLayer:LOCAL_POINTS_TEST_LAYER
                                         withGeometries:geometriesToAdd
                                              inStorage:NMACLE2StorageTypeLocal];
[task subscribeWithBlock:^(NMACLE2OperationResult * _Nonnull result, NSError * _Nonnull error) {
            finished_ = YES;
            error_ = error;
            if(error)
            {
                NSLog(@"error: %@", [error description]);
            }
            else
            {
                NSLog(@"Finished for layer %@", LOCAL_POINTS_TEST_LAYER);
            }
}];

[task start];
```

Uploading a Local Layer

It is possible to upload a locally stored layer to the server. Since this requires two operations (fetch from local storage and upload), it's a good candidate to run individual tasks in synchronous manner to avoid callback hell creeping in. Of course, this needs to be done on its own thread, for example using AsyncTask.

```
// first fetch all geometries

NMACLE2Task<NSMutableArray<NMACLE2Geometry *> *> *fetchTask

fetchTask = [dataManager fetchLocalLayersTask:[NSArray arrayWithObjects:layerName, nil]];
[fetchTask startWithBlock:^(NSMutableArray < NMACLE2Geometry * > *_Nonnull result, NSError *_Nonnull
 error) {
            if (error) {
                NSLog(@"Error fetching local geometries: %@", error.description);
                return;
            }
            // Success, upload all geometries from the selected local layer
            NMACLE2Task<NMACLE2OperationResult *> *uploadtask;
            uploadtask = [dataManager uploadLayerTask:layerName withGeometries:result];
            // start the upload
            [uploadtask startWithBlock:^(NMACLE2OperationResult *_Nonnull result, NSError *_Nonnull
 error) {
                if (error) {
                    NSLog(@"Upload error: %@", error.description);
                    return;
                }
                NSLog(@"Layer Upload Complete. %@ geometries uploaded.", result.affectedItemCount);
            }];
        }];
```

Data Management Considerations

The following are a few tips to help with data management when using CLE2 in an offline context.

Local-only Geometries

All CLE2Geometry objects have the following properties:

1. Geometry ID, accessible with `geometry_id`
2. Locality flag, accessible with `is_local`

The geometry ID is unique to a layer. If a geometry object has just been created, its geometry ID is `null` and the locality flag is `false`.

The locality flag tells whether this geometry belongs to a local context only, meaning it was not retrieved or passed through the CLE2 server. A geometry with a `true` locality flag has a locally generated unique geometry ID. Otherwise, it contains a server-provided ID. This server-provided ID is not related to the locally generated IDs of geometries stored directly in the database created via `geometryTask:onLayer:withGeometries:inStorage:`.

Note: The functionality of locally storing geometries without passing through the server is provided so that you did not need to manage data persistence on these objects when a connection is not available.

For simplicity, when saving geometries directly to the local database, keep them using a separate layer name. If at a later desired point in time these geometries should be shared with the server, fetch all local geometries using `fetchLocalLayersTask:` method of

`NMACLE2DataManager` and then upload them either using `uploadLayerTask:withGeometries:` or `geometryTask:onLayer:withGeometries:inStorage:` with a create operation (`NMACLE2OperationCreate`). This avoids the requirement to check for `is_local` property.

By using these concepts, you can move geometries to different layers, contexts, and use these tools to organize data.

Data Consistency

Use of `uploadLayerTask:withGeometries:` should be primarily restricted to administrative users because this method deletes all existing geometries in the server and recreates the layer with the provided ones. If the user does not have the latest information for this layer, data loss may occur, as it can overwrite another user's upload.

Therefore, for a scenario with continuous or concurrent geometry upload, use `geometryTask:onLayer:withGeometries:inStorage:` method with `NMACLE2OperationCreate` or `NMACLE2OperationUpdate`. Operating in an "append only" manner or only updating the existing geometries prevents data loss even if users are uploading geometries concurrently to the server.

Current Limitations

Currently individualized user account management for the CLE2 server is not available. For security reasons care must be taken that your app credentials are kept well hidden. If your application requires a user account access feature, see *Service Support* on page 11 to contact us for more details .

Note: Since geospatial queries are the focus of CLE2, HERE SDK does not support attribute searches in offline mode. You can filter the data using one of the geospatial queries (such as proximity) to narrow down the results to a small enough number that most applications do not suffer performance impact by iterating the geometry attributes key-value dictionary to filter results further.

Toll Cost Extension

Toll Cost Extension provides you with the possibility to easily access Toll Cost Extension API from HERE SDK. HERE Toll Cost Extension (TCE) allows you to determine the toll costs for a specified route for a defined vehicle profile.

TCE Classes

Class	Description
NMATollCostOptions	All parameters for the toll cost calculation including route, vehicle profile, currency, and departure date. It implements `[NSObject description]` method.
NMATollCostRequest	Creates the TCE request for the toll cost data.
NMATollCostResult	After a TCE data request run the result is returned with this class.
NMATollCostVehicleProfile	All parameters of the vehicle to be used. It implements `[NSObject description]` method.

Requesting the Toll Cost Data

To use Toll Cost Extension, you need to first have a route calculated in online mode as the Toll Cost Extension requires permanent directed link IDs. After having a route from the core router, you can then retrieve the toll cost of the route.

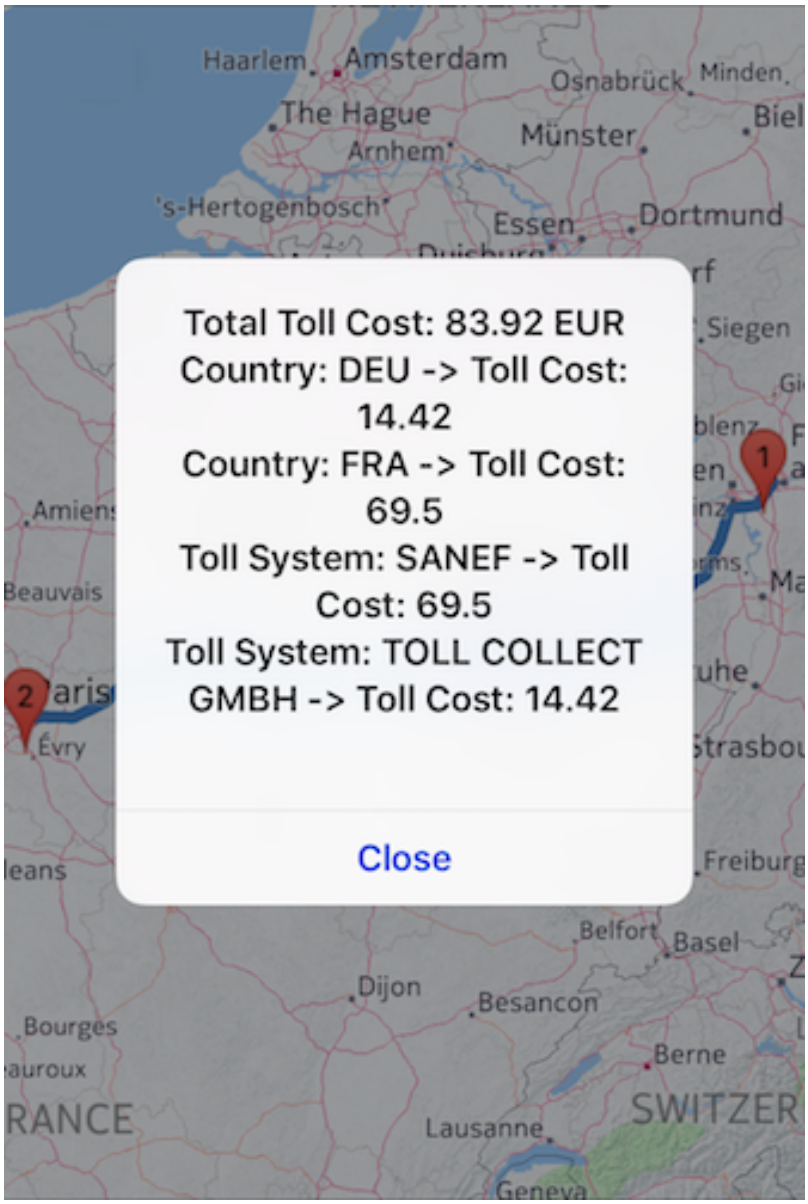
```
// Create the core router
NMACoreRouter *coreRouter = [[NMACoreRouter alloc] init];

// We need link IDs, NMARoute.permanentDirectedLinkIds, so we force online routing
coreRouter.connectivity = NMACoreRouterConnectivityOnline;
```

You can provide options for the toll cost request through an instance of `NMATollCostOptions`. If the vehicle needs non-default settings, such as if this toll request is for a specific type of vehicle, create an `NMATollCostVehicleProfile` object.

Note: It is your responsibility to provide a compatible route and toll cost options. When setting the toll cost options, make sure that they match the route options. For example, if the toll cost option vehicle type is set as truck but the route is created for a car, they are incompatible. In this case the toll cost result may not be valid.

Next, create an `NMATollCostRequest` request object. If the request object is valid and the route contains permanent directed linkids, then you can execute the toll cost request via a block or a listener. When the result is received, first it is checked for any errors. If there are no errors, its toll cost contents are retrieved.

Figure 77: A Toll Cost Example




```
// Step 1: Get the input

// Mandatory. Assume it is calculated via the above core router for a truck.
NMARoute route;
// Optional. Set it to the date and time for the trip
NSDate departureTime;
NMATollCostVehicleProfile *vehicleProfile = [[NMATollCostVehicleProfile alloc] init];

vehicleProfile.tollVehicleType = NMATollCostVehicleTypeTruck;
vehicleProfile.trailerType = NMATollCostTrailerTypeNone;
vehicleProfile.trailersCount = NMATollCostTrailersCountNone;
vehicleProfile.vehicleNumberAxles = 2;
vehicleProfile.emissionType = NMATollCostEmissionTypeEuroVI;
vehicleProfile.hybridType = NMATollCostHybridTypeNone;
vehicleProfile.height = 3.8f;
```

```
vehicleProfile.vehicleWeight = 11.0f;
vehicleProfile.limitedWeight = 11.0f;
vehicleProfile.passengersCount = 1;
vehicleProfile.tiresCount = 4;
vehicleProfile.commercial = true;
vehicleProfile.shippedHazardousGoods = NMATollCostShippedHazardousGoodsNone;
vehicleProfile.heightAbove1stAxle = 1.0f;

// Step 2: Wrap all the input
NMATollCostOptions *options = [NMATollCostOptions alloc] initWithVehicleProfile:route];
parameter.departure = departureTime; // Optional
parameter.currency = @"USD"; // Optional

// Step 3: Create the TCE request
NMATollCostRequest request = [[NMATollCostRequest alloc] initWithRoute:route andOptions:options];

// Step 4: Execute the TCE request with a block
// Is the request valid?
if (request) {
 [request startWithBlock:^(NMATollCostRequest *request, NMATollCostResult *result, NSError *error) {
  if (error) {
   // Something has gone wrong
   NSLog(@"Error occurred!\n"
      "Code: %ld\n"
      "Description: %@\n",
      (long)error.code,
      error.localizedDescription];
  } else {
   // Retrieved the result successfully

   // What is the total toll cost?
   NSString *tollCost = [NSString stringWithFormat:@"%.2f", result.tollCost.doubleValue];
   NSLog(@"Total Toll Cost: %@ %@\n", tollCost, options.currency];

   // What is the toll cost per country along the route?
   NSDictionary<NSString*, NSString*> *countryTollMap = result.tollCostByCountry;
   for (NSString *country in countryTollMap) {
    NSlog(@"Country: %@ -> Toll Cost: %@\n", country, countryTollMap[country]);
   }

   // What is the toll cost per toll system along the route?
   NSDictionary<NSString*, NSString*> *tollSystemMap = result.tollSystemMap;
   for (NSString *system in tollSystemMap) {
    NSlog(@"Toll System: %@ -> Toll Cost: %@\n", system, tollSystemMap[system]);
   }
  }
 }];
} else {
 NSLog(@"Invalid request!");
}
```

Turn-by-Turn Navigation for Walking and Driving

Note: [Important] Application developers using Turn-by-turn Guidance APIs are required to thoroughly test their applications in all expected usage scenarios to ensure safe and correct behavior. Application developers are responsible for warning app users of obligations including but not limited to:

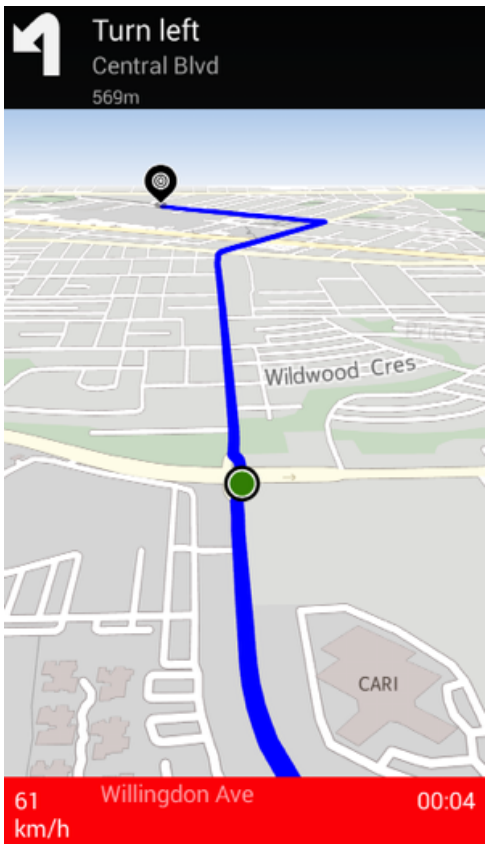
1. Do not follow instructions that may lead to an unsafe or illegal situation.
2. Obey all local laws.
3. Be aware that using a mobile phone or some of its features while driving may be prohibited.

4. Always keep hands free to operate the vehicle while driving.
5. The first priority while driving should be road safety.

HERE iOS SDK supports navigation on pedestrian, car, and truck routes. Using this feature, your app can check the current device position against a calculated route and provide just-in-time navigational instructions, both as visual and voice. Two navigation modes are supported for walking and driving: *Turn-by-Turn Navigation Mode*, which takes the calculated route and matches the current position against the route, and *Tracking Mode*, which only tracks the current position without using a route. In Tracking Mode no voice instructions are provided. For more information on voice instructions please see *Voice Instructions* on page 167.

Note: Your application should switch to the navigation-specific map schemes while performing car or pedestrian navigation. For more information on using these schemes see *Map Schemes* on page 39.

Figure 78: Turn-by-Turn Navigation with Speed Warning



Navigation Example on GitHub

You can find an example that demonstrates this feature at *https://github.com/heremaps/* (Obj-C) and *https://github.com/heremaps/* (Swift).

Navigation rerouting

If you are using Turn-by-Turn Navigation for driving, you have to implement rerouting delegate methods to display actual `NMAMapRoute` on `NMAMapView` object. The rerouting is triggered when navigation manager has determined deviation between the route and actual user position. Navigation manager triggers the following methods:

- `navigationManager:navigationManagerWillReroute:` - when rerouting is triggered.
- `navigationManager:didRerouteWithError:` - when attempt to reroute finished (it doesn't guarantee that new route was created).
- `navigationManager:didUpdateRouteWithResult:` - when a change is made to the route being navigated.

The `navigationManager:didUpdateRouteWithResult:` is the main place for `NMAMapRoute` redraw. This method might be called in different scenarios:

- User changed route programatically using

```
[[NMANavigationManager sharedNavigationManager] setRoute:newRoute];
```

- After successfully rerouting due to deviation between navigation route and actual user position.
- When navigation is stopped.

The following code is an example of an `navigationManager:didUpdateRouteWithResult:` implementation:

```
- (void)navigationManager:(nonnull NMANavigationManager *)navigationManager
 didUpdateRouteWithResult:(nonnull NMARouteResult *)routeResult
{
    if (routeResult && routeResult.routes.count > 0) {
        // Let's add the 1st result onto the map
        self.route = routeResult.routes[0];

        // remove previously created map route from map
        if (self.mapRoute) {
            [self.mapView removeMapObject:self.mapRoute];
        }
        // create new one based on provided route
        if (route) {
            self.mapRoute = [NMAMapRoute mapRouteWithRoute:self.route];
            self.mapRoute.traveledColor = [UIColor clearColor];
            [self.mapView addMapObject:self.mapRoute];
        }
    } else {
        // The routeResult doesn't contain route for redraw.
        // It might occur when navigation stop was called.
    }
}
```

Background Navigation

If you are using the Turn-by-Turn Navigation Mode for driving, you can also set HERE SDK to perform guidance (including voice instructions and event callbacks) while the app is in the background. However, there are a few limitations that you should be aware of:

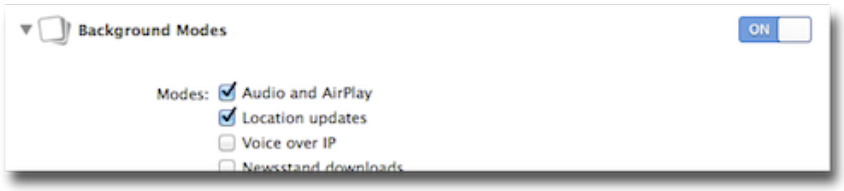
- Unlike the foreground navigation scenario, HERE SDK does not stream map data during background navigation. To support background navigation, HERE requires preloading map data (such as for the current city or state) using *NMAMapLoader* if your app may be used in the background.
- You cannot render a map (e.g. send it to an external device) when the app is in the background.
- Navigation cannot be started from the background unless this has been previously enabled while the app is in the foreground. Therefore, an app that launches directly to the background

cannot reliably launch a navigation session. For more information about this check `backgroundNavigationStartEnabled` property on the `NMANavigationManager` class.

To enable this feature, perform the following steps:

1. In Xcode **Capabilities** tab enable **Background Modes** and check the following entries:
   - Audio and AirPlay
   - Location updates

   Figure 79: Background Modes option in Xcode

2. Next, turn on background navigation in `NMANavigationManager`:

```
[NMANavigationManager sharedNavigationManager].backgroundNavigationEnabled = YES;
```

NMANavigationManager Class

`NMANavigationManager` class is responsible for providing voice and visual instructions to the user while driving or walking. The Navigation Manager is a singleton class, and the singleton instance can be accessed using `[NMANavigationManager sharedNavigationManager]`.

You can start navigation using the following methods:

- `startTurnByTurnNavigationWithRoute:` - Starts the Navigation Manager in Navigation Mode
- `startTrackingWithTransportMode:` - Starts the Navigation Manager in Tracking Mode

Note:

- Navigation operations require device positioning. When navigation is started in the Navigation Manager, the `NMAPositioningManager` is also started automatically. By using `NMAPositioningManager` with a position data log, HERE SDK can perform a simulated navigation session. For more details on using simulated position data consult *Basic Positioning* on page 86.
- HERE SDK may not start voice guidance immediately if the user is far from a road when Turn-by-Turn Navigation Mode begins. Your application should display a message such as "head to the nearest road" until voice guidance begins.

`NMANavigationManager` also provides relevant information that your application can display during a navigation session such as distances, upcoming maneuvers, and average travel speed. Upcoming maneuvers are represented by `currentManeuver` and `nextManeuver` properties. `currentManeuver` represents the most immediate upcoming maneuver while `nextManeuver` represents the next most immediate upcoming maneuver. Maneuvers contain road-related information such as road name, the turn to be taken, and the maneuver orientation.

`currentManeuver` and `nextManeuver` may not be always readily available. You can use `navigationManager:hasCurrentManeuver:nextManeuver:` method in the `NMANavigationManagerDelegate` protocol to receive notifications when maneuvers have been updated. For more information see the section about `NMANavigationManagerDelegate`.

You can take advantage of `lowSpeedOffset`, `highSpeedOffset`, `speedBoundary`, and `speedWarningEnabled` properties to set up the speed warning feature. When speed warning is enabled, `navigationManager:didUpdateSpeedingStatus:forCurrentSpeed:speedLimit` notification is sent to the navigation manager delegate when the boundary plus an offset has been violated. The speed warning feature is enabled by default.

Note: The `NMANavigationManager` only returns car speed warnings. Truck speed warnings are not currently supported.

Map Tracking

`NMANavigationManager` also provides properties for customizing map tracking. Map tracking refers to the ability for the map location to be automatically updated during navigation. By default, the map tracking feature is enabled but it can be toggled through `mapTrackingEnabled` property. When tracking is enabled, you can use `mapTrackingOrientation` property to set whether the map is oriented dynamically or always pointed North. You can also set whether the map automatically adjusts the zoom level, based on the current device speed, by using `mapTrackingAutoZoomEnabled` property. By default dynamic orientation and auto zoom are enabled.

After starting `NMANavigationManager` in Navigation or Tracking mode, you can move the `positionIndicator` closer to the bottom of the screen by changing `transformCenter` property in `NMAMapView`. Having the `transformCenter` at the bottom of the screen ensures that the route and turning animations are more visible to the user.

Natural Guidance

`NMANavigationManager setNaturalGuidanceMode:` method can be used to enable natural guidance. Natural guidance refers to a type of dynamic information available during navigation where route guidance instructions contain contextual elements around a decision point. These contextual elements may include services, cartographic features, traffic signals. Some examples of natural guidance instructions are:

- "Go past the park on your right, then turn left at Anderson school on Bayview street"
- "Go through the traffic light and turn right before the petrol station"
- "Continue on your route passing the dome building on your right"

The available types of natural guidance information are defined by `NMANaturalGuidanceOption` enum. These options may be used individually or in combination, although typically only a single type of guidance instruction would be given for a particular maneuver. To disable natural guidance, pass `NMANaturalGuidanceNone` to `setNaturalGuidanceMode:` method.

Note: While using `NMANaturalGuidanceLandmark` option, the device locale and navigation voice package language should be set to match the user's physical location. Otherwise, landmark information may not be spoken. For example, while navigating in a country that uses French as its spoken language, the voice package and the device locale should be set to French.

NMANavigationManagerDelegate protocol

`NMANavigationManager` notifies the client of navigation events through its delegates that implement the `NMANavigationManagerDelegate` protocol. The delegate protocol includes the following methods:

- `navigationManagerDidReachDestination:` - Signifies that the destination of navigation was reached

- `navigationManager:hasCurrentManeuver:nextManeuver:` - Signifies that there is new instruction information available that can be fetched by accessing `nextManeuver` property in `NMANavigationManager`
- `navigationManager:didUpdateRoute:` – Provides the newly calculated `NMARoute` object which can be used to render a new route on the map
- `didUpdateSpeedingStatus:forCurrentSpeed:speedLimit:` - Signifies that the user has exceeded the speed limit with the road name and speed limit provided
- `navigationManagerDidLosePosition:` - Signifies that the system has lost its GPS signal
- `navigationManagerDidFindPosition:` - Signifies that the system has acquired a GPS signal
- `navigationManagerWillReroute:` – Signifies that a route recalculation has begun. This is a result of the current position deviating from the original route
- `navigationManagerDidReroute:` - Signifies that a route recalculation has finished
- `navigationManager:didFindAlternateRouteWithResult:` - Signifies that the navigation manager has found an improved route
- `navigationManager:shouldPlayVoiceFeedbackWithText:` - Signifies that a voice feedback is ready to be played. Use this callback return values to retrieve the text to be spoken (if applicable) and to control whether the current voice feedback should be played
- `navigationManager:willPlayVoiceFeedbackWithText:` - Signifies that a voice feedback event will occur (for example, when a maneuver is being announced)
- `navigationManager:didPlayVoiceFeedbackWithText:` - Signifies that a voice feedback event has finished
- `navigationManager:didFindAlternateRoutes:` - Signifies that alternative routes have been found during navigation with traffic avoidance mode enabled

Lane Information

The `NMANavigationManagerDelegate` also provides `navigationManager:didUpdateLaneInformation:roadElement:` method. This callback occurs when the user has arrived at a point in the route where lane information should be presented, such as before a highway exit. The `NMALaneInformation` class represents a lane turn direction and whether this lane is on the current route. For example, an application may receive `navigationManager:didUpdateLaneInformation:roadElement:` callback as the user navigates to an intersection. If the route requires a left turn and the current road has three lanes — a left-turn lane and two straight lanes — then the callback contains three `NMALaneInformation` objects. Since `NMALaneInformation` objects are always returned in the callback method in a left-to-right order, the first `NMALaneInformation` has a `directions` property as "left" and `recommendationState` property, which indicates whether the lane is on the current route. If there isn't enough data to determine whether the lane is on or off-route, `recommendationState` property returns `NMALaneInformationRecommendationStateNotAvailable`.

Note: Information about lanes requires extra map data for a route. The application must first download the required map data using `NMAMapDataPrefetcher` or `NMAMapLoader`. Otherwise, lane information will not be returned properly.

School zone

`NMASchoolZoneWarnerDelegate` provides callback methods to check whether there is school zone ahead, whether user enters or leaves school zone. Callback `didDetectSchoolZone` occurs when there is school zone ahead if user follows current route direction, it will be triggered in about 100 meters before each `NMARoadElement` with school zone. Callback `didUpdateSchoolZone` occurs either when user entered school zone or left. For example:

```
- (void)schoolZoneWarner:(nonnull NMASchoolZoneWarner *)schoolZoneWarner
     didDetectSchoolZone:(nonnull NMASchoolZoneNotification *)notification
{
    // distance to the road element with school zone
    NSUInteger distance = notification.distance;
    // road element associated with this school zone
    NMARoadElement* roadElement = notification.roadElement;
    // begin time, e.g. Monday 09:00
    NSDate* timeBegin = notification.schoolZoneInfo.timeBegin;
    // end time, e.g. Monday 17:00
    NSDate* timeEnd = notification.schoolZoneInfo.timeEnd;
    float speedLimit = notification.schoolZoneInfo.speedLimit;
}

- (void)schoolZoneWarner:(nonnull NMASchoolZoneWarner *)schoolZoneWarner
     didUpdateSchoolZone:(nonnull NMASchoolZoneInfo *)zoneInfo
{
    if (zoneInfo) {
        // user enters school zone
        float speedLimit = zoneInfo.speedLimit;
    } else {
        // user left school zone
    }
}
```

Developer can also use utility methods from `NMASchoolZoneWarnerDelegate` to check whether specific road element has school zone. For example:

```
 // Get road element from geo coordinate
 // Use instance of NMAMapView to get roadElementAtCoordinates
NMARoadElement* roadElement = [NMAMapView roadElementAtCoordinates:coordinate];
 if (roadElement) {
     // Check whether road element has active school zone
     NMASchoolZoneInfo* info = [NMASchoolZoneWarner schoolZoneInfoForRoadElement:roadElement];
     if (info) {
         // provided road element has school zone
     }

     // Check whether school zone is active at specific time
     // GMT: Sunday, 8 March 2020, 00:00:00
     NSDate *sunday = [NSDate dateWithTimeIntervalSince1970:1583625600];
     NMASchoolZoneInfo* infoAtTime = [NMASchoolZoneWarner schoolZoneInfoForRoadElement:roadElement
                                                                                  time:sunday];
     if (infoAtTime) {
         // school zone is active at provided time
     }
 } else {
     // NMARoadElement is nil either because there is no road at the provided location or map
     // data is not loaded. Use NMAMapLoader API to download map area, or use road element from
     // another provider, e.g. from the calculated route.
 }
```

Realistic View: 2D Signposts and Junction View

In addition to the data offered through the Lane Info feature, your application can also use HERE iOS SDK to offer image previews of signposts and junctions on certain highways. For example, as a user approaches a fork on a highway, your application can show a preview of two instruction signs above the highway and the lanes near the junction with indicator arrows highlighting the correct lanes. These image previews are known as 2D Signposts and Junction View, and together they are known as Realistic View.

Figure 80: An Example of a 2D Signpost

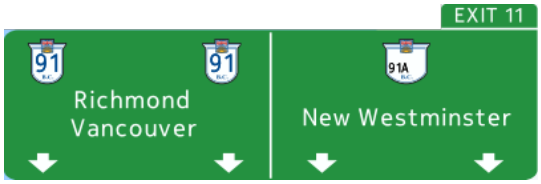


Figure 81: An Example of a Junction View

The Realistic View feature is disabled by default. To enable it, use `NMANavigationManager.realisticViewMode` property and set the view mode to `NMARealisticViewDay` or `NMARealisticViewNight`. Next, register the desired image aspect ratios by using `NMANavigationManager.realisticViewAspectRatios` property.

After adding your delegate implementation, your application begins receiving the following method callbacks as the user approaches a highway section that supports Realistic View. The *next maneuver* callback occurs when a realistic view is available in the second upcoming maneuver (for example, if the next maneuver is to enter a junction), and the *current maneuver* callback occurs as the most immediate upcoming maneuver is entered (for example, if the user is entering the junction).

- `navigationManager:didUpdateRealisticViewsForCurrentManeuver:`
- `navigationManager:didUpdateRealisticViewsForNextManeuver:`

The first two method callbacks mentioned above return an `NSDictionary` object named `realisticViews`, which holds one or more `NSDictionary` of images. To retrieve the `NSDictionary` of images from `realisticViews`, use one of the following aspect ratio strings as the key:

- `NMARealisticView16x9Key`
- `NMARealisticView3x5Key`
- `NMARealisticView4x3Key`
- `NMARealisticView5x3Key`

After performing this step, you can extract the signpost and junction view images from the `NSDictionary` through `NMANavigationManagerSignpostKey` and `NMANavigationManagerJunctionViewKey` keys.

Note: It is possible to add multiple aspect ratios and receive multiple images of the same signpost or junction view. However, please be aware that this may cause some performance impact.

Note: To dismiss the Realistic View images, your navigation manager delegate should also implement and listen for `navigationManagerDidInvalidateRealisticViews:` method. This method is called when realistic view images should be dismissed (for example, after the user has entered the junction).

The following is an example of a `navigationManager:didUpdateRealisticViewsForCurrentManeuver:` implementation:

```
- (void)navigationManager:(NMANavigationManager *)navigationManager
  didUpdateRealisticViewsForCurrentManeuver:(NSDictionary *)realisticViews
{
 NSDictionary *realisticView = [realisticViews objectForKey:NMARealisticView3x5Key];
 NMAImage *junction = [realisticView objectForKey:NMANavigationManagerJunctionViewKey];
 if (junction) {
  //** display the junction image
 }
 NMAImage *signpost = [realisticView objectForKey:NMANavigationManagerSignpostKey];
 if (signpost) {
  //** display the signpost image
 }
}
```

Voice Instructions

Voice instructions are available in HERE SDK as voice packages. Voice packages are available in two forms: pre-packaged or downloadable through the voice catalog. You can set a voice package to be used for navigational instructions. However, if a package is not set, HERE SDK sets the navigation voice language to US English, which is pre-packaged with HERE SDK.

Note: Voice instructions are only available in Navigation Mode for driving. Users of the pedestrian Navigation Mode receive audio beeps and vibration alerts at the change of each maneuver. Please note that HERE SDK temporarily lowers the volume of other audio when a beep or voice instruction occurs.

NMAVoicePackage class

`NMAVoicePackage` class encapsulates spoken voices that can be used for navigation guidance. It contains information such as name, gender, and the spoken language. This information can be accessed through its class properties.

A list of loaded `NMAVoicePackage` instances can be accessed by using `NMAVoiceCatalog` singleton instance. Multiple voice package can be loaded to the device but only one can be selected for navigation voice playback. An `NMAVoicePackage` can be added to `NMANavigationManager` through `voicePackage` property. Each `NMAVoicePackage` is represented by a unique ID in `packageId` property.

HERE SDK supports two types of voice packages: text-to-speech and pre-recorded. Pre-recorded voice skins provide basic maneuver instructions, such as "turn right in 300 metres", while text-to-speech voices also support spoken street names, such as "turn right in 300 metres onto Granville Street".

NMAVoiceCatalog class and NMAVoiceCatalogDelegate protocol

`NMAVoiceCatalog` class is used to access voice package files from the local device. An `NMAVoiceCatalog` object instance can be retrieved by calling:

```
NMAVoiceCatalog* voiceCatalog = [NMAVoiceCatalog sharedVoiceCatalog];
```

Then using `installedVoicePackages` property, you can fetch a list of `NMAVoicePackage` that are stored on the device. By default HERE SDK includes a number of pre-installed voice packages. You can also use `voicePackages` property to obtain a list of voice packages that are available for download from the voice packages server. To retrieve the latest list of packages from the server, call `updateVoicePackages`.

`NMAVoiceCatalogDelegate` is used for receiving callbacks that are related to certain asynchronous `NMAVoiceCatalog` operations. Examples of these operations include updating the catalog and installing a voice package.

Selecting a Voice Package and Starting Navigation

The following is an example of how to use an `NMAVoicePackage` with the navigation manager:

1. Get `NMAVoiceCatalog` by retrieving the shared object instance.

   ```
   NMAVoiceCatalog* voiceCatalog = [NMAVoiceCatalog sharedVoiceCatalog];
   ```

2. Implement and add an `NMAVoiceCatalogDelegate` to `NMAVoiceCatalog` using `delegate` property.
3. Next, get the catalog of voice packages by calling `updateVoiceCatalog`. When this operation is complete, a callback to `voiceCatalog:didUpdateWithError:` delegate method occurs.

   ```
   [voiceCatalog updateVoiceCatalog];
   ```

4. Using the list of `voicePackages`, find the desired voice package. If the voice package is not already in the list of installed voice packages (`installedVoicePackages`), install it. When the installation operation is complete, a callback to `voiceCatalog:didInstallPackage:withError` delegate method occurs.

   ```
   [voiceCatalog installVoicePackage:selectedPackage]
   ```

5. Once the package is installed, set it to `NMANavigationManager`.

   ```
   [NMANavigationManager sharedNavigationManager].voicePackage = selectedPackage;
   ```

6. Get a calculated `NMARoute` from `NMACoreRouter`. For more details on how to do this, please refer to the code samples in *Routing* section.
7. Start navigation according to user selection.

   ```
   NSError* error = [[NMANavigationManager sharedNavigationManager]
    startTurnByTurnNavigationWithRoute:calculatedRoute];
   ```

Controlling Audio Playback

Your application may require control over the timing of when HERE SDK navigation audio, such as a voice instruction, is played. For example, you may need to interrupt audio feedback to play another audio clip. To do this, use `cancelVoiceFeedback` method in `NMANavigationManager`. When this method is called, the current voice feedback stops and any queued feedback is also cleared.

In other scenarios you may need to suppress navigation audio for a period of time. To do this, implement `navigationManager:shouldPlayVoiceFeedbackWithText:` callback in `NMANavigationManagerDelegate` and return `NO`. This event is triggered whenever audio feedback is ready to be played, and returning `NO` skips this feedback from being played.

Traffic-Aware Navigation

With HERE SDK developers can enable turn-by-turn route navigation that takes live traffic information into account. `setTrafficAvoidanceMode:` method in `NMANavigationManager` can be used to set the way in which traffic should be handled during navigation.

Three modes are available for traffic avoidance, and they are defined by the following `NMATrafficAvoidanceMode` enumerations. The default mode is `NMATrafficAvoidanceDisabled`.

- `NMATrafficAvoidanceDisabled` - Disables use of traffic for rerouting purpose.

  In this mode the giudace engine disableas use of traffic for rerouting purpose, which means that online traffic information is not taken into account while rerouting. The live traffic data is not considered by NavigationManager unless setTrafficAvoidanceMode is enabled.
- `NMATrafficAvoidanceDynamic` - Performs traffic-aware rerouting without user input.

  In this mode the guidance engine performs periodic route calculations while the device is online. A route calculation is a server request where the server finds the most optimal route by avoiding traffic congestions and calculating speed limits. If the calculated route is different from the current route, the navigation manager automatically switches to the new route. It also triggers `navigationManager:didUpdateRoute:` delegate method.

  Note: You can set the frequency of the route request by using `setRouteRequestInterval:`.
- `NMATrafficAvoidanceManual` - Provides traffic-aware rerouting callback `navigationManager:didUpdateRoute:`.

  In this mode no rerouting occurs unless the `NMANavigationManager` is explicitly set with the new `NMARoute`. For more information see the next section.

Manual Traffic-Based Rerouting

If the device is online and `NMATrafficAvoidanceManual` mode is selected, the guidance engine periodically performs route recalculations while the device is online. You can listen to this recalculation event by implementing `navigationManger:didChangeRoutingState:` method in `NMANavigationManagerDelegate` and checking for the state change from `NMATrafficEnabledRoutingStateOn` or `NMATrafficEnabledRoutingStateNotAvilable` to `NMATrafficEnabledRoutingStateOngoingRequest`.

Note: `navigationManger:didChangeRoutingState:` callback is also called in automatic avoidance mode.

Route recalculation is a server request. The calculation finds the most optimal route by avoiding congestions and calculating speed limits. If the calculated route is different from the current route, the new route is returned through `navigationManager:didFindAlternateRoute:` delegate callback. You can then set the new `NMARoute` to the `NMANavigationManager` manually by calling `setRoute:`.

NMATrafficWarner Class

`NMATrafficWarner` class is responsible for enabling and handling traffic notifications. Traffic notifications occur if there is a traffic event on the current route and the user's current position is near the event.

To retrieve an instance of `NMATrafficWarner` object, use `trafficWarner` property from `NMANavigationManager`. You can then call `start` method on the `NMATrafficWarner` to initialize it.

To listen for traffic notifications, implement `NMATrafficWarnerDelegate` and `trafficWarner:didDetectTraffic:` callback method.

One or more of the following methods can be used to operate traffic warner or to retrieve a notification of a route:

- `isTrafficNotificationAhead:` - determines whether or not a traffic notification is ahead of the last callback position
- `isTrafficNotification:onRoute:` - determines if a traffic notification is on a given route
- `trafficNotificationOnCurrentRoute` - retrieves the traffic notification for the route that is in use by the navigation manager
- `trafficNotificationOnRoute:` - retrieves the traffic notification for the specified route
- `stop` - stops the traffic warner

NMATrafficNotification and NMATrafficNotificationInfo Classes

`NMATrafficWarner.Listener` provides a callback that returns an `NMATrafficNotification` object that is relevant to the current navigation session. This `NMATrafficNotification` contains a list of `NMATrafficNotificationInfo` instances associated with the traffic notification retrievable through `trafficNotificationInfo` property.

`NMATrafficNotificationInfo` class encapsulates the details of a traffic notification. `NMATrafficNotificationType` defines the type of traffic notification with regards to the current route.

The following properties and methods can be used to retrieve details about an `NMATrafficNotificationInfo` instance:

- `type` - the type of traffic notification info
- `severity` - severity of the current traffic notification event
- `affectedLength` - length, in metres, of the traffic notification event
- `distance` - gets the distance from the current `NMAPositioningManager` position to the traffic notification

For more information please consult the API Reference.

Finding Alternative Routes with Traffic Avoidance Mode Enabled

It is called when the navigation manager has found alternative routes for current one during navigation.

Note: Please do not confuse `navigationManager:didFindAlternateRoutes:` and `navigationManager:didFindAlternateRouteWithResult:`.
`navigationManager:didFindAlternateRoutes:` - optional routes.
`navigationManager:didFindAlternateRouteWithResult:` - an improved route.

The following is an example of an `navigationManager:didFindAlternateRoutes:` implementation:

```
- (void) navigationManager:(nonnull NMANavigationManager*)navigationManager
```

```
    didFindAlternateRoutes:(nonnull NMARouteResult *)routeResult
{
    //** remove old alternative routes from current map if any
    [self.activeMapView removeMapObjects: self.alternativeRoutes];
    self.alternativeRoutes = [[NSMutableArray alloc] init];

    //** enumerate all available alternative routes
    for (NMARoute *route in routeResult.routes) {
        //** Instantiate new map route object
        NMAMapRoute *mapRoute = [[NMAMapRoute alloc] initWithRoute:route];

        //** Install random color for the one route
        mapRoute.color =
        [UIColor colorWithHue:drand48() saturation:1.0 brightness:1.0 alpha:1.0];

        //** Update internal storage
        [self.alternativeRoutes addObject:mapRoute];
        //** Show alternative route simultaneously with current guidance route
        [self.activeMapView addMapObject:mapRoute];
    }
}
```

Audio Management

This section outlines how you can control and manage audio from HERE SDK. With the information from this section you can

- Control when audio clips are played
- Pass your application audio to `NMAAudioManager` by using its audio queue
- Disable automatic `NMAAudioManager` playback
- Receive audio playback events

The section also contains information about audio routing, e.g. how to use HERE SDK audio with a Bluetooth device.

NMAAudioManager and the Audio Queue

`NMAAudioManager` is the central class that is used by HERE iOS SDK to modify the application *AVAudioSession* and play audio. It is the interface that the `NMANavigationManager` uses to play audio feedback such as voice instructions. You can also use `NMAAudioManager` to change whether hardware keys directly control HERE SDK volume, and also use it to set volume as a factor relative to the user's device volume.

The `NMAAudioManager` contains a queue of audio output objects. You can add to this queue by calling `playOutput:` with `NMAAudioFileOutput`, `NMATTSAudioOutput`, or your own `NMAAudioOutput` implementation. You can also use `NMAAudioManager` methods such as `clearQueue`, `skipCurrentOutput`, and `stopOutputAndClearQueue` to manage audio output in this queue.

Audio Output Types

By default HERE SDK provides two `NMAAudioOutput` subtypes: `NMAAudioFileOutput` and `NMATTSAudioOutput`. You can play or enqueue these (or your own custom `NMAAudioOutput` subtype) by calling `playOutput:` method with `NMAAudioManager`.

`NMAAudioFileOutput` represents a collection of audio files that are compatible with the iOS *AVAudioPlayer*. You can get an instance by calling `NMAAudioFileOutput audioOutputWithFiles:` with an array of compatible files. All files in an `NMAAudioFileOutput` instance are together considered as a single output segment when it is played by the `NMAAudioManager`.

`NMATTSAudioOutput` represents a text-to-speech audio output segment. It contains a text string to be spoken and an *AVSpeechSynthesisVoice*. You can get an instance of this class by calling `NMATTSAudioOutput audioOutputWithText:` with the text to be converted to a speech sample. By default the text-to-speech (speech synthesis) engine selects the voice dialect based on the current device locale.

Delaying Audio Playback

You can use `audioRouteLatencyCompensation` property in `NMAAudioManager` to introduce some silence before audio playback. This is useful if you need to compensate for Bluetooth connection or audio ducking, and to prevent clipping guidance audio.

Disabling Automatic Playback

In case you would like more control over the application `AVAudioSession`, you can use `managesAudioSession` property to manage automatic playback.

When `managesAudioSession` property is `YES`, the `NMAAudioManager` automatically configures, activates, and deactivates the `AVAudioSession` as needed to play HERE SDK or application audio. If `managesAudioSession` is `NO`, the application `AVAudioSession` is not touched, and it is up to the application to configure the audio session for both its own and HERE SDK audio.

When `managesAudioSession` property is set to `NO`, `NMAAudioManager` continues to attempt to play audio output. However, the `AVAudioSession` is not modified. Instead, you should configure `AVAudioSession` to meet your own audio playback requirements. If you want to reconfigure `AVAudioSession` for `NMAAudioManager` events, implement `NMAAudioManagerDelegate` protocol methods.

Overriding Default Audio Playback Using NMAAudioManagerDelegate

Whether or not the `NMAAudioManager` is set to automatically manage your application HERE SDK audio session, you can choose to implement `NMAAudioManagerDelegate` to listen for relevant audio output events and perform custom logic before using the audio output for playback.

`NMAAudioManagerDelegate` contains the following methods:

- `audioManager:shouldPlayOutput:` - This method is called when the audio manager has output to play. If you implement this method, you must return `NO` to prevent audio playback, or return `YES` and then call `playOutput:` method. You can use this to customize audio playback to some degree. For example, you can choose to configure `AVAudioSession` before playback, if you opted to manage the `AVAudioSession` yourself.

- `audioManagerWillPlayOutput:` - Callback that occurs just before the output is played by `NMAAudioManager`.
- `audioManagerDidPlayOutput:` - Callback that occurs after the output is played by `NMAAudioManager`.

Audio Routing

With iOS 8 or above devices you can route `NMAAudioManager` audio playback to the device speaker or Bluetooth peripherals, such as a headset or a car stereo, by using `setAudioRoute:` method before `playOutput:` is called.

`setAudioRoute:` accepts the following `NMAAudioRoute` values. The default audio route is `NMAAudioRouteDefault`.

- `NMAAudioRouteDefault` - Uses the default audio route as determined by the operating system. This audio route may represent connected wired headphones, connected A2DP Bluetooth devices, or the device speaker.
- `NMAAudioRouteDeviceSpeaker` - Plays audio over the device speaker even if headphones or other peripherals are connected.
- `NMAAudioRouteBluetoothHFP` - Plays audio using Bluetooth the Hands-Free Profile (HFP).

If you set the audio route to `NMAAudioRouteBluetoothHFP`, HERE SDK attempts to play audio over the connected Bluetooth HFP device. If no device is connected, `NMAAudioManager` automatically sets the audio route back to `NMAAudioRouteDefault`. You can receive a notification for this fallback event by observing `NMAAudioRouteDidChangeNotification` using `NSNotificationCenter`.

You can retrieve the current audio route from `NMAAudioManager` by using `audioRoute` property.

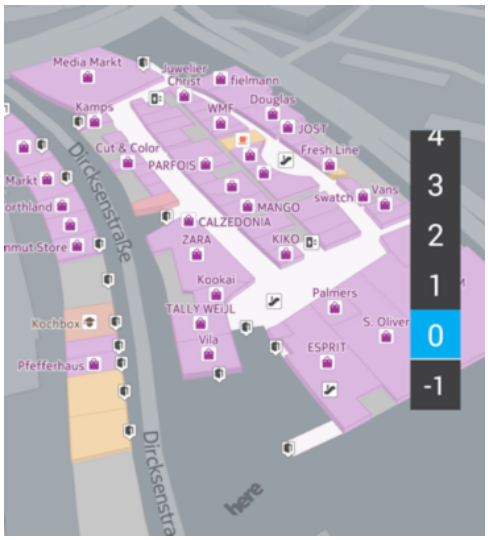
3D Venues

This section gives an overview of the classes and interfaces associated with the 3D Venues feature. Examples of available 3D venues include shopping malls and airports. Three feature use cases are explored: searching for a venue, opening a venue, and getting a notification when a venue is visible in the viewport.

Note: Public Venues Deprecation: Venues of publicly accessible locations such as airports or shopping malls are no longer being maintained by HERE. If you are a venue owner and want HERE to continue to maintain and surface your venue in our offerings, or those of our partners, contact us at *venues.support@here.com*.

The classes covered in this section include `NMAVenue3dMapLayer`, `NMAVenue3dService`, `NMAVenue3dMapLayerDelegate`, `NMAVenue3dController`, `NMAVenue3dServiceListener`.

Figure 82: 3D venue map of a Berlin shopping center



The 3D Venues feature can be used with or without a map. To use it with a map, use `NMAVenue3dMapLayer` class, which is retrieved by using `venue3dMapLayer` property in `NMAMapView`. To use 3D Venues without a map, use `NMAVenue3dService` class.

Initialization

`NMAVenue3dMapLayer` provides developers with access to all 3D venue-related features on a map. To begin using the layer, `NMAVenue3dMapLayer` needs to be retrieved from `NMAMapView`, for example:

```
_venueMapLayer = self.mapView.venue3dMapLayer;
```

Note: `NMAVenue3dMapLayer` must be retrieved from `NMAMapView`. You cannot directly instantiate it.

After retrieving the `NMAVenue3dMapLayer`, call `start:` method before beginning to use it. To ensure that the `NMAVenue3dMapLayer` has started and to receive venue-related events, implement the `NMAVenue3dMapLayerDelegate`.

```
- (void)venueMapLayerDidStart:(NMAVenue3dMapLayer *)venueMapLayer {
 // venue map layer is ready to be used
}
```

You can also retrieve the service status using `initializationStatus` property in `NMAVenue3dService`.

Working with 3D Venue Models

Once an `NMAVenue3dMapLayer` is correctly initialized, 3D-enabled venues are visible on the map. These venues can be distinguished by their colors and icons, as in the screenshot below.

Figure 83: A 3D venue on the Map

`NMAVenue3dMapLayer` offers two ways to select a venue and open the indoor map view. When a user taps the venue, `venueMapLayer:didTapVenue:atPoint:` method in `NMAVenue3dMapLayerDelegate` is called with an `NMAVenue3dVenue` object as a parameter. The venue can then be opened by giving the `NMAVenue3dVenue` object to `selectVenue:`. For example:

```
- (void)venueMapLayer:(NMAVenue3dMapLayer *)venueMapLayer didTapVenue:(NMAVenue3dVenue *)venue
 atPoint:(CGPoint)point
{
 _venue = venue;
 _tappedPosition = [self.mapView geoCoordinatesFromPoint:point];
 _tappedVenueId = [NSString stringWithString:_venue.uniqueId];
 [_venueMapLayer selectVenue:_venue];
}
```

When the venue is selected, `venueMapLayer:didSelectVenue:` callback of the `NMAVenue3dMapLayerDelegate` interface is called.

A venue can also be selected and opened by giving its identifier to `selectVenueWithVenueId:` method in `NMAVenue3dMapLayer`. For example, a typical scenario is a venue search with successful search results in an opened venue. You can also open a venue with a specific space selected via `selectVenueWithVenueId:spaceId:` method. This is suitable for searching specific space inside the venue.

`selectVenue:` method opens the venue right away in a synchronous manner by taking a downloaded `NMAVenue3dVenue` object as a parameter. However, `selectVenueWithVenueId:` and `selectVenueWithVenueId:spaceId:` methods may involve downloading the venue from the backend, asynchronously, while the venue is selected and opened. You can get a notification when asynchronous loading is complete by listening for `venueService:didGetVenue:` callback in `NMAVenue3dServiceListener`.

It is also possible to receive a notification when there is a venue in the viewport. You can use this callback to implement a feature such as drawing user attention to the venue when it is visible.

The triggering area for this is a rectangle at the center of the viewport. The width of the area is two thirds of the screen width, and the height is equal to the width. When the center point of the venue enters this triggering area, e.g. during map panning, `venueMapLayer:didShowVenue:` of `NMAVenue3dMapLayerDelegate` is called. `venueMapLayer:didHideVenue:` callback is triggered when the center point leaves the triggering area.

Note that to get the notifications, you must first set `shouldCheckVenuesInViewport` property to `YES`, for example:

```
// enabling venue visible notification
_venueMapLayer.shouldCheckVenuesInViewport = YES;
// ...
- (void)venueMapLayer:(NMAVenue3dMapLayer *)venueMapLayer
 didShowVenueWithController:(NMAVenue3dController *)controller
{
 // venue entered triggering area
}
```

Note: This feature does not cause much processing as checking is performed only once when map movement stops. The notification is not sent during continuous movement, even when there is a venue in the triggering area.

To change the current floor for a given venue, retrieve an `NMAVenue3dController` object by using `controllerForVenue:venue:` method, and then change the `level` value.

Note that you can enable animations for both venue selection and floor transitions by setting `animatesFloorChange` and `animatesVenueSelection` to `YES` in the map layer.

The following example shows how to add an `NMAMapMarker` to a space upon a `didSelectSpace` event, and how to remove it when the space is deselected.

```
NMAMapMarker *_marker;

- (void)venueMapLayer:(NMAVenue3dMapLayer *)venueMapLayer
       didSelectSpace:(NMAVenue3dSpace *)space
              inVenue:(NMAVenue3dVenue *)venue
{
    [self removeMarker];
    UIImage *image = [UIImage imageNamed:@"markerIcon"];
    NMAImage *icon = [NMAImage imageWithUIImage:image];
    _marker = [NMAMapMarker mapMarkerWithGeoCoordinates:space.geoCenter icon:icon];
    [_marker setAnchorOffsetUsingLayoutPosition:NMALayoutPositionBottomCenter];
    _marker.mapLayerType = NMAMapLayerTypeForeground;
    _marker.zIndex = 100;
    [self.mainVC.activeMapView addMapObject:_marker];
}

- (void)venueMapLayer:(NMAVenue3dMapLayer *)venueMapLayer
     didDeselectSpace:(NMAVenue3dSpace *)space
              inVenue:(NMAVenue3dVenue *)venue
{
    [self removeMarker];
}

- (void)removeMarker
{
    if (_marker != nullptr) {
        [self.mainVC.activeMapView removeMapObject:_marker];
```

```
        _marker = nullptr;
    }
}
```

NMAVenue3dService

Venue Service and the venue features can be used without a map. To use the venue service, obtain an `NMAVenue3dService` instance by using `sharedVenueService` method and start the service using `start` method. To ensure `NMAVenue3dService` initialization has successfully completed, check the initialization result in `venueServiceDidInitialize:withResult:` callback method in `NMAVenue3dServiceListener`. For example:

```
- (void)venueServiceDidInitialize:(NMAVenue3dService *)venueService
  withResult:(NMAVenue3dServiceInitializationStatus)result
{
 // initialized
 if (result == NMAVenue3dServiceInitializationStatusOnlineSuccess) {
  // init ok
 } else if (result == NMAVenue3dServiceInitializationStatusOfflineSuccess) {
  // failed to authenticate but cached content available
 } else {
  // something else has gone wrong
 }
}

- (void)venueService:(NMAVenue3dService *)venueService didGetVenue:(NMAVenue3dVenue *)venue
{
 // venue has been loaded
 _venue = venue;
}
```

`NMAVenue3dService` offers methods for searching and loading venues without using a map. For example, the code below retrieves the closest venue inside a given radius near a given location. The area can also be defined by `NMAGeoBoundingBox` rather than a radius.

```
- (void)loadClosestVenue
{
 NMAVenue3dService *venueService = [NMAVenue3dService sharedVenueService];
 NMAGeoCoordinates *myLocation = [NMAGeoCoordinates geoCoordinatesWithLatitude:60.43704
  longitude:22.212710];
 float radiusInMeters = 5000.0f;
 NMAVenue3dVenueInfo *closetVenueInfo = [venueService venueAtGeoCoordinates:myLocation
  radius:radiusInMeters];
 [venueService getVenueWithInfo:closetVenueInfo];
}
```

For more information consult the API Reference. The `NMAVenue3dService` is also invoked when `NMAVenue3dMapLayer` is used so it can be used in a similar manner.

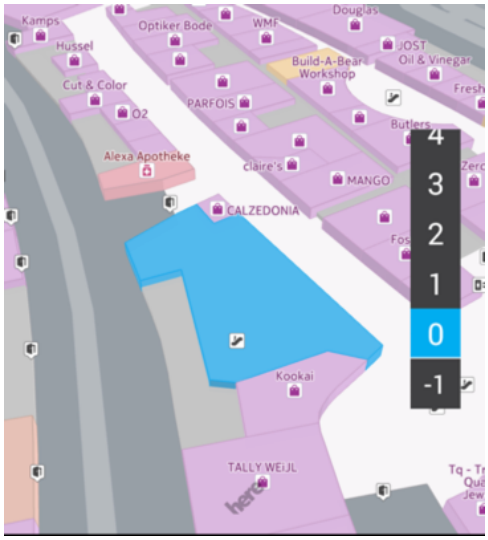
Working with Venues

An `NMAVenue3dVenue` object consists of one or more `NMAVenue3dLevel` objects. The `NMAVenue3dLevel` objects represent physical levels of the venue. Each `NMAVenue3dLevel` object consists of one or more `NMAVenue3dOuterArea` objects. For example, a building with common ground level areas can contain two separate towers on top of that common area, and so there would be two `NMAVenue3dOuterArea` objects in higher levels in that venue. An `NMAVenue3dOuterArea` consists of one or more `NMAVenue3dSpace` objects.

`NMAVenue3dVenue`, `NMAVenue3dOuterArea` and `NMAVenue3dSpace` objects can be interacted by the user through tapping. For an opened venue use `venueMapLayer:didSelectVenue:` callback in `NMAVenue3dMapLayerDelegate`. For a selected space use `venueMapLayer:didSelectSpace:inVenue:` callback. Selected spaces are highlighted with different colors in an opened venue.

Both `NMAVenue3dVenue` and `NMAVenue3dSpace` objects contain `NMAVenue3dContent` objects. `NMAVenue3dContent` encapsulates information related to the object such as name, address, other contact information, and category of the venue or space. There is also a concept of selected floor, which is the same as the visible floor. This is demonstrated in the following screenshot. Please note that the floor selection widget in this screenshot is not part of HERE SDK.

Figure 84: Selected Space



Open Mode

When open mode is enabled, venues that are in the viewport are opened automatically rather than requiring the user to click on the venue to open it. The venue closest to the center of the screen is always selected.

Open mode can be enabled on `NMAVenue3dMapLayer`.

```
venueMapLayer.openMode = YES;
BOOL isOpenMode = venueMapLayer.openMode;
```

Dynamic Styles

`NMAVenue3dStyleSettings` encapsulates the parameters that have an impact on the visual appearance of opened venues. You can set space names, icons, and colors by using this object. The fill and outline colors can be also set separately for selected and unselected spaces.

The following example shows how to set the name, label, fill color, and outline color to the given `NMAVenue3dSpace` object. The code snippet does not contain completed code but assumes that variables have been initialized.

```
- (void)updateStylesFor:(NMAVenue3dSpace *)space
                inVenue:(NMAVenue3dVenue *)venue
{
```

```
    UIColor* selectedColor =
  [UIColor colorWithRed:1.0f green: 0.0f blue:0.0f alpha:1.0f];
    UIColor* unselectedColor =
  [UIColor colorWithRed:1.0f green: 1.0f blue:0.0f alpha:1.0f];
    UIColor* outlineColor =
  [UIColor colorWithRed:0.0f green: 0.0f blue:1.0f alpha:1.0f];

    NMAVenue3dController *controller = _venueMapLayer.venueController;
    NMAVenue3dStyleSettings *settings = [[NMAVenue3dStyleSettings alloc] init];

    settings.labelName = @"My Space";
    settings.labelImage = [NMAImage imageWithUIImage:[UIImage imageNamed:@"MyLabel.png"]];

    settings.selectedFillColor = selectedColor;
    settings.fillColor = unselectedColor;
    settings.outlineColor = outlineColor;

    [controller setStyleSettings:settings forSpace:space];
}
```

Nearby Spaces

You can find all spaces in a radius around a given position by using an `NMAVenue3dLevel` or an `NMAVenue3dOuterArea` object. The position needs to be given as a geocoordinate and the radius in meters. The returned list of spaces contains all spaces that fall within or intersect the radius.

```
NMAGeoCoordinates *myLocation =
 [NMAGeoCoordinates geoCoordinatesWithLatitude:60.43704 longitude:22.212710];
NSArray nearbySpaces = [level nearbySpacesAroundPosition:myLocation withinRadius:10.0];
```

Area at Position

You can retrieve areas in a level by specifying a position. The area returned will either be an `NMAVenue3dSpace` or an `NMAVenue3dOuterArea`. Similarly, you can also get spaces in an `NMAVenue3dOuterArea`.

In case of nested spaces the innermost nested space encompassing the position is returned.

```
NMAGeoCoordinates *myLocation =
 [NMAGeoCoordinates geoCoordinatesWithLatitude:60.43704 longitude:22.212710];
NMAVenue3dArea *area = [level areaAtPosition:myLocation];
NMAVenue3dSpace *space = [outerArea spaceAtPosition:myLocation];
```

Frequently Asked Questions

You can find additional information about 3D Venues in *3D Venues FAQ* on page 211.

Private Venues

This feature allows you to use a different source of venue data in addition to or instead of the default HERE backend. The private content backend must be configured by HERE, which is an operation transparent to a developer using HERE iOS SDK. Access to private venue data is at the discretion of its legal owner and by definition it is not public.

Note: For more information about configuring a private venue data backend see *Service Support* on page 11 to contact us for more details .

If a private backend has been configured, call `setPrivateContent:` method on the `NMAVenue3dService` class at application initialization to indicate that you want your application to use it.

The code below demonstrates a call to this method. Note that the code does not show the entire initialization sequence necessary before you can start using HERE venue maps.

```
// Earlier initialization steps ...
// Assumption: NMAVenue3dMapLayer has been initialized

NMAVenue3dService service3d = [NMAVenue3dService sharedVenueService];
service3d.setPrivateContent:YES;
// Remember to start or restart Venue Service after setting private content

// Further implementation code here
...
```

Dynamic Content

By default HERE SDK uses public HERE 3D venue content. It is possible to use customer-specific content instead by using `privateContent` property on `NMAVenue3dService` as shown in the examples below. It is also possible to use both private and public content together and define which one has priority. If this kind of combined content is needed, use `combinedContent` property.

```
// Obtain Venue Service
NMAVenue3dService *venueService = [NMAVenue3dService sharedVenueService];

// Use only HERE SDK content (this is the default behavior)
[venueService setPrivateContent:false];
[venueService setCombinedContent:false];

// Use only private content
[venueService setPrivateContent:true];
[venueService setCombinedContent:false];

// Prefer HERE SDK content and use private as an alternative
[venueService setPrivateContent:false];
[venueService setCombinedContent:true];

// Prefer private content and use HERE SDK as an alternative
[venueService setPrivateContent:true];
[venueService setCombinedContent:true];
```

Multiple `NMAVenue3dService` objects can run at the same time. For example, if some part of an application requires access to only private content and another part requires HERE SDK content, two `NMAVenue3dService` objects can be instantiated and configured differently. Activities in one service do not have an impact on another service. For example, notifications related to loading (`venueService:didGetVenue:withVenueInfo:withStatus:`) are sent only to the client that initiated the loading. To obtain an additional `NMAVenue3dService` object, use `initAdditionalVenueService` method.

```
// get the main instance
NMAVenue3dService *mainInstance = [NMAVenue3dService sharedVenueService];
// get an additional instance
NMAVenue3dService *additionalInstance = [mainInstance initAdditionalVenueService];
```

For more information please see the API Reference.

Note: `NMAVenue3dService` is also invoked when `NMAVenue3dMapLayer` is used. As such, some venue features can be used in a common manner between these classes.

Venue Routing

HERE iOS SDK extends its 3D venue maps functionality to provide indoor routing. The SDK supports the following use cases:

- Routing from store A to store B within a venue
- Routing from an outside point to a point in a venue
- Routing from a point in a venue to an outside point
- Routing from a venue to another venue, with endpoints being a store or a point in a venue

Both online and offline routing are supported.

Please note that if HERE has no routing information for an area between the outdoor part of the route and the venue entry point, the route visualization represents the unknown section of the route with a dotted line.

Routing Between Locations in a Venue

This section demonstrates how to calculate and display an indoor route by using a code example. The example is based on a scenario where a device user wants to find out how to reach another location in the same venue. In real life the user would select the starting point and destination for the route by tapping on the map of the venue. However, for the sake of simplicity, the code below calculates a route from the first space (shop or Point of Interest) in the venue and to a destination which is the last space in the venue.

Note: See *Getting Indoor Location Based on a Tap Point* section for a code example of how to handle tap events to get a location for indoor venue routes.

The following code shows the implementation and is assumed to be part of an application. The code comments explain each step.

```
// Get a routing controller object
NMAVenue3dRoutingController *routingController =
  _venueMapLayer.venueRoutingController;

// Set a listener for the "route calculation completed" message.
[routingController addObserver:id<NMAVenue3dRoutingControllerObserver>(self)];

// Get a pointer to a venue the user selected/tapped.
NMAVenue3dVenue* venue3d = _testVenue;
NSArray *spaces = venue3d.spaces;

// Route start is the first space in the array of spaces. (An
// alternative is to get a space from didSelectSpace event.)
NMAVenue3dSpace *startSpace = [spaces objectAtIndex:0];

// Set route start:
NMAVenue3dSpaceLocation *startLocation =
   [NMAVenue3dSpaceLocation spaceLocationWithSpace:startSpace
   inVenue:venueMapLayer.venueController];

// Get route destination - the last element from the array of spaces.
NMAVenue3dSpace *endSpace = [spaces objectAtIndex:spaces.count-1];

// Set route destination:
NMAVenue3dSpaceLocation *endLocation =
   [NMAVenue3dSpaceLocation spaceLocationWithSpace:endSpace
   inVenue:venueMapLayer.venueController];
```

```
// To initialize routing options, first set routing mode:
NMARoutingMode *routingMode;

// ... We want the shortest pedestrian route and only one result.
routingMode = [[NMARoutingMode alloc] initWithRoutingType:NMARoutingTypeShortest
   transportMode:NMATransportModePedestrian
   routingOptions:0];

// Set routing options, using the configured routing mode:
NMAVenue3dRouteOptions *routeOptions =
   [NMAVenue3dRouteOptions optionsWithRoutingMode:routingMode];

// Calculate route - this is an asynchronous call, once the calculation is done
// we receive the didCalculateRoute message.
[routingController calculateRouteFrom:startLocation to:endLocation
   withParams:routeOptions];

...
#pragma mark - NMAVenue3dRoutingControllerDelegate
// Callback invoked when the route calculation is done to display the route passed
// to it as an argument.
- (void)didCalculateRoute:(NMAVenue3dCombinedRoute *)combinedRoute
{
 // Show route
 [_venueMapLayer.venueRoutingController showRoute: combinedRoute];
}
```

Routing Between Venues

Routing from one venue to another is also possible. In this case the start and end locations have different `NMAVenue3dController` objects and this is demonstrated below:

```
// In this example startVenueController and endVenueController are assumed to contain
// proper NMAVenue3dController references to the start and destination venues
// Set route start:
NMAVenue3dSpaceLocation *startLocation =
   [NMAVenue3dSpaceLocation spaceLocationWithSpace:startSpace
            inVenue:startVenueController];
// Set route end:
NMAVenue3dSpaceLocation *endLocation =
   [NMAVenue3dSpaceLocation spaceLocationWithSpace:startSpace
            inVenue:endVenueController];
// For other parts see the previous example
```

Routing to an Indoor Endpoint

The following is an example of how to implement a route calculation where the starting point is outdoors and the destination indoors.

```
// Set route start:
NMAGeoCoordinates *startPosition =
   [NMAGeoCoordinates geoCoordinatesWithLatitude:52.517072 longitude:13.411232];
NMAVenue3dOutdoorLocation *startLocation =
   [NMAVenue3dOutdoorLocation outdoorLocationWithGeoCoordinates:startPosition];

// Set route end, using the first object inside a venue:
NMAVenue3dVenue* venue3d = _venueMapLayer.venueController.venue;
NSArray *spaces = venue3d.spaces;
NMAVenue3dSpace *space = [spaces objectAtIndex:0];
```

```
NMAVenue3dSpaceLocation *endLocation =
   [NMAVenue3dSpaceLocation spaceLocationWithSpace:space
   inVenue:_venueMapLayer.venueController];

// Set routing options. We want the shortest car route.
NMARoutingMode *routingMode =
   [[NMARoutingMode alloc] initWithRoutingType:NMARoutingTypeShortest
   transportMode:NMATransportModeCar
   routingOptions:0];

// Calculate the route:
[routingController calculateRouteFrom:startLocation
   to:endLocation
   withParams:[NMAVenue3dRouteOptions optionsWithRoutingMode:routingMode]];
```

Routing Using an Arbitrary Indoor Location

It is also possible to use an arbitrary indoor location that is not at a store or designated space as a route endpoint. An example of this kind of location is a point in a corridor. The next code snippet demonstrates the initialization of these points.

```
NMAVenue3dController* _venueController;
float _tapX;
float _tapY;
// ...

// Get a routing controller object
NMAVenue3dRoutingController *routingController =
_venueMapLayer.venueRoutingController;

// Create a free point location to be used as a start location
NMAVenue3dLevelLocation* startLocation =
 [_venueController getLocationAtX:tapX Y:tapY WithSpacePrefered:false];

// Create outdoor location based on geocoordinates
NMAVenue3dOutdoorLocation* endLocation =
 [NMAVenue3dOutdoorLocation outdoorLocationWithGeoCoordinates:_destination];

// Set routing mode. We want the shortest car route.
NMARoutingMode *routingMode =
 [[NMARoutingMode alloc] initWithRoutingType:NMARoutingTypeShortest
          transportMode:NMATransportModeCar
         routingOptions:0];

// Set routing options. We want to avoid stairs, and prefer corridors.
NMAVenue3dRouteOptions* options =
   [NMAVenue3dRouteOptions optionsWithRoutingMode:routingMode];
if (_routeOptionsVC)
{
 options.avoidStairs = YES;
 options.preferCorridors = YES;
}

// Calculate the route:
[routingController calculateRouteFrom:startLocation
   to:endLocation
   withParams:[NMAVenue3dRouteOptions
   optionsWithRoutingMode:routingMode]];
```

Venue Route Options

`NMAVenue3dRouteOptions` encapsulate options used in indoor routing. It is possible to set many parameters related to visualization of the route line (for example, color, line width, and whether start and end flags are visible) as well as parameters related to how the route is calculated (for example, if elevators are allowed, if stairs are allowed, or if corridors are preferred).

Getting Indoor Location Based on a Tap Point

The next code example shows how to add a route point to an indoor route using `didReceiveTapAtLocation` method of `NMAMapGestureDelegate`.

```
- (void)mapView:(NMAMapView *)mapView didReceiveTapAtLocation:(CGPoint)tapLocation
{
    // convert tap point to NMAGeoGoordinate
    NMAGeoCoordinates *coords = [mapView geoCoordinatesFromPoint:tapLocation];

    // If any venue is selected, get related venue controller
    NMAVenue3dController* venueController = _venueMapLayer.venueController;

    // If no venue was selected, consider tapped location as outdoor location
    // and add it as route point.
    if (venueController == nil) {
        NMAVenue3dOutdoorLocation *loc = [NMAVenue3dOutdoorLocation
 outdoorLocationWithGeoCoordinates:coords];
        [self addToRoute:loc];
        return;
    }

    // Otherwise find location inside a venue and add it as route point.
    else {
        NMAVenue3dBaseLocation* loc = [venueController getLocationAtX:tapLocation.x
                                                                    Y:tapLocation.y
                                                        WithSpacePrefered:true];
        if ([loc isValid] && ![loc isKindOfClass:[NMAVenue3dOutdoorLocation class]]) {
            [self addToRoute:loc];
        }
    }
}
-(void)addToRoute:(NMAVenue3dBaseLocation*) baseLocation
{
    // do something with the route
}
```

Calculating Route Length

The following code example shows how the total length of a route can be calculated:

```
- (void)didCalculateRoute:(NMAVenue3dCombinedRoute *)combinedRoute
{
    double distance = 0.0;
    NSArray *routeSections = combinedRoute.routeSections;

    for (id routeSection in routeSections) {
        if ([routeSection class] == [NMAVenue3dVenueRouteSection class]) {
            NMAVenue3dVenueRouteSection *section = (NMAVenue3dVenueRouteSection *)routeSection;
            NSArray *maneuvers = section.routeManeuvers;
            NMAVenue3dRouteManeuver *lastManeuver = [maneuvers lastObject];
            distance += [lastManeuver distanceFromStart];
        } else if ([routeSection class] == [NMAVenue3dLinkRouteSection class]) {
            NMAVenue3dLinkRouteSection *section = (NMAVenue3dLinkRouteSection *)routeSection;
```

```
            NMAGeoCoordinates *start = section.from;
            NMAGeoCoordinates *destination = section.to;
            distance += [start distanceTo:destination];
        } else if ([routeSection class] == [NMAVenue3dOutdoorRouteSection class]) {
            NMAVenue3dOutdoorRouteSection *section = (NMAVenue3dOutdoorRouteSection *)routeSection;
            NMARoute *route = section.route;
            distance += route.length;
        }
    }
    // do something with distance information
}
```

Bounding Boxes for Parts of Venue Routes

Venue and outdoor route sections provide axis-aligned bounding boxes for the route. Axis-aligned bounding boxes are provided for individual route segments for each level. These methods return `NIL` if a route has no segment on the given level.

```
NMAGeoBoundingBox *obb = outdoorRouteSection.boundingBox;
NMAGeoBoundingBox *vbb = venueRouteSection.boundingBox;
NMAGeoBoundingBox *lbb = [venueRouteSection boundingBox:level];
```

The bounding box for the venue route (`vbb` in the example) also provides altitude information that may be extracted using `topLeftFront` and `bottomRightBack` properties of `NMAGeoBoundingBox`.

Venue Navigation

This section gives an overview of the classes and interfaces associated with 3D Venues navigation features.

Important: Venue Navigation is currently offered as a beta feature. APIs may change without notice.

Natural Guidance for Venue Maneuvers

Venue maneuvers provide the names of the closest POIs for natural guidance purposes. For each maneuver, this is the closest POI within a natural guidance radius around the position of the maneuver. If no POI exists within this radius, an empty string is returned. The natural guidance radius is a global parameter common to all maneuvers that may be set and queried by the user.

Note: For more information about natural guidance see *Turn-by-Turn Navigation for Walking and Driving* on page 159.

```
// Set the max distance a POI may have to a maneuver to be considered for natural guidance to 10
 meters
NMAVenue3dRouteManeuver.naturalGuidanceRadius = 10.0;
// Get the natural guidance POI of a maneuver
float naturalGuidanceRadius = NMAVenue3dRouteManeuver.naturalGuidanceRadius;
// If there is no suitable POI in the specified radius around the maneuver,
// the returned string will be empty.
NSString *naturalGuidance = myManeuver.naturalGuidancePOI;
```

Indoor Navigation

Your app can use the venue navigation manager to receive indoor navigation events.

The venue navigation manager is obtained by a call to `NMAVenue3dMapLayer.venueNavigationManager`. From this point on let's assume that the variable `venueNavigationManager` points to the venue navigation manager.

The following is an example of how to listen for indoor navigation events:

```
#import "NMAVenue3dCombinedRoute.h"
#import "NMAVenue3dNavigationManager.h"
#import "NMAVenue3dRouteManeuver.h"
#import "NMAVenue3dVenueRouteSection.h"

@interface VenueNavigationExample<NMAVenue3dNavigationListener>

/**
 * Called when the destination of turn-by-turn navigation is reached.
 *
 * When the destination is reached, NMAVenue3dNavigationManager -stop is automatically
 * called. When this callback is received, the navigation manager state will be
 * NMAVenue3dNavigationStateIdle.
 *
 */
- (void)navigationManagerDidReachDestination:(nonnull NMAVenue3dNavigationManager *)navigationManager;

/**
 * Called when the current (upcoming) maneuver is updated.
 *
 * \param navigationManager The NMAVenue3dNavigationManager object.
 * \param maneuver The current (upcoming) maneuver to be made.
 * \param nextManeuver The maneuver to be made AFTER THE CURRENT MANEUVER.
 *
 * \note The "current" maneuver is the upcoming, or next, maneuver to be taken. The "next"
 * maneuver is actually the maneuver to be taken after the current maneuver.
 */
- (void)navigationManager:(nonnull NMAVenue3dNavigationManager *)navigationManager
    hasCurrentManeuver:(nullable NMAVenue3dRouteManeuver *)maneuver
    nextManeuver:(nullable NMAVenue3dRouteManeuver *)nextManeuver;

/**
 * Called when the navigation manager loses its indoor position.
 *
 * \param navigationManager The NMAVenue3dNavigationManager object.
 */
- (void)navigationManagerDidLosePosition:(nonnull NMAVenue3dNavigationManager *)navigationManager;

/**
 * Called when the navigation manager finds its indoor position.
 *
 * \param navigationManager The NMAVenue3dNavigationManager object.
 */
- (void)navigationManagerDidFindPosition:(nonnull NMAVenue3dNavigationManager *)navigationManager;

/**
 * Called when a change is made to the route section being navigated.
 *
 * This can occur after successful rerouting due to the user leaving the current route (see
 * navigationManager:navigationManagerWillReroute).
 *
 * \param navigationManager The NMAVenue3dNavigationManager object.
 * \param routeSection NMAVenue3dVenueRouteSection representing the route section that was set.
 * \param combinedRoute NMAVenue3dCombinedRoute representing the current route.
 */
- (void)navigationManager:(nonnull NMAVenue3dNavigationManager *)navigationManager
 didUpdateRouteSection:(nonnull NMAVenue3dVenueRouteSection *)routeSection
    inCombinedRoute:(nonnull NMAVenue3dCombinedRoute *)combinedRoute;

/**
 * Called when rerouting is triggered due to the user leaving the current route section.
 *
 * If a new route section is successfully calculated, it is immediately applied to
 * the current navigation session and navigationManager:didUpdateRouteSection: is called.
 * After rerouting, navigationManager:didUpdateRouteSection: is called.
 *
 * \param navigationManager The NMAVenue3dNavigationManager object.
 */
```

```
- (void)navigationManagerWillReroute:(nonnull NMAVenue3dNavigationManager *)navigationManager;

/**
 * Called when rerouting, due to the user leaving the current route section, has finished.
 *
 * This method just means an attempt to reroute finished and does not guarantee that
 * a new route was successfully created.
 *
 * \param navigationManager The NMAVenue3dNavigationManager object.
 */
- (void)navigationManagerDidReroute:(nonnull NMAVenue3dNavigationManager *)navigationManager;

@end

- (void)navigationManager:(nonnull NMAVenue3dNavigationManager *)navigationManager
    hasCurrentManeuver:(nullable NMAVenue3dRouteManeuver *)maneuver
    nextManeuver:(nullable NMAVenue3dRouteManeuver *)nextManeuver
{
 NSString *guidance = maneuver.naturalGuidancePOI;
 if ([guidance length] == 0) {
  NSLog(@"maneuver without natural guidance");
 } else {
  NSLog(@"maneuver natural guidance: %@", guidance);
 }
}
```

This class can be registered with the venue navigation manager by invoking `[venueNavigationManager addListener:[[VenueNavigationExample alloc] init]]`.

Combined Indoor and Outdoor Navigation

Your app can register with the combined navigation manager to receive combined navigation events. Combined navigation events are used when a route contains at least one indoor and one outdoor segment.

For example, the venue navigation manager is obtained by a call to `NMAVenue3dMapLayer.combinedNavigation`. From this point on let's assume that the variable `combinedNavigationManager` points to the venue navigation manager.

The following is an example of how to listen for combined navigation events:

```
#import "NMAVenue3dCombinedNavigation.h"
#import "NMAVenue3dCombinedRoute.h"
#import "NMAVenue3dLinkRouteSection.h"
#import "NMAVenue3dMapLayer.h"
#import "NMAVenue3dOutdoorRouteSection.h"
#import "NMAVenue3dVenueRouteSection.h"

@interface CombinedNavigationExample<NMAVenue3dCombinedNavigationListener>

@property NMAVenue3dMapLayer *venueLayer;

/**
 * Called when the destination of turn-by-turn navigation is reached.
 *
 * When the destination is reached, NMAVenue3dCombinedNavigation -stop is automatically
 * called.
 */
- (void)navigationManagerDidReachDestination:(nonnull NMAVenue3dCombinedNavigation *)navigationManager

/**
 * Called when a change is made to the combined route being navigated.
 *
 * This can occur after successful rerouting due to the user leaving the current route (see
 * navigationManager:navigationManagerWillReroute).
 *
 * \param navigationManager The NMAVenue3dCombinedNavigation object.
 * \param combinedRoute NMAVenue3dCombinedRoute representing the current route.
 */
- (void)navigationManager:(nonnull NMAVenue3dCombinedNavigation *)navigationManager
   didUpdateCombinedRoute:(nonnull NMAVenue3dCombinedRoute *)combinedRoute;
```

```
/**
 * Called when an indoor section of the combined route is started.
 *
 * \param navigationManager The NMAVenue3dCombinedNavigation object.
 * \param indoorSection NMAVenue3dVenueRouteSection representing the next indoor section.
 * \param combinedRoute NMAVenue3dCombinedRoute representing the current route.
 */
- (void)navigationManager:(nonnull NMAVenue3dCombinedNavigation *)navigationManager
   willStartIndoorSection:(nonnull NMAVenue3dVenueRouteSection *)indoorSection
          inCombinedRoute:(nonnull NMAVenue3dCombinedRoute *)combinedRoute;

/**
 * Called when a link section of the combined route is started.
 *
 * \param navigationManager The NMAVenue3dCombinedNavigation object.
 * \param linkingSection NMAVenue3dLinkRouteSection representing the next link section.
 * \param combinedRoute NMAVenue3dCombinedRoute representing the current route.
 */
- (void)navigationManager:(nonnull NMAVenue3dCombinedNavigation *)navigationManager
  willStartLinkingSection:(nonnull NMAVenue3dLinkRouteSection *)linkingSection
          inCombinedRoute:(nonnull NMAVenue3dCombinedRoute *)combinedRoute;

/**
 * Called when an outdoor section of the combined route is started.
 *
 * \param navigationManager The NMAVenue3dCombinedNavigation object.
 * \param outdoorSection NMAVenue3dOutdoorRouteSection representing the next outdoor section.
 * \param combinedRoute NMAVenue3dCombinedRoute representing the current route.
 */
- (void)navigationManager:(nonnull NMAVenue3dCombinedNavigation *)navigationManager
  willStartOutdoorSection:(nonnull NMAVenue3dOutdoorRouteSection *)outdoorSection
          inCombinedRoute:(nonnull NMAVenue3dCombinedRoute *)combinedRoute;

@end

- (void)navigationManager:(nonnull NMAVenue3dCombinedNavigation *)navigationManager
   didUpdateCombinedRoute:(nonnull NMAVenue3dCombinedRoute *)combinedRoute
{
    // Show route
    NMAVenue3dRoutingController* routingController = venueLayer.venueRoutingController;
    [routingController showRoute:combinedRoute];
}
```

This class can be registered with the combined navigation manager by invoking `[combinedNavigationManager addListener:[[CombinedNavigationExample alloc] init]]`.

Audio and Haptic Feedback

It is possible to enable audio feedback (beeps) and haptic feedback (vibrations) for specific navigation events such as maneuver change, rerouting, and reaching destinations.

To enable audio feedback on maneuver events, invoke `venueNavigationManager.beepsEnabled = YES`.

To enable haptic feedback on maneuver events, invoke `venueNavigationManager.vibrationEnabled = YES`.

See the API documentation of `NMAVenueNavigationManager` for methods to query the current status of the audio and haptic feedback settings.

Maneuver Zoom

It is possible to enable automatically zooming in to a certain zoom level in the vicinity of a maneuver. This zooming in will only occur if the current zoom level is less than the zoom level specified for maneuver zoom.

For example, if the maneuver zoom level has been set to 19, but the user is already viewing the map at a zoom level of 20, no zooming in (or out) will occur.

You can enable maneuver zoom by invoking `venueNavigationManager.maneuverZoomEnabled = YES`.

Set the distance from a maneuver at which maneuver zoom will be activated to five meters by invoking `venueNavigationManager.maneuverZoomDistance = 5.0`.

Set the zoom level to which maneuver zoom will zoom in in the vicinity of a maneuver to 20 by invoking `venueNavigationManager.maneuverZoomLevel = 20`.

See the API documentation of `NMAVenue3dNavigationManager` for methods to query the current status of the maneuver zoom settings.

LiveSight

LiveSight enables user experiences that use the real world as an interface. With LiveSight, developers can overlay geospatial content on the real world which is displayed using the device camera. Additionally, an immersive experience is created by using the device sensors to track movement in space and update the view accordingly.

The key concepts covered in this section include adding LiveSight to an iOS application, starting LiveSight in Camera Mode, and customizing the LiveSight experience. The classes covered include `NMACompositeView` and `NMAARController`.

LiveSight requires iOS users to grant your application access to the back camera. If the camera is not available, `NMAARErrorCameraAccessNotAuthorized` error code is returned when starting LiveSight.

NMACompositeView

The `UIView` subclass related to LiveSight functionality is the `NMACompositeView`. `NMACompositeView` exposes LiveSight functionality in one iOS UI component. For the remainder of this section we use `NMACompositeView` in code samples and discussions.

Note: `NMACompositeView` does not support a map view. As such, the composite view only supports LiveSight, and it is rendered as a black screen until LiveSight is started. Support for using the map view along with the camera view will be added to `NMACompositeView` in a future release.

Adding and Initializing the NMACompositeView

The first step to integrate LiveSight functionality into an application is to insert an `NMACompositeView` into the view layout. This is accomplished by adding `NMACompositeView` subview to the current iOS view.

As with other `UIView` objects, the `NMACompositeView` must be initialized by calling `initWithFrame:` method. During this asynchronous initialization the `NMAARController` is also created. More information about `NMAARController` can be found in *Customizing LiveSight* on page 194.

Starting and Stopping LiveSight

To control LiveSight, two methods from `NMAARController`, `start` and `stopWithAnimation:`, are used. You can call `start` method after creating the composite view to trigger the LiveSight Mode. Calling `start`

while already in LiveSight Mode results in `NMAARErrorInvalidOperation` error code being returned. Use `stopWithAnimation:` method to stop LiveSight.

Figure 85: LiveSight

Adding and Interacting with LiveSight Content

This section covers how to add content to be displayed in LiveSight and how to handle user interactions with that content. The classes covered in this section are `NMAARObject` and `NMAARIconObject`. Additionally, several `NMAARController` methods and properties are used:

- `addObject:`
- `removeObject:`
- `pressObject:`
- `unpressObject:`
- `selectObject:`
- `deselectObject`
- `focusObject:`
- `defocusObject`
- `objectsAtPoint:`
- `objectsInRect:`

The following delegate protocols are also used alongside `NMAARController`:

- `NMAARControllerDelegate`
- `NMAARControllerGestureDelegate`

LiveSight Object Model

A LiveSight object has several visual representations. The representation to be displayed is decided based on the state of the object. The object state which influences display is the Focus state. The Focus state is discussed in detail later.

While in camera-enabled LiveSight view, there are two planes in which an object can be displayed, and the object representation is different in each. The planes are the "Front" plane and the "Back" plane. By default, objects which are geographically closer to the LiveSight center are displayed in the Front plane, and objects which are further away are displayed in the Back plane. Objects can be moved from one plane to the other using the vertical pan gesture.

Figure 86: Icons in the Camera View

Figure 87: Visualization of the Front and Back Planes

While in the Front plane, a LiveSight object is represented by both its icon as well as an information view (Info View) which extends out from the side of the icon. This information view is intended to act as a mechanism for displaying more detailed information about the object. In the Back plane, an object is initially represented by a single icon. It is possible to have an object in the Back plane display its Info View by putting it in focus. The icon for the Front plane and the Back plane can be different, and by default the transition from one plane to the other is animated.

NMAARObject Class

`NMAARObject` is the base class for all other objects which can be added to LiveSight in order to be displayed. It contains methods common to all LiveSight objects enabling the following operations:

- Set and retrieve the object current position
- Set and retrieve the object front and back icons
- Set and retrieve the object front and back icon sizes
- Set and retrieve the size, image, and extension state of the information view

Note: LiveSight only supports a single front icon size to be used at a time. Mixing objects with different front icon sizes results in icon distortion. The front icon size should be set via `setFrontPlaneIconSize:` method in `NMAARController`.

NMAARIconObject Class

The single child class to `NMAARObject` is `NMAARIconObject`. `NMAARIconObject` represents the object model described in *LiveSight Object Model* on page 190. Because it is the only concrete `NMAARObject`, most of its functions reside in `NMAARObject`.

Adding and Interacting with NMAARObject

Adding an `NMAARObject` to LiveSight is accomplished by way of `addObject:` method in `NMAARController`:

```
NMAGeoCoordinates *objLocation =
 [NMAGeoCoordinates geoCoordinatesWithLatitude:49.276744 longitude:-123.112049];

NMAARIconObject *iconObject =
 [NMAARIconObject iconObjectWithIcon:[NMAImage imageWithUIImage:image1]
  infoImage:[NMAImage imageWithUIImage:image2]
  coordinates:objLocation];

[arController addObject:iconObject];
```

Similarly, `NMAARObjects` can be removed using `removeObject:` method in `NMAARController`:

```
[arController removeObject:iconObject];
```

To facilitate interactivity with `NMAARObjects`, an `NMAARControllerGestureDelegate` can be registered by way of `gestureDelegate` property in `NMAARController`. When a tap event occurs, the `NMAARObject` at the tap point can be found through `arController:shouldProcessTouchUpOnObjects:atPoint:` method in `NMAARControllerGestureDelegate`. If you would like to provide custom handling for this gesture, you can implement the method as follows:

```
-(BOOL)arController:(NMAARController *)arController shouldProcessTouchUpAtPoint:(CGPoint)point
{
 // Get object closest to touch point if more than one
    NMAARObject* arObject =
     objects.count > 0 ? [objects objectAtIndex:0] : nil;

    if (arObject) {
        // perform some custom action such as focus on the object
    }

    return NO;
}
```

The `NMAARObject` can be put into focus by way of `focusObject:` method. While in focus, an info pane for an `NMAARObject`, which is in the back plane, is displayed. Only one `NMAARObject` may have focus at a time. To defocus an `NMAARObject`, call `focusObject:` on another `NMAARObject`. You can also call `defocusObject` method to defocus from the focused `NMAARObject`.

In addition to event driven `NMAARObject` retrieval, `objectsAtPoint:` and `objectsInRect:` methods can be used to programmatically get `NMAARObject` at a screen location:

```
CGPoint point = {50, 50};
NSArray* objectsAtPoint = [arController objectsAtPoint:point];
```

```
CGRect viewRect = CGRectMake(50, 50, 25, 25);
```

```
NSArray* objectsInViewRect = [arController objectsInRect:viewRect);
```

Selecting NMAARObjects

After retrieving an `NMAARObject`, you can choose to select and deselect it by calling `selectObject:` and `deselectObject` methods. Selecting an object causes an object Info image to collapse, and the Back plane image to replace the Front plane image (if the object is in the Front plane).

Note: A single `NMAARObject` cannot be focused and selected simultaneously. However, it is possible to have one `NMAARObject` focused and another `NMAARObject` selected at the same time.

Reading the Current Pose

`NMAARController` provides a convenient way to retrieve the current positional and directional (pose) values of your LiveSight session. By using `poseReading` property you can retrieve the `NMAARPoseReading` instance, which contains the following values:

- Heading (Yaw)
- Pitch
- Roll
- Location (Latitude, Longitude, Altitude)
- Timestamp

Although these `NMAARPoseReading` values are derived from device sensors, they are interpolated and smoothed by the LiveSight engine.

3D Objects

LiveSight also supports two types of 3D objects: `NMAARBillboardObject` and `NMAARMeshObject`. Both of these classes are derivatives of `NMAARModelObject`, which provides the ability to control basic properties such as object opacity, scale, and rotation. Because `NMAARModelObject` classes do not derive from `NMAARObject` class, you cannot use them with some `NMAARController` methods such as `focus` or `press`. To add or remove an `NMAARModelObject`, use `addModelObject:` and `removeModelObject:` methods in `NMAARController`.

An `NMAARBillboardObject` can be oriented in the Up state in two ways. In `FIXED` orientation mode it may be "attached" to a surface by specifying the up and normal vectors using `upDirection` and `surfaceNormal` properties. In `BILLBOARD` orientation mode a billboard is set to be always upright and facing the camera. An `NMAARBillboardObject` can be positioned in one of the following ways:

- Anchored to a specific geo location using `initWithGeoCoordinates:texture:` initialization method, as well as `geoPosition` property.
- Anchored relative to the camera using `initWithLocalPosition:texture` initialization method and `localPosition` property. The `NMAVector3d` represents the location of the billboard center in meters away from the camera.

An `NMAARMeshObject` represents a 3D object mesh. As with an `NMAARBillboardObject`, a mesh object may be anchored to a geo location or relative to the screen. You can control the orientation of the `NMAARMeshObject` by providing `NMAGeoCoordinates` that the object can point towards.

Customizing LiveSight

LiveSight includes customization options that allow developers and designers to create many different and immersive experiences. `NMAARController` class serves as a facade for overall LiveSight functionality containing all of the methods and callbacks available for controlling and customizing LiveSight behavior.

NMAARCameraParameters

`cameraParameters` property, of type `NMAARCameraParameters`, encapsulates parameters that are related to camera-enabled LiveSight view. `size` property allows you to set the camera resolution to be used for LiveSight camera view. Please note that using a high camera resolution may cause performance degradation. The default pixel resolution is `680x480`. A list of supported resolutions can be retrieved by using `supportedSizes` property.

Other NMAARController Settings

In addition to `NMAARCameraParameters`, you can use other properties and methods in `NMAARController` to customize the following areas in LiveSight:

- Alternative Center Location — LiveSight is not only limited to the current device location. It is possible to use `alternativeLocation` property to set an alternative location ("space shift") for your LiveSight experience.
- Camera View Configuration — Adjust settings related to the object display such as only showing Front items.
- Layout Updates — By default LiveSight icons are set to update dynamically according to the current device position. However, you can optionally set `NMAARController` so that the Layout (containing the front and back icons) does not update until the device position has changed significantly past a threshold.

Platform Data Extension

Platform Data Extension (PDE) provides the ability to easily access the Platform Data Extension API from HERE iOS SDK. You can use this extension to access a wide range of data that can be later used for different use cases. Some examples include displaying road elevation, slopes, and traffic signs. For more information about use cases and the types of data that can be accessed through PDE, check *Platform Data Extension API Developer's Guide* .

PDE Thematic Layers

PDE divides map content across many thematic layers. Each thematic data layer serves a specific use case and only contains the data required for it, such as road elevation. To use PDE, you need to first decide on the required data for your app and select the PDE thematic layers accordingly. The available thematic layers can be found via the PDE *Layers API*. You can then check the targeted thematic layer via the individual *Layer API*. Before starting to use PDE in your app, you need to select the correct thematic layers, and then decide on what data to use and how to use it. This is a crucial step.

Note: It is common for routes to start on smaller roads, climb to bigger roads, stay on motorways for the main part, and finally step down to smaller roads again when approaching the destination. Since retrieving all information about smaller roads along the entire route requires a large amount of data, *road link*-related thematic layers are actually split into five layers each, corresponding to the functional road classes in HERE map. Functional Class 1 roads are generally motorways, while Functional Class 5 roads are small roads that are only used near a destination. To use these layers, you need to specify the tile layer by appending the functional class suffix "_FCx" where x is a number from 1 to 5. For example, ROAD_GEOM_FC1. If a layer isn't related to road links, such as the PSTLCB_GEN layer, you don't need to append the "_FCx" suffix and specify the tile layer.

PDE Classes

Class	Description
`NMAPlatformDataRequest`	Creates a PDE data request with the specified layers and `NMAGeoBoundingBox` object.
`NMAPlatformDataResult`	After a PDE data request the result is returned as an object of this type. Provides an extract method to convert the result into an `NSDictionary` object.
`NMAPlatformDataItemCollection`	Array of `NMAPlatformDataItem` objects. For example, assuming the result is an `NMAPlatformDataResult` type object, the `NMAPlatformDataItemCollection` for the `ROAD_GEOM_FC1` layer data can be accessed with `result[@"ROAD_GEOM_FC1"]`. This class also provides an extract method to convert the collection into an `NSArray` object.
`NMAPlatformDataItem`	After a PDE data request each layer data is returned with objects of this class. For example, assuming the item is an `NMAPlatformDataItem` object containing the `ROAD_GEOM_FC1` layer data, the name property can be accessed with `item[@"NAME"]`. If the specified property isn't found, `nil` is returned. Note that several properties shortcuts are available like `item.linkId` versus `item[@LINK_ID"]`. This class also provides extract method to convert the PDE data into an `NSDictionary` object.

Example: Requesting the PDE Data

The example below shows how a feature is implemented using the PDE data. The goal is to colorize each road segment according to its average height. For this feature you need the PDE data from `ROAD_GEOM_FC1` and `BASIC_HEIGHT_FC1` layers. As of now, the only way to request the PDE data requires the layers specified with an `NMAGeoBoundingBox`.

Note: The `ROAD_GEOM_FC[number]` and `BASIC_HEIGHT_FC[number]` layers are also referred to as "tile" layers since their data is split into multiple tiles. Due to server limitations, up to 15 tiles can be requested at a time. PDE also supports non-tile layers.

To use the data from both layers, you need to join the two thematic layers by using `LINK_ID` property. This is demonstrated in the next section.

The following are sample results from the Layer request.

ROAD_GEOM_FC1 Layer API result:

```
{
  "description": "Ungeneralized road, ferry and rail ferry geometry (polylines).<br/>If a road link
 crosses a tile boundary, it will be written into each of the tiles, each including the full link
 geometry. This simplifies use cases other than pure display of all geometry within a rectangle.",
  "attributes": {
    "LINK_ID": "Permanent link ID. Positive 64 bit Integer that globally identifies the road, carto
 or building footprint link, also across map releases. Link IDs are never reused.",
    "LONG_HAUL": "This link or polygon or POI is of major importance. It should be displayed at high
 zoom lavels, and it should be included for routing in/through regions where no detailed routing is
 supported.",
    "NAME": "A name of this road line. Roads can have multiple names, in the same or multiple
 languages. This field contains any of those.",
    "NAMES": "List of all names for this object, in all languages [...]",
    "TUNNEL": "Is this navigable link or railroad a tunnel?",
    "BRIDGE": "Is this navigable link or railroad a bridge?",
    "LAT": "Latitude coordinates [10^-5 degree WGS84] along the polyline. ",
    "LON": "Longitude coordinates [10^-5 degree WGS84] along the polyline. ",
    "ZLEVEL": "(-4 ... 11) indicates the height of the point relative to another point on a grade
 separated crossing with any other line. Comma separated. If z-level is null then the value '0' is
 left out."
  },
  "referencedStaticContents": [],
  "tileRequestsLevel": 9,
  "tileX": 499,
  "tileY": 403,
  "isStaticContent": false
}
```

BASIC_HEIGHT_FC1 Layer API result:

```
{
```

```
  "description": "Link height values computed from a Digital Terrain Model, cleaned up for
 continuity along links, bridges and tunnels. Less accurate than ADAS link height values, but full
 coverage and sufficient for certain use cases.",
  "attributes": {
    "LINK_ID": "Permanent link ID. Positive 64 bit Integer that globally identifies the road, carto
 or buildin footprint link, also across map releases. Link IDs are never reused.",
    "DTM_MIN_HEIGHT": "The minimum height [cm above WGS84 ellipsoid] encountered along the link.",
    "DTM_MAX_HEIGHT": "The maximum height [cm above WGS84 ellipsoid] encountered along the link.",
    "DTM_AVG_HEIGHT": "The average height [cm above WGS84 ellipsoid] along the link.",
    "DTM_REF_ZCOORD": "Height [cm above WGS84 ellipsoid] at the reference node of the link.",
    "DTM_NONREF_ZCOORD": "Height [cm above WGS84 ellipsoid] at the non-reference node of the link."
  },
  "referencedStaticContents": [],
  "tileRequestsLevel": 9,
  "tileX": 496,
  "tileY": 358,
  "isStaticContent": false
}
```

To begin using the PDE layers, create a request object of `NMAPlatformDataRequest` type:

```
NSSet<NSString *> *layers = [NSSet setWithObjects:@"ROAD_GEOM_FC1", @"BASIC_HEIGHT_FC1", nil];
NMAGeoBoundingBox *box = self.mapView.boundingBox;
NMAPlatformDataRequest *request = [[NMAPlatformDataRequest alloc] initWithLayers:layers
 geoBoundingBox:box];
```

If the request is found to be invalid, `nil` is returned. You can run the request with a block or listener:

```
// Is the request valid?
if (request) {
    [request startWithBlock:^(NMAPlatformDataRequest *request, NMAPlatformDataResult *result,
 NSError *error) {

        if (error) {
            NSLog(@"Error occurred!\n"
                   "Code: %ld\n"
                   "Description: %@\n"
                   "Failure reason: %@\n",
                  (long)error.code,
                  error.localizedDescription,
                  error.localizedFailureReason];
        }
        else {
            // Retrieved the result successfully, now process it
            RoadElevationProcessor *processor =
                [[RoadElevationProcessor alloc] initWithResult:result
                                                 andMapContainer:self.mapContainer];

            [processor process:^void (BOOL result, NSString *msg) {
                if (result) {
                    [self addElevationLegend];
                } else {
                    NSLog(@"Elevation data processing failed!");
                }

                if (msg) {
                    NSLog(msg);
                }
            }];
        }
    }];
}
else {
    NSLog(@"Invalid request!");
};
```

Example: Processing the PDE Data

After the data result is successfully retrieved, you need to process the data. In the example above you have asked for `ROAD_GEOM_FC1` and `BASIC_HEIGHT_FC1` data restricted with the map view bounding box. For tutorial purposes, we then implemented a sample data processor class, `PlatformDataProcessor`, which provides methods to join the layers and can be extended for consuming the joined data. In this example it is required to implement the colorize feature.

The following is the class declaration for `PlatformDataProcessor`:

```
/**
 * Defines the code block that returns a string given a data item object.
 * It is used to index data item objects for quickly joining the data.
 */
typedef NSString* (^NMAPlatformDataProcessorIndexerBlock)(NMAPlatformDataItem*);

/**
 * Defines the code block that returns a string array given a data item
 * object. It is used to index data item objects for quickly joining
 * the data.
 */
typedef NSArray<NSString*>* (^NMAPlatformDataProcessorMultiIndexerBlock)(NMAPlatformDataItem*);

/**
 * Defines the code block that runs after join a operation.
 * The first argument sets the result and the other one sets the message
 * if available.
 */
typedef void (^NMAPlatformDataProcessorResultBlock)(BOOL, NSString*);

/**
 * \brief This class is for joining the PDE data coming from different
 *        layers using the passed indexers.
 */
@interface PlatformDataProcessor : NSObject

/**
 * This method joins two different layers data using the indexers and returns
 * a dictionary of items.
 *
 * \param mainLayer The layer data that will joined with the other layer data.
 *
 * \param mainIndexer The indexer for quickly joining the main layer data.
 *
 * \param otherLayer The other data that will joined with the main layer data.
 *
 * \param otherIndexer The indexer for quickly joining the other layer data.
 *
 * \return The dictionary of items that are joined where the key is the PDE data
 *         coming from the main layer data and the value is the PDE data coming
 *         from the other layer data.
 */
- (NSDictionary<NMAPlatformDataItem*, NMAPlatformDataItem*>*)
    joinLayer:(NMAPlatformDataItemCollection *)mainLayer
    usingIndexer:(NMAPlatformDataProcessorIndexerBlock)mainIndexer
    withOtherLayer:(NMAPlatformDataItemCollection *)otherLayer
    usingIndexer:(NMAPlatformDataProcessorIndexerBlock)otherIndexer;

/**
 * This method joins two different layers data using the indexers and returns
 * a dictionary of items.
 *
 * \param mainLayer The layer data that will joined with the other layer data.
 *
```

```
 * \param mainMultiIndexer The indexer for quickly joining the main layer data.
 *
 * \param otherLayer The other data that will joined with the main layer data.
 *
 * \param otherMultiIndexer The indexer for quickly joining the other layer data.
 *
 * \return The dictionary of items that are joined where the key is the PDE data
 *         coming from the main layer data and the value is the array of PDE data
 *         coming from the other layer data.
 */
- (NSDictionary<NMAPlatformDataItem*, NSArray<NMAPlatformDataItem*>*>*)
    joinLayer:(NMAPlatformDataItemCollection *)mainLayer
    usingMultiIndexer:(NMAPlatformDataProcessorMultiIndexerBlock)mainMultiIndexer
    withOtherLayer:(NMAPlatformDataItemCollection *)otherLayer
    usingMultiIndexer:(NMAPlatformDataProcessorMultiIndexerBlock)otherMultiIndexer;

/**
 * This method processes the data joined.
 *
 * \note The derived classes must implement this method.
 *
 * \param block The block that runs after the join operation. Note that its first argument brings
 *              the result and the other one brings a message if available.
 */
- (void)process:(NMAPlatformDataProcessorResultBlock)block;
```

Note that `[PlatformDataProcessor joinLayer:usingIndexer:withOtherLayer:usingIndexer]` method is for inner joining the thematic layers with properties where more than one data item is possible. For example, the `TRAFFIC_SIGN_FCx` thematic layer has `LINK_IDS` property where as much as two `LINK_ID`s are contained.

The following is `PlatformDataProcessor` class implementation:

```
@interface PlatformDataProcessor ()
{
}

- (NSDictionary<NSString*, NMAPlatformDataItem*>*)
    map:(NMAPlatformDataItemCollection *)collection
    withIndexer:(NMAPlatformDataProcessorIndexerBlock)indexer;

- (NSDictionary<NSString*, NSArray<NMAPlatformDataItem*>*>*)
    multiMap:(NMAPlatformDataItemCollection *)collection
    withMultiIndexer:(NMAPlatformDataProcessorMultiIndexerBlock)indexer;

@end

@implementation PlatformDataProcessor

#pragma mark - Public methods

- (NSDictionary<NMAPlatformDataItem*, NMAPlatformDataItem*>*)
    joinLayer:(NMAPlatformDataItemCollection *)mainLayer
    usingIndexer:(NMAPlatformDataProcessorIndexerBlock)mainIndexer
    withOtherLayer:(NMAPlatformDataItemCollection *)otherLayer
    usingIndexer:(NMAPlatformDataProcessorIndexerBlock)otherIndexer
{
    NSDictionary<NSString*, NMAPlatformDataItem*>* mappedItems =
        [self map:otherLayer withIndexer:otherIndexer];
    NSMutableDictionary<NMAPlatformDataItem*, NMAPlatformDataItem*>* dictionary =
        [[NSMutableDictionary alloc] init];

    for (NMAPlatformDataItem *item in mainLayer) {
        NSString *key = mainIndexer(item);
        NMAPlatformDataItem *matchedItem = mappedItems[key];
```

```
        if (matchedItem) {
            dictionary[item] = matchedItem;
        }
    }

    return dictionary;
}

- (NSDictionary<NMAPlatformDataItem*, NSArray<NMAPlatformDataItem*>*>*)
    joinLayer:(NMAPlatformDataItemCollection *)mainLayer
    usingMultiIndexer:(NMAPlatformDataProcessorMultiIndexerBlock)mainMultiIndexer
    withOtherLayer:(NMAPlatformDataItemCollection *)otherLayer
    usingMultiIndexer:(NMAPlatformDataProcessorMultiIndexerBlock)otherMultiIndexer
{
    NSDictionary<NSString*, NSArray<NMAPlatformDataItem*>*>* mappedItems =
        [self multiMap:otherLayer withMultiIndexer:otherMultiIndexer];
    NSMutableDictionary<NMAPlatformDataItem*, NSMutableArray<NMAPlatformDataItem*>*>* dictionary =
        [[NSMutableDictionary alloc] init];

    for (NMAPlatformDataItem *item in mainLayer) {
        NSArray<NSString*> *array = mainMultiIndexer(item);

        for (NSString *key in array) {
            NSArray<NMAPlatformDataItem*> *matchedItems = mappedItems[key];

            if (matchedItems) {
                NSMutableArray<NMAPlatformDataItem*> *entry = dictionary[item];

                if (!entry) {
                    entry = [[NSMutableArray alloc] init];
                    dictionary[item] = entry;
                }

                [entry addObjectsFromArray:matchedItems];
            }
        }
    }

    return dictionary;
}

- (void)process:(NMAPlatformDataProcessorResultBlock)block
{
    NSLog(@"Derived classes must implement [PlatformDataProcessor process:] method!");
    abort();
}

#pragma mark - Private methods

- (NSDictionary<NSString*, NMAPlatformDataItem*>*)
    map:(NMAPlatformDataItemCollection *)collection
    withIndexer:(NMAPlatformDataProcessorIndexerBlock)indexer
{
    NSMutableDictionary<NSString*, NMAPlatformDataItem*>* dictionary =
        [[NSMutableDictionary alloc] init];

    for (NMAPlatformDataItem* item in collection) {
        NSString* key = indexer(item);

        dictionary[key] = item;
    }

    return dictionary;
}

- (NSDictionary<NSString*, NSArray<NMAPlatformDataItem*>*>*)
```

```
    multiMap:(NMAPlatformDataItemCollection *)collection
    withMultiIndexer:(NMAPlatformDataProcessorMultiIndexerBlock)indexer
{
    NSMutableDictionary<NSString*, NSMutableArray<NMAPlatformDataItem*>*>* dictionary =
        [[NSMutableDictionary alloc] init];

    for (NMAPlatformDataItem* item in collection) {
        NSArray<NSString*>* array = indexer(item);

        for (NSString* key in array) {
            if (!dictionary[key]) {
                dictionary[key] = [[NSMutableArray  alloc] init];
            }

            [dictionary[key] addObject:item];
        }
    }

    return dictionary;
}

@end
```

Next, create `RoadElevationProcessor`. `RoadElevationProcessor` class extends `PlatformDataProcessor` for coloring the road segments according to their average height. This class needs the result and a dedicated map container intended for the `NMAMapPolyline` objects. Note that the base class `PlatformDataProcessor` performs the inner joining of the thematic layer data depending on the indexer blocks provided. `RoadElevationProcessor` class utilizes the joined data to implement the intended feature.

Here's the declaration for `RoadElevationProcessor` class:

```
/**
 * \brief Extends the PlatformDataProcessor class for the road elevation
 *        handling.
 */
@interface RoadElevationProcessor : PlatformDataProcessor

/**
 * Creates a road elevation processor object.
 *
 * \param result The PDE road elevation data result to be processed.
 *
 * \param mapContainer The polylines that will be colored according to the average road
 *                     segments will be inserted to this map container.
 *
 * \return If the parameters are valid, a RoadElevationProcessor object configured with
 *         the parameters and nil otherwise.
 */
- (instancetype)initWithResult:(NMAPlatformDataResult *)result
               andMapContainer:(NMAMapContainer *)mapContainer;

@end;
```

The main method for `RoadElevationProcessor` class is the process method below:

```
- (void)process:(NMAPlatformDataProcessorResultBlock)block
{
    BOOL result = FALSE;
    NSString *msg;
    NSDictionary<NMAPlatformDataItem*, NMAPlatformDataItem*>* joinedItems =
        [self joinLayer:_result[@"ROAD_GEOM_FC1"]
            usingIndexer:^NSString* (NMAPlatformDataItem *item) {
```

```
                return item.linkId;
            }
            withOtherLayer:_result[@"BASIC_HEIGHT_FC1"]
            usingIndexer:^NSString* (NMAPlatformDataItem *item) {
                return item.linkId;
            }];

    // Any joined data?
    if (joinedItems && joinedItems.count) {
        // Set the message
        msg = [NSString stringWithFormat:@"Joined elevation items count = %ld\n",
            (unsigned long)joinedItems.count];

        // One-by-one check the joined data
        for (NMAPlatformDataItem *roadGeometry in joinedItems) {
            // Filter out the road geometries having less than 2 coordinates
            // as they show nothing on the map
            if (roadGeometry.coordinates.count > 1) {
                NMAPlatformDataItem *height = joinedItems[roadGeometry];
                NMAMapPolyline *polyline =
                    [NMAMapPolyline mapPolylineWithVertices:roadGeometry.coordinates];

                polyline.lineColor = [self getColorDependingOnHeight:height.averageHeightCm];
                polyline.lineWidth = 10;

                [_mapContainer addMapObject:polyline];
            }
        }

        result = TRUE;
    }

    // Done: pass the result along with the msg
    block(result, msg);
}
```

The first thing this method does is joining the data coming from two different thematic layers with the "indexer" blocks. Here the data is joined via `LINK_ID`s. Note that `linkId` property is available as a shortcut property in `NMAPlatformDataItem` class. After joining the data, each `ROAD_GEOM_FC1` data item is inner joined with the relevant `BASIC_HEIGHT_FC1` data item, and the joined data is returned as a dictionary where the actual type is `NSDictionary<NMAPlatformDataItem*, NMAPlatformDataItem*>`. Each key-value pair provides the necessary data for the feature: the key of `NMAPlatformDataItem` type representing `ROAD_GEOM_FC1` thematic layer data provides the road geometry such as coordinates of the road segment, and the value of `NMAPlatformDataItem` type representing `BASIC_HEIGHT_FC1` thematic layer data provides the average height of that road segment. The joined data provides everything needed to implement the intended feature.

Joined data using `LINK_ID` should look like the following:

```
{
    BRIDGE = N;
    LAT = "5246124,2";
    "LINK_ID" = 936938339;
    LON = "1342560,24";
    "LONG_HAUL" = Y;
    NAME = A100;
    NAMES = "GERBNTunnel BritzGERY\"tU|n@l \"brItsGERBNA100GERN\"?a: \"hUn|d6t;GERN\"?aU|to:|ba:n ?
aIn|\"hUn|d6t;GERY\"?a: ?aIn|\"hUn|d6t;GERN\"?aU|to:|ba:n \"hUn|d6tGERBNStadtring BerlinGERY\"Stat|
rIN bEr|\"li:n";
    TUNNEL = Y;
    ZLEVEL = ",";
} = {
```

```
    "DTM_AVG_HEIGHT" = 8500;
    "DTM_MAX_HEIGHT" = 8693;
    "DTM_MIN_HEIGHT" = 8099;
    "DTM_NONREF_ZCOORD" = 8500;
    "DTM_REF_ZCOORD" = 8500;
    "LINK_ID" = 936938339;
}
```

Note that the key-value pair has the same `LINK_ID`.

Chapter 4
Supplemental Information

Topics:

This section provides supplemental information for using HERE iOS SDK.

Logging HERE SDK Version

For troubleshooting purposes we recommend that you add HERE SDK version number in your application logs. You can get the SDK version by calling `[NMAApplicationContext sdkVersion]`.

Create a Simple HERE SDK App Using Swift

This tutorial provides instructions on how to create a simple application using *Swift programming language*. It is equivalent to the Objective-C tutorial, which is located at *Create a Simple App Using HERE SDK* on page 14.

Development tasks for this basic application include:

- Create a new Xcode project
- Add necessary resources and a map view to the project
- Acquire credentials from HERE for accessing map services
- Initialize the map view so that a map instance is created for rendering on the client device

The contents of this guide apply to Xcode 11.3 and the iOS 12 SDK.

Sample Project in HERE iOS SDK

A copy of the Xcode project described in this tutorial is available in `sample-apps` folder in your HERE iOS SDK package. To run the project, double-click on `SwiftHelloMap.xcodeproj` and follow the instructions in `README.txt` file.

Create a New Single View Application

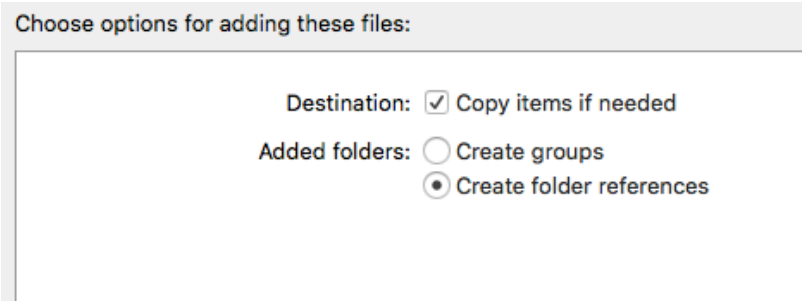
1. From Xcode menu, select **File** > **New** > **Project** to open the New project dialog (or press Shift + Command + N).
2. Select **iOS** > **Application** > **Single View Application** as the application type you want to create. Press Next.
3. In the next dialog, enter your **Product Name** (such as `HelloMap`) and **Organization Identifier** (such as `edu.self`).
4. Choose "Swift" under **Language**, then click **Next**.
5. Navigate to the directory where you want your project to be stored and then select **Create**.
6. The next step is to configure this project to use HERE SDK.

Configure the Application

1. Extract the HERE iOS SDK archive to somewhere in your local file system.
2. Add the *NMAKit* framework to your Xcode project. To add the *NMAKit* framework to your Xcode project, click on your app target and choose the "General" tab. Find the section called "Embedded Binaries", click the plus (+) sign, and then click the "Add Other" button. From the file dialog box, select the

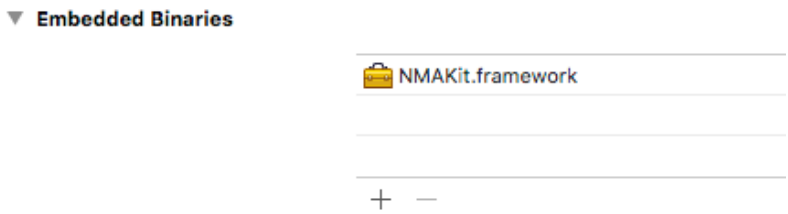
"NMAKit.framework" folder. Ensure that "Copy items if needed" and "Create folder reference" options are selected, then click **Finish**.

Figure 88: Add File to Target



3. Ensure that `NMAKit.framework` appears in "Embedded Binaries" and the "Linked Frameworks and Libraries" sections.

Figure 89: Embedded Binaries



4. Run the application. From the Xcode menu bar, select **Product** > **Run**. Ensure that the project runs in the iOS Simulator without errors.
5. HERE iOS SDK is now ready for use in your Xcode project. Now that you have your project configured to work with HERE SDK, try extending the sample application to render a map.

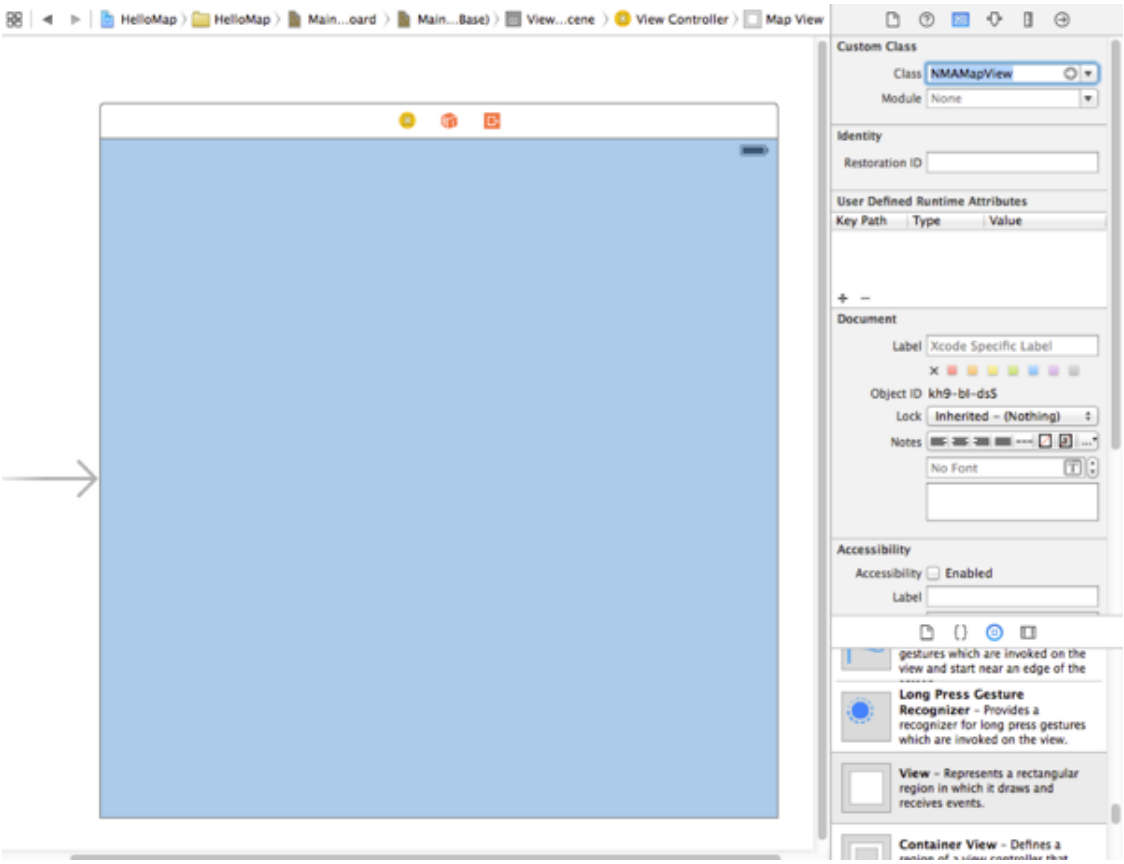
Create the Map View

In this section we utilize `NMAMapView` and `NMAGeoCoordinates` classes to render a Map.

1. Create an `NMAMapView`.
   a. Select `Main.storyboard` in the navigator, then open the Utilities view by pressing the key combination Command + Option + Control + 3. Drag and drop a View object from the Object Library onto the View Controller. If necessary, resize the View so it takes up the entire viewable area.
   b. In the Interface Builder click on the created View and then open the Identity Inspector in the Utilities view by pressing the key combination Command + Option + 3. Change the class value from `UIView`

to `NMAMapView` and press return. In the Document Outline you should see that the name of the View has changed from *View* to *Map View*.

Figure 90: MapView



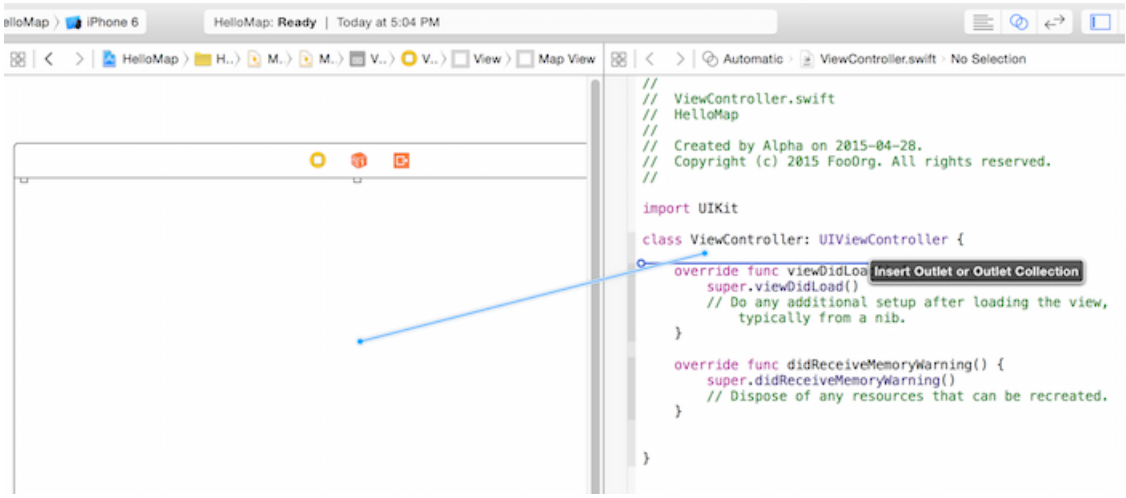
2. Create an outlet for `NMAMapView` in *ViewController*.
   a. Select `Main.storyboard` in the navigator.
   b. Press Command + Option + Return to open the Assistant Editor. It should show *ViewController.swift*.
   c. Add the following import statement to the top of this file:

```
import NMAKit
```

   d. Hold the Control key on the keyboard and click to drag from the Map View to the interface block in *ViewController.swift*. You should see a blue line and tooltip which says "Insert Outlet or Outlet Connection". Release the mouse button and a dialog appears allowing you to create an outlet.

e. Name the outlet *mapView*, keep the other default options and then select Connect.

Figure 91: Create an Outlet



3. Now an outlet to `NMAMapView` is set. The modified file should be as follows:

```
import UIKit
import NMAKit

class ViewController: UIViewController {

    @IBOutlet weak var mapView: NMAMapView!

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }

}
```

4. Implement `NMAMapView` setup and lifecycle code by replacing `viewDidLoad()` function with `viewWillAppear(animated)` and `addMapCircle()`:

```
override func viewWillAppear(_ animated: Bool) {
 super.viewWillAppear(animated)
 mapView.useHighResolutionMap = true
 mapView.zoomLevel = 13.2
 mapView.set(geoCenter: NMAGeoCoordinates(latitude: 49.258867, longitude: -123.008046),
  animation: .linear)
 // Note, logo is not shown when its size is greater than 1/8 of NMAMapView's height or width.
 mapView.copyrightLogoPosition = NMALayoutPosition.bottomCenter
 addMapCircle()
}

func addMapCircle() {
 if mapCircle == nil {
  let coordinates: NMAGeoCoordinates =
   NMAGeoCoordinates(latitude: 49.258867, longitude: -123.008046)
  mapCircle = NMAMapCircle(coordinates: coordinates, radius: 50)
```

```
  mapView.add(mapCircle!)
 }
}
```

5. Add your HERE application credentials.

   a. Open *AppDelegate.swift* file and import *NMAKit* by adding the following import statement to the top of the file.

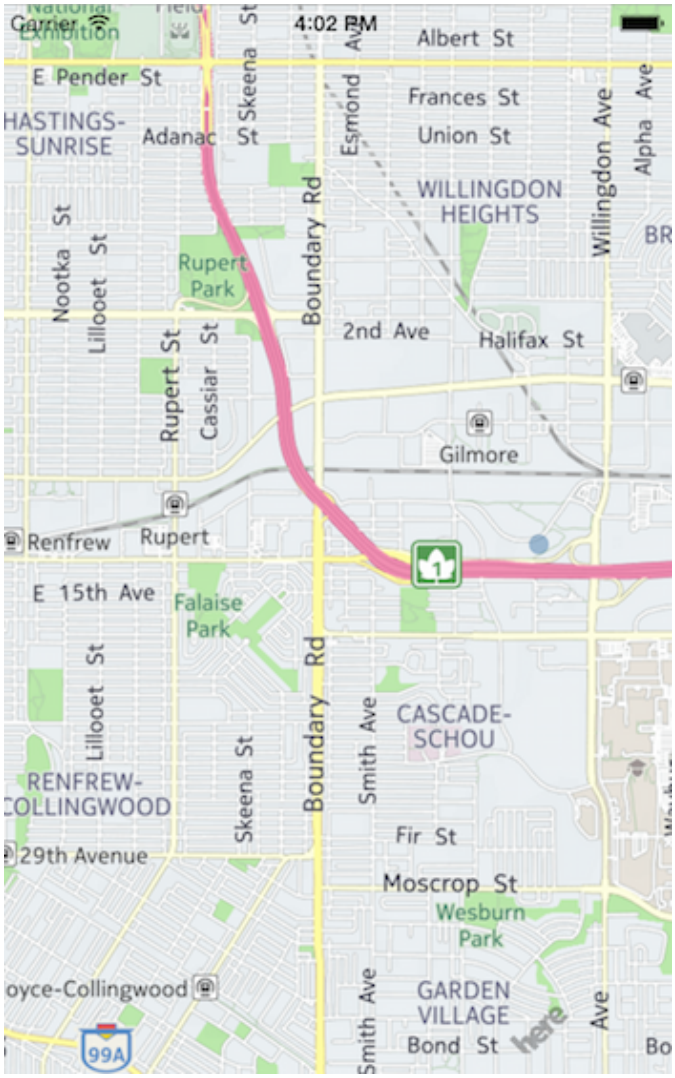
   ```
   import NMAKit
   ```

   b. Add the following in `didFinishLaunchingWithOptions` function replacing `YOUR_APP_ID`, `YOUR_APP_CODE` and `YOUR_LICENSE_KEY` with the credentials that you received from your *http://developer.here.com*.

   ```
   NMAApplicationContext.setAppId("YOUR_APP_ID",
           appCode: "YOUR_APP_CODE",
           licenseKey: "YOUR_LICENSE_KEY")
   ```

6. Build and run the application. If the build is successful, you now have an application that displays a map similar to the following screenshot and allows you to manipulate it using gestures.

Figure 92: Running the App



Swift Support in HERE SDK

This section provides information on the ways that HERE iOS SDK supports development using Swift 3 or above. The Swift 4.2 is recommended for optimal operation.

Nullability Annotations

Where appropriate, properties, method parameters, and return types are marked with nullability annotations. This provides more clarity on the generated Swift types and avoids implicitly unwrapped optionals.

For more information on this read the blog post on *"Nullability and Objective-C"*.

Designated Initializers

Where appropriate, class header files have annotation to indicate the designated initializer method.

For more information see the documentation on *initialization in Swift* and *initialization in Objective-C*.

Suggested Names for Swift

HERE SDK contains suggested Swift method names to better align with standard Swift naming conventions. For example, `createGeocodeRequestWithQuery:searchArea:locationContext:` is `create(query:searchArea:locationContext:)` in Swift.

Attention: An API Reference containing Swift method names is not currently available. You can find Swift method names in header files.

For more information see the blog post on *Swift API Design Guidelines*.

Stronger Typing Using Lightweight Generics

Where appropriate, HERE SDK contains lightweight generics information for collection types. This provides stronger typing in the generated Swift code. For example, `referenceIdentifiers(forSource:)` method in `NMAPlace` returns `[String]` rather than an `[AnyObject]`.

For more information see *Importing Objective-C Lightweight Generics*.

Supported Thread Usage

Developers should be aware of the following threading guidelines while using HERE iOS SDK:

- All `NMA` interface methods are designed to be called from the main thread.
- All `NMA` protocol callbacks and blocks are dispatched to the main thread.
- Any exceptions to this rule are clearly documented in the applicable protocol or block definition.

Network Access

In addition to the user's device connectivity setting you can also force HERE SDK to be offline by using `setNetworkAccessAllowed:` method in `NMAApplicationContext`. You can then check for this setting by calling `isNetworkAccessAllowed` method.

`NMAApplicationContext` class also provides `isOnline` method so that you can conveniently check if network access is allowed and HERE SDK is currently online. You can watch for changes to this value by subscribing to the following notification:

```
NMAApplicationContextOnlineStatusDidChangeNotification
```

3D Venues FAQ

Can I customize the venue color scheme and venue icon?

Yes. POI attributes can be customized using `NMAVenue3dStyleSettings` object.

There are two types of icons:

- Icons displayed on the map for a space, point-space, or facility (SVG file format)
- Icons displayed in native controls such as a search filter list (PNG file format)

Map icons can be customized in two ways, either by replacing the respective SVG asset files with customized ones, or by dynamically replacing it at runtime using `NMAVenue3dStyleSettings` object. See the following code snippet as an example on how to customize icon for a space.

```
if ([venue.uniqueId isEqual:@"DM_12468"]) {
  NSString *placeId = @"Lv16133Ds1286559";
  NMAVenue3dSpace* space = [venue spaceWithId:@"Lv26051Ds3186063"];
  if (space != nil) {
   NMAVenue3dController* controller = venueMapLayer.venueController;
   NMAVenue3dStyleSettings* newStyleSet = [[NMAVenue3dStyleSettings alloc] init];
   NSData* imageData = [[NSData alloc]
         initWithContentsOfURL:[NSURL URLWithString:@"https://placeholdit.imgix.net/"
              @"~text?txtsize=2&txt=T&w=5&h=5]];
   newStyleSet.labelImage = [NMAImage imageWithUIImage:[UIImage imageWithData:imageData]];
   [controller setStyleSettings:newStyleSet forSpace:space];
  }
 }
```

Note: Using a private HERE account, PNG icons that are displayed in native controls can also be customized by replacing the respective PNG icon files with customized ones. Note that there are 4 different sizes for each iOS resolution type. To make these customizations, see *Service Support* on page 11 to contact us for more details .

Can logos be added for individual POIs, such as for a restaurant?

Yes. You can modify the icon for an individual *space* dynamically using `NMAVenue3dStyleSettings` object. For details see `NMAVenue3dStyleSettings` API Reference.

Is it possible to expose venue data, in a classified manner, for a single user?

Yes, you can create a private HERE account and expose a "customized" version of the venue to it. See *Service Support* on page 11 to contact us for more details.

What fonts are supported by the Venues feature?

There is currently no support for fonts. Only the default font is supported.

Are there venue routing penalties for different types of connectors, such as elevators?

Penalties are applied when the route is calculated in "fastest" mode. For "shortest" mode connectors are considered only if the geometry has a significant distance.

How can I create a colored `NMAMapMarker`?

See the following code snippet or `NMAMapMarker` API Reference.

```
NMAMapMarker *_marker;

- (void)venueMapLayer:(NMAVenue3dMapLayer *)venueMapLayer
  didSelectSpace:(NMAVenue3dSpace *)space
   inVenue:(NMAVenue3dVenue *)venue
{
 [self removeMarker];
 UIImage *image = [UIImage imageNamed:@"markerIcon"];
 NMAImage *icon = [NMAImage imageWithUIImage:image];
 _marker = [NMAMapMarker mapMarkerWithGeoCoordinates:space.geoCenter icon:icon];
 [_marker setAnchorOffsetUsingLayoutPosition:NMALayoutPositionBottomCenter];
 _marker.mapLayerType = NMAMapLayerTypeForeground;
 _marker.zIndex = 100;
 [self.mainVC.activeMapView addMapObject:_marker];
}

- (void)venueMapLayer:(NMAVenue3dMapLayer *)venueMapLayer
  didDeselectSpace:(NMAVenue3dSpace *)space
    inVenue:(NMAVenue3dVenue *)venue
{
 [self removeMarker];
}

- (void)removeMarker
{
 if (_marker != nullptr) {
  [self.mainVC.activeMapView removeMapObject:_marker];
  _marker = nullptr;
 }
}
```

How do I select point-spaces (spaces without geometry) and calculate a route to such a position?

Take the tapped position and calculate the route from those coordinates. The following is an example of how to handle the map tap events to get a location for a route.

```
- (void)mapView:(NMAMapView *)mapView didReceiveTapAtLocation:(CGPoint)tapLocation
{
 NMAVenue3dController* controller = _venueMapLayer.venueController;
 NMAVenue3dBaseLocation* location;
```

```
 if (controller != nil) {
  location = [controller getLocationAtX:tapLocation.x Y:tapLocation.y WithSpacePrefered:YES];
  [self addToRoute:location];
 }
 else {
  NMAGeoCoordinates *coords = [mapView geoCoordinatesFromPoint:tapLocation];
  location = [NMAVenue3dOutdoorLocation outdoorLocationWithGeoCoordinates:coords];
 }
 [self addRoutePoint:location];
}

- (void)addRoutePoint:(NMAVenue3dBaseLocation *)location
{
 // Logic for saving route points
}
```

How do I calculate the length of a route?

1. From `NMAVenue3dCombinedRoute` result object, get all the route sections. Route sections can be of three types: `NMAVenue3dRouteSectionTypeVenue`, `NMAVenue3dRouteSectionTypeLink`, or `NMAVenue3dRouteSectionTypeOutdoor`.
2. For each route type you need to calculate its distance individually. See the following example on how to calculate length of different route types.

```
- (void)calculateDistanceFor:(NMAVenue3dCombinedRoute*) combinedRoute
{
 double distance = 0.0;
 NSArray* routeSections = [combinedRoute routeSections];
 for (NMAVenue3dRouteSection* section : routeSections) {
  if ([section isKindOfClass:[NMAVenue3dVenueRouteSection class]]) {
 // get length of indoor sections
   NSArray* maneuvers = ((NMAVenue3dVenueRouteSection*)section).routeManeuvers;
   NMAVenue3dRouteManeuver* maneuver = maneuvers.lastObject;
   distance += (double)maneuver.distanceFromStart;
  }
  else if ([section isKindOfClass:[NMAVenue3dLinkRouteSection class]]) {
 // get length of link sections
   NMAGeoCoordinates* from = ((NMAVenue3dLinkRouteSection*)section).from;
   NMAGeoCoordinates* to = ((NMAVenue3dLinkRouteSection*)section).to;
   distance += [from distanceTo:to];
  }
  else if ([section isKindOfClass:[NMAVenue3dOutdoorRouteSection class]]) {
 // get length of outdoor sections
   NMARoute* route = ((NMAVenue3dOutdoorRouteSection*)section).route;
   distance += (double)route.length;
   NSLog(@"--> +distance (outdoor): %f", distance);
  }
 }
}
```

How can my app automatically open a venue right after the app starts?

At app start make sure `NMAVenue3dService` is initialized and then select the desired venue using `NMAVenue3dMapLayer` as shown in the following example:

```
//...
 // on applicaton startup register the app to receive venue service events
 [service addListener:self];
//...
```

```
- (void)venueServiceDidInitialize:(NMAVenue3dService *)venueService withResult:
(NMAVenue3dServiceInitializationStatus)result
{

 if (result == NMAVenue3dServiceInitializationStatusOnlineSuccess ||
  result == NMAVenue3dServiceInitializationStatusOfflineSuccess) {
  // if the venue service started ok, select the venue
  [_venueMapLayer selectVenueWithVenueId:@"DM_7171"];
}
//...
```

How can my app detect whether a route passes a specific type of space, such as a security gate?

To identify if a route passes a certain type or category of space, loop through the route sections and inspect if a maneuver contains a space in the expected category. For example, the following code snippet shows how to find the first venue maneuver with category 4, which represents a security gate.

```
- (BOOL)category:(NSString*)category isInvolvedInRoute:(NMAVenue3dCombinedRoute*)route
{
 NSArray* routeSections = [route routeSections];
 for (NMAVenue3dRouteSection* section : routeSections) {
  if ([section isKindOfClass:[NMAVenue3dVenueRouteSection class]]) {
   NSArray* maneuvers = ((NMAVenue3dVenueRouteSection*)section).routeManeuvers;
   for (NMAVenue3dRouteManeuver* maneuver : maneuvers) {
    if (maneuver.space != nil && maneuver.space.content != nil) {
     NMAVenue3dContent* content = maneuver.space.content;
     if ([content.placeCategoryId isEqual:category]) {
      return YES;
     }
    }
   }
  }
 }
 return NO;
}
```

We get the spaces and facilities by iterating through all the levels and accessing `NMAVenue3dLevel.spacesAndFacilities` property. Is it possible to also get the entrances?

This is the correct way to get a list with all the spaces and facilities. However, entrances (accessors) are not exposed as POIs at the moment.

Is there a way to only show the venue model without showing the map tiles beneath it?

At the moment a venue can only be shown on top of the map.

Is there a way to integrate 3D Venues with Apache Cordova?

We do not support a Cordova plug-in. Only native Android and iOS SDKs are supported.

How can I get a nearby POI based on a space and a radius?

This is not currently supported.

How can I get a nearby POI based on an indoor routing line?

This is not possible at the moment. This is planned for a future release.

How can I get a certain area around a venue model for offline mode?

Once a venue has been downloaded, it is cached and will be available later in offline mode. However, you need to be online and request the venue prior to this.

Can venue entrance markers be customized?

HERE SDK already marks entrances on the map and on the route with entrance icons. For private venues you can customize these icons. See *Service Support* on page 11 to contact us for more details.

How do I pre-cache individual venues without user interaction?

You need to download venues using `NMAVenue3dService`.

- `venueWithId:`
- `venuesAtGeoCoordinates:`
- `venuesInGeoBoundingBox:`

For more information see `NMAVenue3dService` API Reference.

How do I translate geo locations into indoor locations, expressed as latitude, longitude, and level, for a given functional call within HERE SDK?

You need to have the geo location which you want to map inside the venue, as well as the level on which you intend to place those coordinates. Thus, you can use `NMAVenue3dLevelLocation` to construct the indoor location.

```
- (void)mapView:(NMAMapView *)mapView didReceiveTapAtLocation:(CGPoint)tapLocation
{
 // convert tap point to NMAGeoGoordinate
 NMAGeoCoordinates *coords = [mapView geoCoordinatesFromPoint:tapLocation];

 // If any venue is selected, get related venue controller
 NMAVenue3dController* venueController = _venueMapLayer.venueController;

 // If no venue was selected, consider tapped location as outdoor location
 // and add it as route point
 if (venueController == nil) {
  NMAVenue3dOutdoorLocation *loc = [NMAVenue3dOutdoorLocation
 outdoorLocationWithGeoCoordinates:coords];
 [self addToRoute:loc];
 return;
 }

  // Otherwise find location inside a venue and add it as route point
 else {
  NMAVenue3dBaseLocation* loc = [venueController getLocationAtX:tapLocation.x
                  Y:tapLocation.y
              WithSpacePrefered:true];
  if ([loc isValid] && ![loc isKindOfClass:[NMAVenue3dOutdoorLocation class]]) {
   [self addToRoute:loc];
  }
 }
```

```
}
-(void)addToRoute:(NMAVenue3dBaseLocation*) baseLocation
{
 // do something with the route
}
```

Calculating Route Length

The following code example shows how the total length of a route can be calculated:

```
- (void)didCalculateRoute:(NMAVenue3dCombinedRoute *)combinedRoute
{
 double distance = 0.0;
 NSArray *routeSections = combinedRoute.routeSections;

 for (id routeSection in routeSections) {
  if ([routeSection class] == [NMAVenue3dVenueRouteSection class]) {
   NMAVenue3dVenueRouteSection *section = (NMAVenue3dVenueRouteSection *)routeSection;
   NSArray *maneuvers = section.routeManeuvers;
   NMAVenue3dRouteManeuver *lastManeuver = [maneuvers lastObject];
   distance += [lastManeuver distanceFromStart];
  } else if ([routeSection class] == [NMAVenue3dLinkRouteSection class]) {
   NMAVenue3dLinkRouteSection *section = (NMAVenue3dLinkRouteSection *)routeSection;
   NMAGeoCoordinates *start = section.from;
   NMAGeoCoordinates *destination = section.to;
   distance += [start distanceTo:destination];
  } else if ([routeSection class] == [NMAVenue3dOutdoorRouteSection class]) {
   NMAVenue3dOutdoorRouteSection *section = (NMAVenue3dOutdoorRouteSection *)routeSection;
   NMARoute *route = section.route;
   distance += route.length;
  }
 }
 // do something with distance information
}
```

Size Management

This section provides tips on reducing the size of HERE SDK so your application uses less storage on consumer devices.

Remove Unused Font Files

By default, HERE iOS SDK includes a number of font files to support different languages. These files may range from a few hundred kilobytes to a few megabytes in size. You can remove unused font files to reduce the size of your HERE SDK-enabled application.

To remove HERE SDK font files, perform these steps:

1. In Xcode, right-click on **NMABundle.bundle** and choose **Show in Finder**.
2. In the new Finder window, right-click on **NMABundle.bundle** and choose **Show Package Contents.**
3. Navigate to the **optional-font** directory.
4. Delete unused font files. The removable fonts in this directory include:
    - **NanumGothic_Regular.ttf** - Korean

- **LohitIndic.ttf** and **LohitKanMalTamTel.ttf** - Indic
- **HanSans_ExtraLight.ttf** - Chinese

After completing these steps your application uses less space when it is installed on a device. The maximum amount that you can save with these steps is approximately 5MB.

Signpost Parsing

This section provides guidelines on how to use HERE Mobile SDK Signpost object to present visual maneuver instructions which match the audible voice instructions during turn-by-turn guidance. While the high-level guidelines in this section are applicable to both HERE Android and iOS SDKs, the detailed descriptions are generic and do refer to the specific APIs available on either platform.

Signs represent textual and graphic information posted along roads. The information is always represented as text, but may originate from a graphical icon. Signpost information may be used for route guidance (both audible and visual guidance) and map display. A navigation system may prefer using the signpost text rather than the street/ramp name as the latter may not always match what is on the sign in reality and may confuse a user. The signpost feature supports the user navigating through complex situations and provides a conformation for a maneuver by presenting the same direction information as shown on the street signs in reality.

Signpost Content

The Signpost representation is an association between two links, the "From link" and the "To link". They are not necessarily contiguous. The "From link" is the link prior to a maneuver decision point. A maneuver decision is a point where the road splits and the driver is unable to determine the direction to traverse. The "To link" is the link traversed after the decision point.

Note: Not all maneuvers contain Signpost information.

Based on the map attributes, HERE Mobile SDKs aggregate the following information into a Signpost object:

- Exit number
- Exit text
- Exit icon
- Exit directions
- Foreground / Background color

Exit directions is an array of information shown on the sign and can contain the following information:

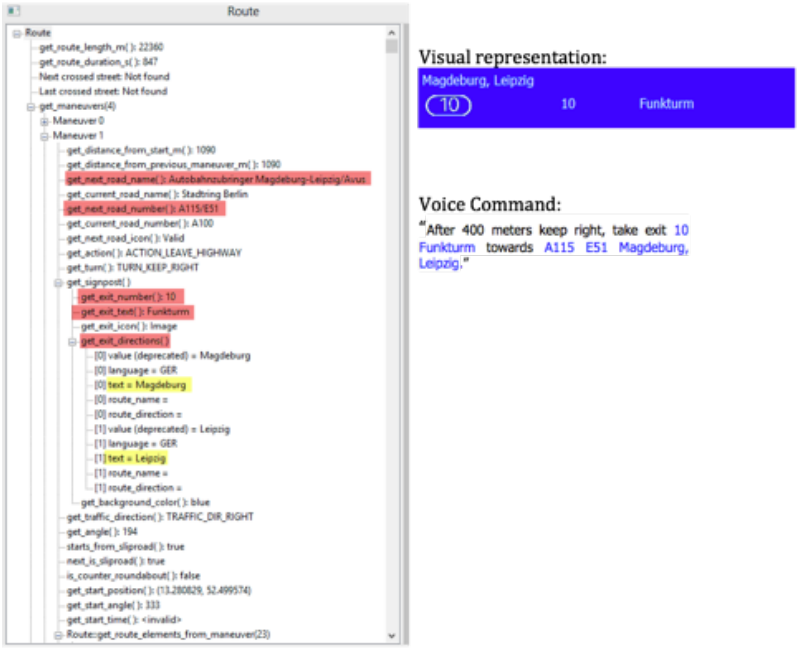
- Language i.e. the three letter MARC code for the label language
- Route direction
- Route name
- Text representing auxiliary information. Typically the auxiliary information contains a destination name such as a city or a characteristic place (such as "airport")

Each index in the exit directions array will typically contain only one of Route direction, Route name and Text. Since the origin data is not always consistent, it is advisable to concatenate the route name and route direction fields.

Parsing Basic Signpost Information

To present an audible or visual maneuver instruction, information from the Maneuver and Signpost objects can be combined as shown in the figure below:

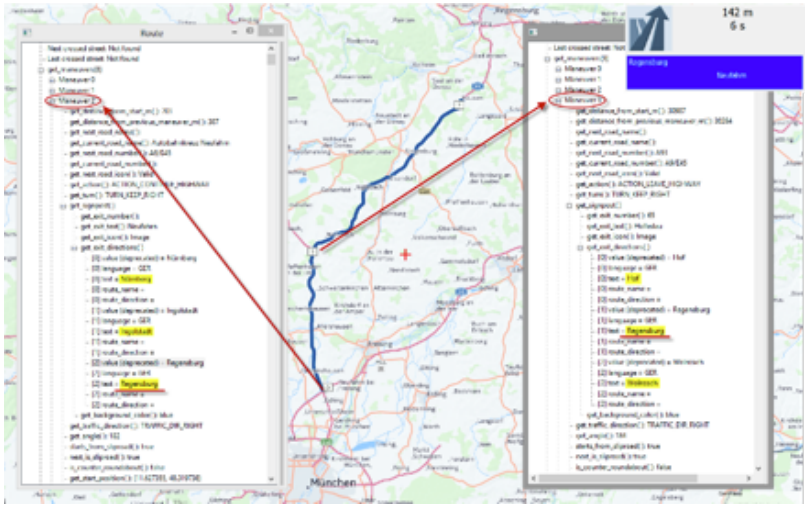
Figure 93: Parsing for audible instructions for a basic sign



Not all exit direction information presented on the real signpost might be relevant to the user's route. To avoid a flood of information, which easily could lead to confusion, especially for the voice output, exit directions information which is irrelevant for a driver's route may be suppressed.

To achieve this, the signpost information of the next maneuver and the next-next maneuver can be compared and only the exit direction information valid for both maneuvers may be presented for the next maneuver. An example of this is shown in the figure below.

Figure 94: Filtering out exit directions information



There may be some cases where there is no matching exit direction information in the next maneuver and the next-next maneuver. In this case the recommendation is to present the first index in the exit directions array of the next maneuver.

Chapter 5
Coverage Information

Topics:

The following list provides coverage information for HERE iOS SDK features. Feature support in HERE SDK may differ depending on the language and locale.

- *Satellite Imagery*
- *Public Transit*
- *Traffic*
- *Routing*
- *Guidance*
- *Point Addresses* (such as house numbers)
- *Online Geocoding / Reverse Geocoding*
- *Online Places and Search*

Downloadable Maps by Country/Region

Downloadable maps are available for the following continental regions:

- North and Central America
- South America
- Europe
- Africa
- Asia
- Australia/Oceania

Downloadable maps are available for the following countries:

- Albania
- Algeria
- Andorra
- Angola
- Argentina
- Armenia
- Aruba
- Australia (region download possible)
  - Australian Capital Territory
  - New South Wales
  - Northern Territory
  - Queensland
  - South Australia
  - Tasmania
  - Victoria
  - Western Australia
- Austria
- Azerbaijan
- Bahamas
- Bahrain
- Bangladesh
- Belarus
- Belgium
- Belize
- Benin
- Bermuda
- Bolivia
- Bosnia and Herzegovina
- Botswana
- Brazil (region download possible)
  - Norte do Brasil
  - Nordeste do Brasil

    - Centro-Oeste do Brasil
    - Sudeste do Brasil
    - Sul do Brasil
    - Acre
    - Alagoas
    - Amapá
    - Amazonas
    - Bahia
    - Ceará
    - Distrito Federal (Federal District)
    - Espírito Santo
    - Goiás
    - Maranhão
    - Mato Grosso
    - Mato Grosso do Sul
    - Minas Gerais
    - Pará
    - Paraíba
    - Paraná
    - Pernambuco
    - Piauí
    - Rio de Janeiro
    - Rio Grande do Norte
    - Rio Grande do Sul
    - Rondônia
    - Roraima
    - Santa Catarina
    - São Paulo
    - Sergipe
    - Tocantins
- Brunei
- Bulgaria
- Burkina Faso
- Burundi
- Cambodia
- Cameroon
- Canada (region download possible)
    - Alberta
    - British Columbia
    - Manitoba
    - New Brunswick
    - Newfoundland and Labrador
    - Nova Scotia
    - Ontario
    - Prince Edward Island
    - Quebec

    - Saskatchewan
    - Northwest Territories
    - Nunavut
    - Yukon
- Cape Verde
- Cayman Islands
- Central African Republic
- Chad
- Chile
- Colombia
- Comoros
- Congo (Republic)
- Congo (Zaire)
- Costa Rica
- Cote d'Ivoire
- Croatia
- Cuba
- Cyprus
- Czech Republic
- Denmark
- Djibouti
- Dominican Republic
- Ecuador
- Egypt
- El Salvador
- Equatorial Guinea
- Eritrea
- Estonia
- Ethiopia
- Fiji
- Finland
- France (region download possible)
    - Auvergne-Rhone-Alpes
    - Brittany
    - Burgundy-Franche-Comte
    - Centre-Val de Loire
    - Corsica
    - Grand Est
    - Hauts-de-France
    - Île-de-France
    - Loire Region
    - Normandy
    - Nouvelle-Aquitaine
    - Occitanie
    - Provence-Alpes-Cote D'Azur
    - Reunion

    - Guadeloupe
    - Martinique
    - French Guiana
    - Saint Barthelemy
- Gabon
- Gambia
- Georgia
- Germany (region download possible)
    - Baden-Wuerttemberg
    - Bavaria
    - Berlin/Brandenburg
    - Hesse
    - Mecklenburg-Western Pomerania
    - Lower Saxony/Bremen
    - North Rhine-Westphalia
    - Rhineland-Palatinate/Saarland
    - Saxony-Anhalt
    - Saxony
    - Schleswig-Holstein/Hamburg
    - Thuringia
- Ghana
- Gibraltar
- Greece
- Guatemala
- Guernsey
- Guinea
- Guinea-Bissau
- Guyana
- Honduras
- Hong Kong and Macau
- Hungary
- Iceland
- India (region download possible)
    - Andhra Pradesh
    - Bihar and Jharkhand
    - Gujarat
    - Haryana
    - Himachal Pradesh
    - Jammu and Kashmir
    - Karnataka
    - Kerala
    - Madhya Pradesh and Chhattisgarh
    - Maharashtra and Goa
    - Northeast
    - Orissa
    - Punjab

    - Rajasthan
    - Tamil Nadu
    - Uttar Pradesh and Uttarakhand
    - West Bengal and Sikkim
    - Delhi
    - Telangana
- Indonesia (region download possible)
    - Bali and Nusa Tenggara Islands
    - Java Island
    - Kalimantan Island
    - Maluku Islands
    - Papua Island
    - Sulawesi Island
    - Sumatra Island
- Iraq
- Israel
- Italy (region download possible)
    - Piemonte
    - Valle D'Aosta
    - Lombardy
    - Trentino-Alto Adige
    - Veneto
    - Friuli-Venezia Giulia
    - Liguria
    - Emilia-Romagna
    - Toscana
    - Umbria
    - Marche
    - Lazio
    - Abruzzo
    - Molise
    - Campania
    - Puglia
    - Basilicata
    - Calabria
    - Sicilia
    - Sardegna
- Jamaica
- Jordan
- Kazakhstan
- Kenya
- Kuwait
- Latvia
- Lebanon
- Lesotho
- Liberia

- Libya
- Liechtenstein
- Lithuania
- Luxembourg
- Madagascar
- Malawi
- Malaysia
  - Peninsular Malaysia
  - East Malaysia
- Maldives
- Mali
- Malta
- Mauritania
- Mauritius
- Mayotte
- Mexico
- Monaco
- Montenegro
- Morocco
- Mozambique
- Namibia
- Nepal
- Netherlands
- New Zealand
- Nicaragua
- Niger
- Nigeria
- Norway
- Oman
- Panama
- Paraguay
- Peru
- Philippines
  - Luzon
  - Visayas
  - Mindanao
- Poland
- Portugal
- Qatar
- Republic of Ireland
- Republic of Moldova
- Romania
- Russia (region download possible)
  - Northwestern Federal District
  - Central Federal District
  - Southern Federal District

    - North Caucasian Federal District
    - Volga Federal District
    - Urals Federal District
    - Siberian Federal District
    - Far Eastern Federal District
- Rwanda
- Saint Helena
- Saint Kitts and Nevis
- Saint Vincent and the Grenadines
- San Marino
- Sao Tome and Principe
- Saudi Arabia
- Senegal
- Serbia
- Seychelles
- Sierra Leone
- Singapore
- Slovakia
- Slovenia
- Somalia
- South Africa
- South Korea
- Spain (region download possible)
    - Galicia
    - Asturias
    - Cantabria
    - Basque Country
    - Navarre
    - Castile and Leon
    - La Rioja
    - Aragon
    - Catalonia
    - Madrid
    - Extremadura
    - Castilla la Mancha
    - Valencian Community
    - Balearic Islands
    - Andalucia
    - Murcia
    - Canary Islands
    - Ceuta
    - Melilla
- Sri Lanka
- Suriname
- Swaziland
- Sweden

- Switzerland
- Taiwan
- Tanzania
- Thailand
  - Central Thailand
  - Northern Thailand
  - Northeast Thailand
  - Southern Thailand
- The Former Yugoslav Republic of Macedonia
- Togo
- Trinidad and Tobago
- Tunisia
- Turkey
- USA (region download possible)
  - Alabama
  - Alaska
  - Arkansas
  - Arizona
  - California
  - Colorado
  - Connecticut
  - Delaware
  - Florida
  - Georgia
  - Hawaii
  - Idaho
  - Illinois
  - Indiana
  - Iowa
  - Kansas
  - Kentucky
  - Louisiana
  - Maine
  - Maryland
  - Massachusetts
  - Michigan
  - Minnesota
  - Mississippi
  - Missouri
  - Montana
  - Nebraska
  - Nevada
  - New Hampshire
  - New Jersey
  - New Mexico
  - New York

    - North Carolina
    - North Dakota
    - Ohio
    - Oklahoma
    - Oregon
    - Pennsylvania
    - Rhode Island
    - South Carolina
    - South Dakota
    - Tennessee
    - Texas
    - Utah
    - Vermont
    - Virginia
    - Washington
    - West Virginia
    - Wisconsin
    - Wyoming
    - Puerto Rico
    - US Virgin Islands
    - Washington D.C.
    - Guam
- Uganda
- Ukraine (region download possible)
    - Cherkasy Oblast
    - Chernihiv Oblast
    - Chernivtsi Oblast
    - Crimea
    - Dnipropetrovsk Oblast
    - Donetsk Oblast
    - Ivano-Frankivsk Oblast
    - Kharkiv Oblast
    - Kherson Oblast
    - Khmelnytskyi Oblast
    - Kirovohrad Oblast
    - Kyiv Oblast
    - Luhansk Oblast
    - Lviv Oblast
    - Mykolaiv Oblast
    - Odessa Oblast
    - Poltava Oblast
    - Rivne Oblast
    - Sumy Oblast
    - Ternopil Oblast
    - Vinnytsia Oblast
    - Volyn Oblast

    - Zakarpattia Oblast
    - Zaporizhia Oblast
    - Zhytomyr Oblast
- United Arab Emirates
- United Kingdom (region download possible)
    - England
    - Scotland
    - Wales
    - Northern Ireland
- Uruguay
- Vatican City
- Venezuela
- Vietnam
    - Northern Vietnam
    - Central Vietnam
    - Southern Vietnam
- Virgin Islands
- Yemen
- Zambia
- Zimbabwe

Map Label Languages

HERE iOS SDK is a globally available product with support for many languages. HERE SDK does not require any actions from developers to set the appropriate language as it automatically detects the current device language setting and applies the same language within the SDK if it is supported. If HERE SDK does not support the device language, a fallback language, which is typically English, is used.

Map labels at the street level are always displayed in the local language. For example, Central Park in Manhattan, New York is always displayed as "Central Park" irrespective of the device language setting. Names for states, provinces, regions, cities, mountains, lakes, and rivers, may be localized to the language corresponding to the device language setting. For example, where the device language is set to French, the label for South Carolina reads "Caroline du Sud"; if the device language is set to Spanish, the label for New York reads "Neuva York".

Language	Language code	Marc code mapping	Note
English (UK)	EN	ENG	
French	FR	FRE	
German	GE	GER	
Spanish	SP	SPA	
Italian	IT	ITA	
Swedish	SW	SWA	
Danish	DA	DAN	

Language	Language code	Marc code mapping	Note
Norwegian	NO	NOR	
Finnish	FI	FIN	
English American	AM	ENG	Mapped to ENG
Swiss French	SF	FRE	Mapped to FRE
Swiss German	SG	GSW	Mapped to GER
Portuguese	PO	POR	
Turkish	TU	TUR	
Icelandic	IC	ICE	
Russian	RU	RUS	
Hungarian	HU	HUN	
Dutch	DU	DUT	
Flemish Belgian	BL	DUT	Mapped to DUT
Australian	AU	ENG	Mapped to ENG
Belgian French	BF	FRE	Mapped to FRE
Austrian	AS	GER	Mapped to GER
New Zealand	NZ	ENG	Mapped to ENG
French International	IF	FRE	Mapped to FRE
Czech	CS	CZE	
Slovak	SK	SLO	
Polish	PL	POL	
Slovenian	SL	SLV	
Chinese Taiwan	TC	CHT	
Hong Kong Chinese	HK	CHT	
Chinese PRC	ZH	CHI	
Japanese	JA	JPN	
Thai	TH	THA	
Afrikaans	AF	AFR	
Arabic	AR	ARA	
Bulgarian	BG	BUL	
Catalan	CA	CAT	
Croatian	HR	SCR	
Canadian English	CE	ENG	Mapped to ENG
English International	IE	ENG	Mapped to ENG
South African English	SA	ENG	Mapped to ENG
Estonian	ET	EST	
French Canadian	CF	FRE	Mapped to FRE

Language	Language code	Marc code mapping	Note
Greek	EL	GRE	
Greek (Cyprus)	CG	GRE	Mapped to GRE
Hebrew	HE	HEB	
Hindi	HI	HIN	
Indonesian	IN	IND	
Swiss Italian	SZ	ITA	Mapped to ITA
Latvian	LV	LAV	
Lithuanian	LT	LIT	
Malay	MS	MAY	
Marathi	MR	MAR	
Norwegian Nynorsk	NN	NOR	
Brazilian Portuguese	BP	POR	Mapped to POR
Romanian	RO	RUM	
Serbian	SR	SRP	
International Spanish	OS	SPA	Mapped to SPA
Latin American Spanish	LS	SPA	Mapped to SPA
Finland Swedish	FS	SWE	Mapped to SWE
Cyprus Turkish	CT	TUR	Mapped to TUR
Ukrainian	UK	UKR	
Urdu	UR	URD	
Vietnamese	VI	VIE	
Basque	BA	BAQ	
Malay as appropriate for use in Asia-Pacific regions	MA	MAY	Mapped to ENG

Navigation Voices

HERE iOS SDK supports navigation voices in the following languages, in either text-to-speech or pre-recorded format.

- Pre-recorded voices support spoken maneuver instructions.
- Text-to-speech voices support spoken maneuver instructions and street names.

Text-to-Speech

- English (US)
- English (UK)
- French (France)
- French (Canada)
- German
- Spanish (Spain)
- Spanish (Mexico)
- Indonesian
- Italian
- Japanese
- Norwegian
- Portuguese (Portugal)
- Portuguese (Brazil)
- Russian
- Swedish
- Finnish
- Danish
- Korean
- Chinese (Taiwanese Mandarin)
- Turkish
- Czech
- Polish

Note: This is a list of the potential languages that are supported. Actual audio playback depends on the supported languages in the user's installed version of iOS.

Pre-recorded

- English (UK) - Female
- English (US) - Male
- English (UK) - Male
- Portuguese (Portugal) - Female
- Turkish - Female
- Icelandic - Female
- Russian - Female
- Hungarian - Female
- Hungarian - Male
- Dutch - Female
- Dutch - Male
- French - Female
- Czech - Female
- Slovak - Female
- Polish - Female
- Slovenian - Female
- Mandarin (Taiwan) - Female
- German - Female
- Cantonese - Female
- Mandarin (China) - Female
- German - Male
- Thai - Male
- Afrikaans - Female
- Arabic (North African) - Female
- Arabic (Saudi Arabia) - Male
- Tagalog - Female
- Spanish (Spain) - Female
- Basque - Female
- Galician - Female
- Spanish (Latin America) - Female
- Bulgarian - Female
- Catalan - Female
- Croatian - Female
- Estonian - Female
- Italian - Female
- Farsi - Female
- French (Canada) - Female
- Greek - Female
- Hindi - Female
- Indonesian - Male
- Swedish - Female
- Korean (South Korea) - Female
- Latvian - Female
- Lithuanian - Female
- Danish - Female
- Malay - Female
- Portuguese (Brazil) - Female
- Romanian - Female
- Serbian - Male
- Norwegian - Female
- Spanish (Mexico) - Male
- Swahili - Male
- Tamil - Male
- Finnish - Female
- Finnish - Male
- Ukrainian - Female
- Urdu - Female
- Vietnamese - Female

Safety Camera Coverage

The following table provides an overview of actual Safety Camera coverage.

Please note that currently only Safety Cameras of type 'Speed', 'RedLightAndSpeed', and 'RedLight' are included. Another inclusion criteria is that speed limit value and unit must not be null.

Safety Camera data is not available for Germany, Liechtenstein, Switzerland because of legal restrictions.

Safety Camera data is deliberately excluded for France as it is legally prohibited to include exact location of speed cameras in this country.

Table 7: Safety Camera Coverage by Country

Country	2018Q1
Andorra	10
Argentina	592
Australia	1397
Austria	948
Belgium	1801
Belarus	300
Bosnia and Herzegovina	52
Brazil	17926
Bulgaria	126
Canada	646
Chile	75
Croatia	66
Cyprus	2
Czech Republic	242
Estonia	63
Finland	817
Greece	286
Hungary	146
Iceland	17
Isle of Man	2
Italy	6868
Kazakhstan	963
Kuwait	454
Latvia	51
Lithuania	167

Country	2018Q1
Luxembourg	20
Malaysia	402
Mexico	327
Netherlands	1229
New Zealand	184
Norway	355
Poland	602
Portugal	83
Qatar	571
Romania	96
Russia	12155
Saudi Arabia	1362
Serbia	43
Singapore	221
Slovakia	16
Slovenia	70
South Africa	723
Spain	1377
Sweden	1596
Taiwan	2792
Thailand	198
United Arab Emirates	2063
United Kingdom	5242
United States	4734