HERE iOS SDK

Developer's Guide

Starter Edition Version 3.12

Important Information
Notices

Topics:

This section contains document notices.

Legal Notices

Document Information

Product	
Name:	HERE iOS SDK
Version:	Starter Edition Version 3.12

Document	
Name:	HERE iOS SDK Developer's Guide
ID:	ae2a53a0-72ea-409c-80f8-33f881a59310
Status:	FINAL
Date:	2019-Jun-25, 22:20 (GMT)

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.

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.

Feature List

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

Mapping:

- Raster map tiles with high resolution support
- Map styles: Normal, Satellite, Terrain, and more
- Touch gestures such as tap, pan, and pinch
- Overlay objects on the map such as polylines, polygons, icons, routes

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

Directions:

- Online Car and Pedestrian Route Directions
- Specify preferred route type (fastest or shortest) and route options (such as avoiding toll roads, motorways, and parks)
- Alternate routes

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 (`App_Id` and `App_Code` – see also *Authenticating Applications* on page 20) 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.

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 11.

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 Starter 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 10 and the iOS 12 SDK. For more details, see *System Requirements* on page 20.

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 HEREMapsStarter
```

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 12.

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`

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 10 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 55.

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`).
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

The 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, '10.0'
target 'YourApp' do
 pod 'HEREMapsStarter'
end
```

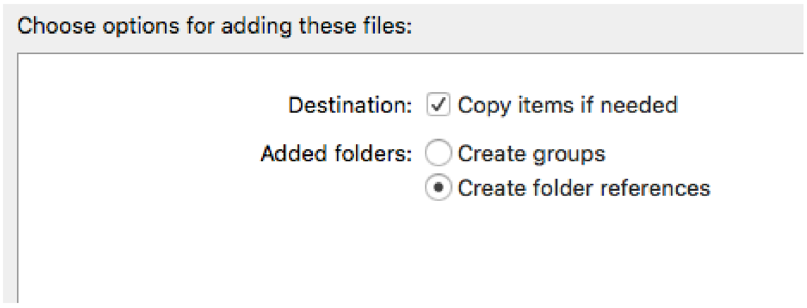
2. Next, open a command-line terminal on your workstation. From your project root folder run this command:

```
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



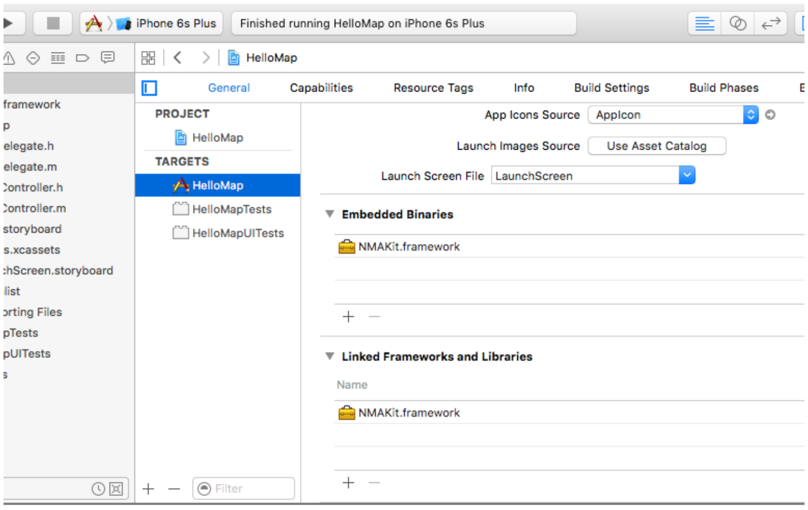
Choose options for adding these files:

Destination: ✓ Copy items if needed

Added folders: Create groups [✓ Create folder references]

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

Figure 2: Added Embedded Binaries



App Icons Source: AppIcon

Launch Images Source: Use Asset Catalog

Launch Screen File: LaunchScreen

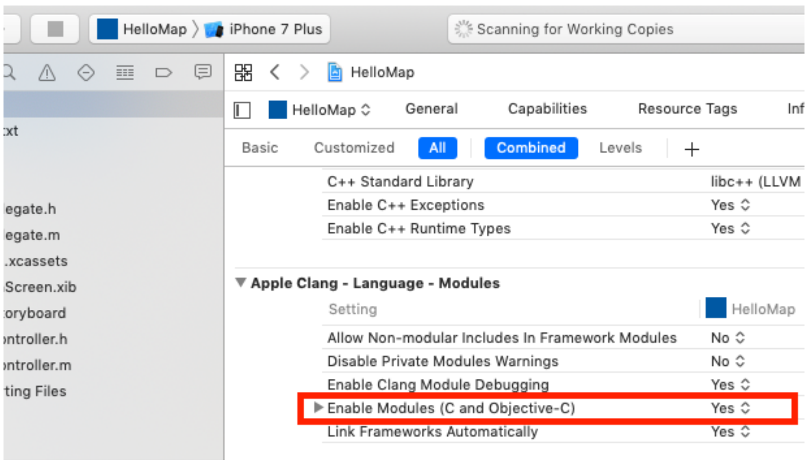
Embedded Binaries: NMAKit.framework

Linked Frameworks and Libraries: Name: NMAKit.framework

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



C++ Standard Library	libc++ (LLVM
Enable C++ Exceptions	Yes
Enable C++ Runtime Types	Yes

▼ Apple Clang - Language - Modules

Setting	HelloMap
Allow Non-modular Includes In Framework Modules	No
Disable Private Modules Warnings	No
Enable Clang Module Debugging	Yes
▶ Enable Modules (C and Objective-C)	Yes
Link Frameworks Automatically	Yes

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.

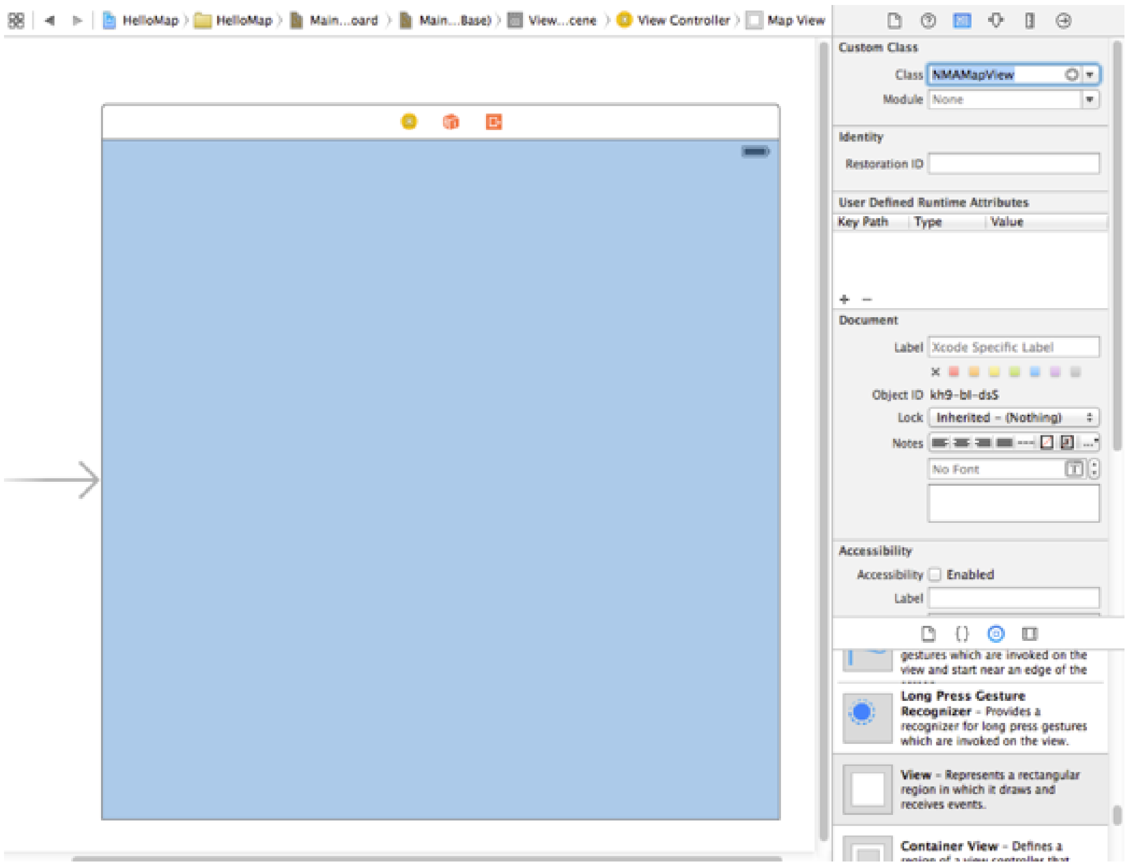
Create the Map View

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



Custom Class
Class NMAMapView
Module None

Identity
Restoration ID

User Defined Runtime Attributes

Key Path	Type	Value

Document
Label Xcode Specific Label
Object ID kh9-bl-dsS
Lock Inherited - (Nothing)
Notes
No Font

Accessibility
Accessibility Enabled
Label

gestures which are invoked on the view and start near an edge of the

Long Press Gesture Recognizer - Provides a recognizer for long press gestures which are invoked on the view.

View - Represents a rectangular region in which it draws and receives events.

Container View - Defines a

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*.

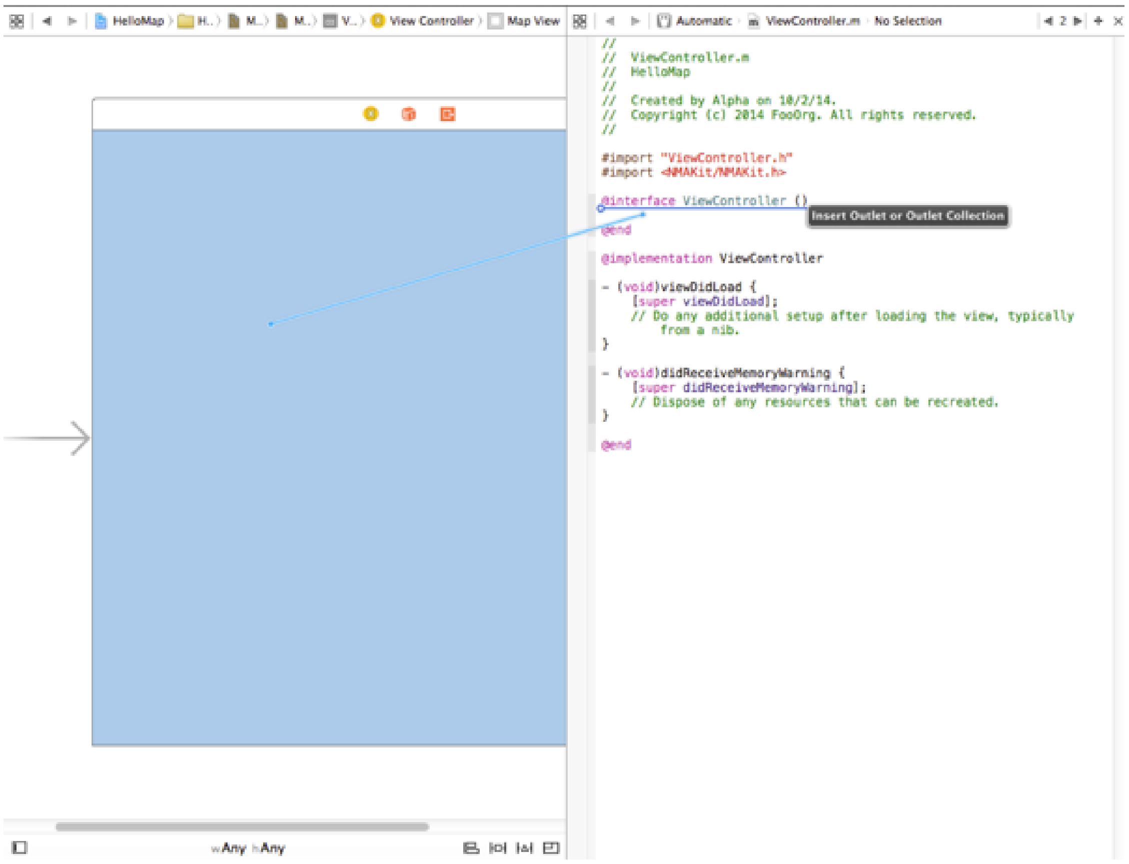
   c. Add the following import statement to the top of this file:

   ```
   @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



```
//
//  ViewController.m
//  HelloMap
//
//  Created by Alpha on 10/2/14.
//  Copyright (c) 2014 FooOrg. All rights reserved.
//

#import "ViewController.h"
#import <NMAKit/NMAKit.h>

@interface ViewController ()

@end

@implementation ViewController

- (void)viewDidLoad {
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically
        from a nib.
}

- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}

@end
```

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];
  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.

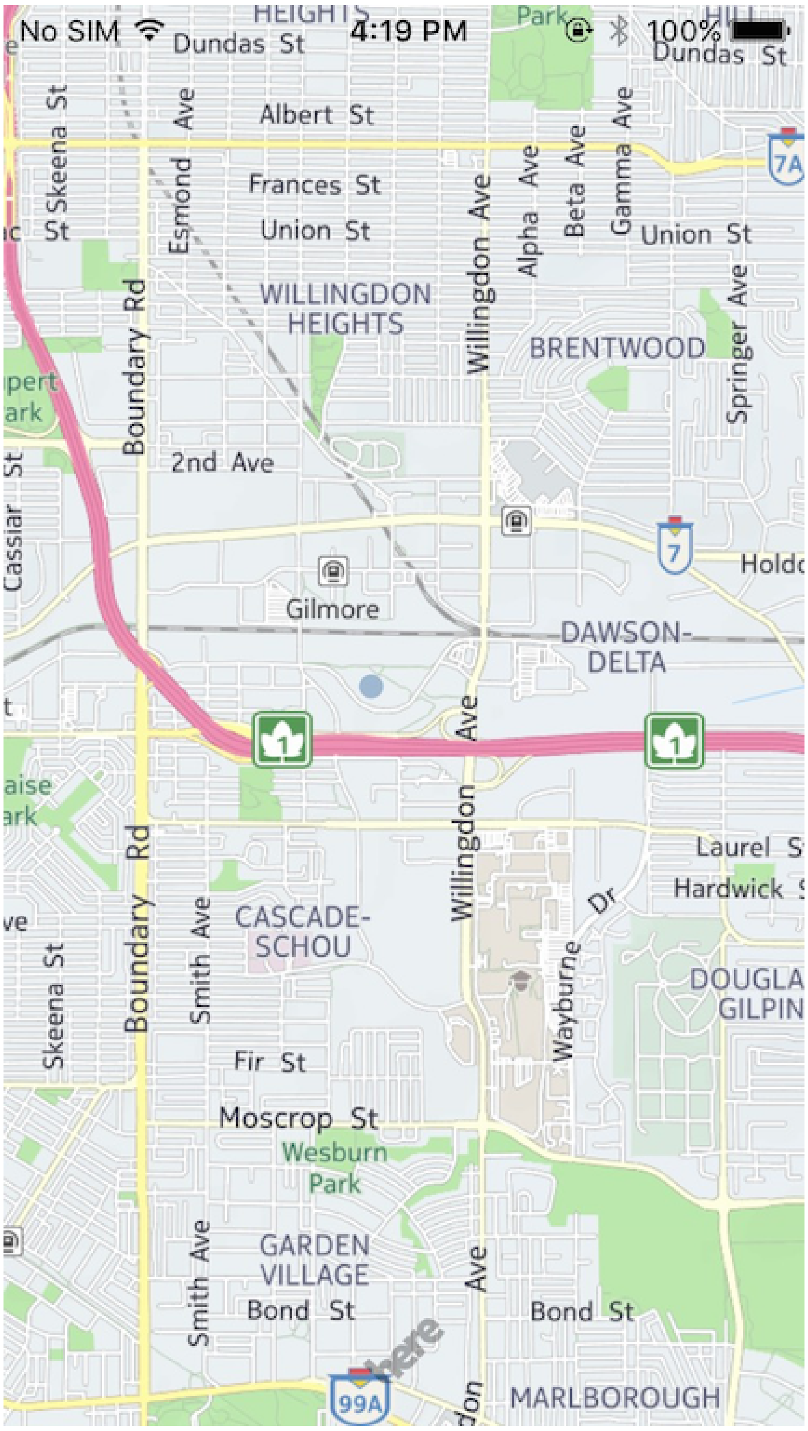
      ```
      @import NMAKit;
      ```

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

      ```
      [NMAApplicationContext setAppId:@"{YOUR_APP_ID}"
            appCode:@"{YOUR_APP_CODE}"];
      ```

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 11 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 10 or above running on macOS 10.13.6 or above.
- A minimum of 10MB per application should be made available for the storage of HERE SDK libraries.
- A minimum of 32MB should be made available for the storage of map data.
- Data connectivity (Wi-Fi or Cellular) is required to download map data.

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` and `App_Code`) 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. Do not reuse credentials across multiple applications.

Adding Credentials

Ensure that you have provided the `app_id` and `app_code` 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}"];

 return YES;
}
```

Examples on GitHub

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

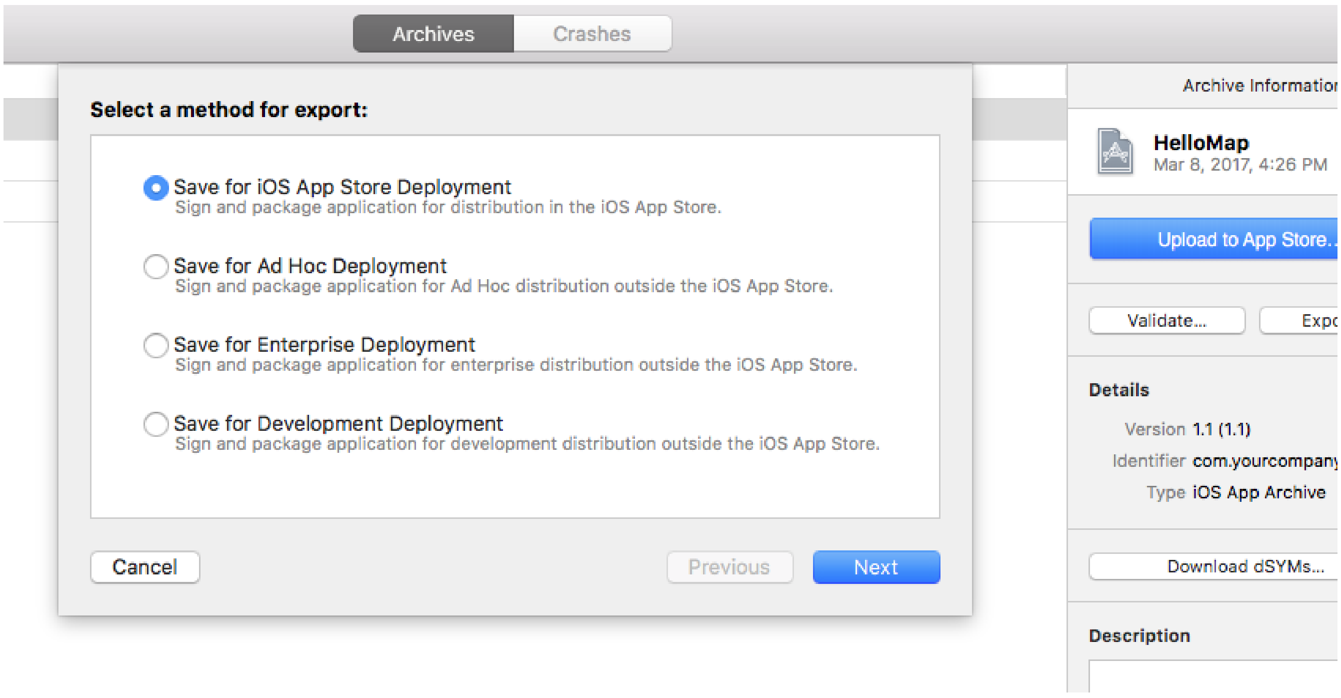
App Submission Requirements

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



Select a method for export:

- [✓ Save for iOS App Store Deployment]
  Sign and package application for distribution in the iOS App Store.
- Save for Ad Hoc Deployment
  Sign and package application for Ad Hoc distribution outside the iOS App Store.
- Save for Enterprise Deployment
  Sign and package application for enterprise distribution outside the iOS App Store.
- Save for Development Deployment
  Sign and package application for development distribution outside the iOS App Store.

Archive Informatio
HelloMap
Mar 8, 2017, 4:26 PM

Details
Version 1.1 (1.1)
Identifier com.yourcompany
Type iOS App Archive

Description

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>
```

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*.

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.

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 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 NMAMapAnimationLinear 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.

Resolution and Text Size

By default HERE SDK uses high-resolution (512 x 512 pixels) map tiles. You can set useHighResolutionMap property in NMAMapView to NO to use lower-resolution (256 x 256 pixels) map tiles instead.

You can also use mapPPI property to change the map pixel-per-inch setting. The default setting is NMAMapPPILow. Setting the property to NMAMapPPIHigh makes street labels and other texts to be bigger.

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 NMAMapViewMinimumZoomLevel 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 =
(NMAMapViewMinimumZoomLevel + NMAMapViewMaximumZoomLevel)/2.0f;
```

Animations

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

- `NMAMapAnimationNone`
- `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
     withAnimation:(NMAMapAnimation) animation
```

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

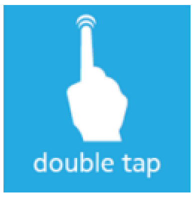
```
// Move to Vancouver preserving zoom level
NMAGeoCoordinates* coord = [[NMAGeoCoordinates alloc]
 initWithLatitude:49.0
 longitude:123.0];
[mapView setGeoCenter:coord
 zoomLevel:NMAMapViewPreserveValue
 withAnimation:NMAMapAnimationNone];
```

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.

2 finger tap

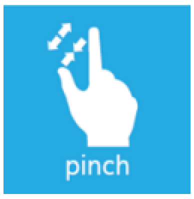
To zoom out a fixed amount, tap the screen with two fingers. Tap continuously to make a continuous zoom.

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

2 finger pan

Press and hold two fingers to the screen and move them in the same direction. This gesture does not have a predefined map action.

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.

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.

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];
```

Gesture Delegation

To receive notifications of gestures, you can implement `NMAMapGestureDelegate` and use the corresponding handler methods. For example:

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

Note: Implementing the handler method does not have any impact on how the map responds to the gesture. For example, the logic within `mapView:didReceiveDoubleTapAtLocation:` is called when the map has been zoomed in.

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;
```

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 8: NMAMapSchemeNormalDay

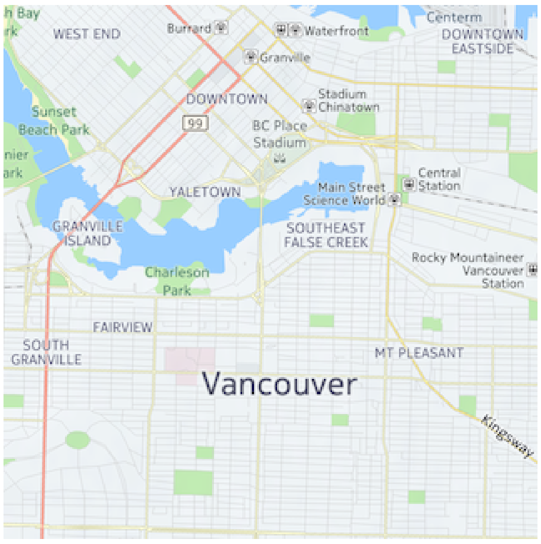


Figure 9: NMAMapSchemeSatelliteDay

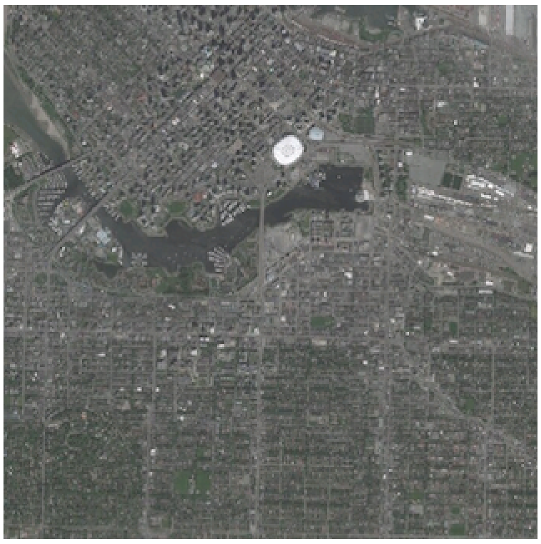


Figure 10: NMAMapSchemeHybridDay

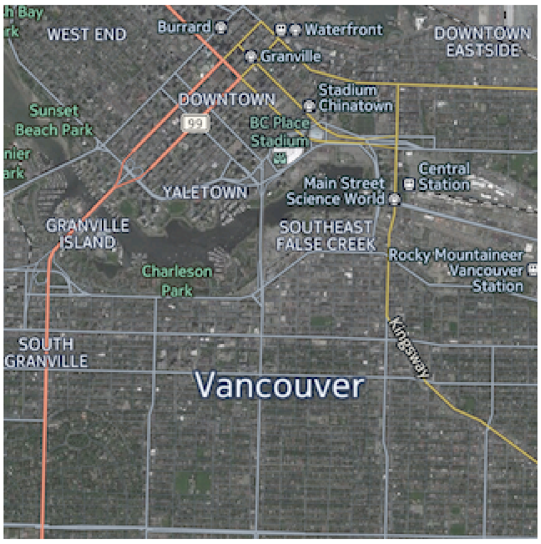
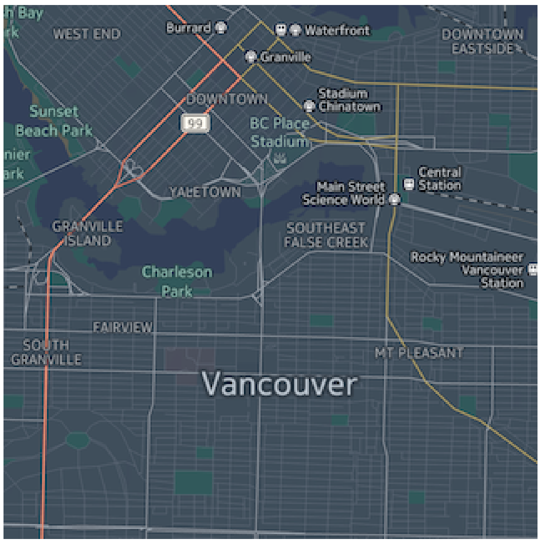


Figure 11: NMAMapSchemeNormalNight



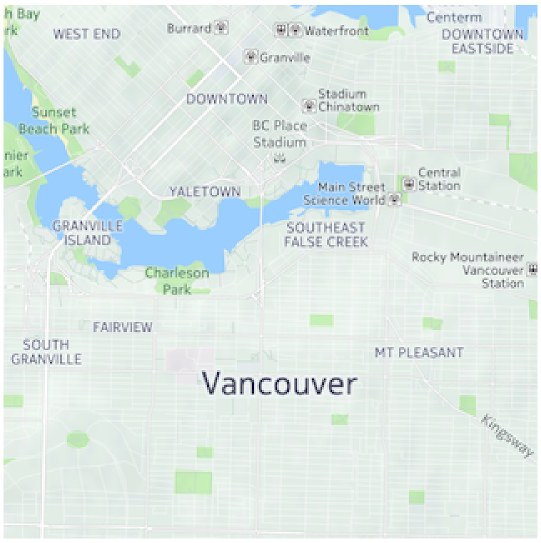


Figure 12: NMAMapSchemeTerrainDay

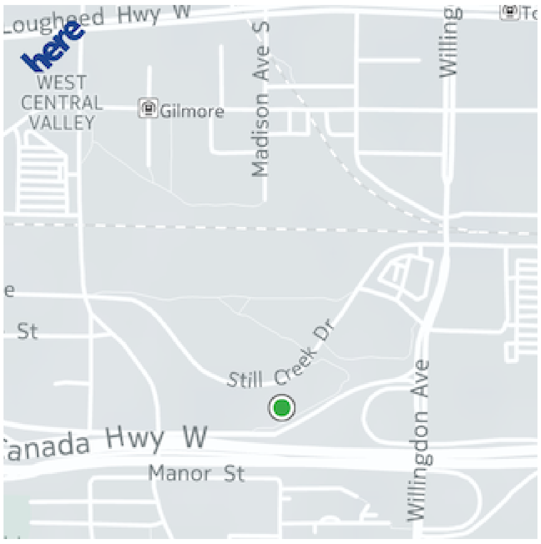


Figure 13: NMAMapSchemeReducedDay

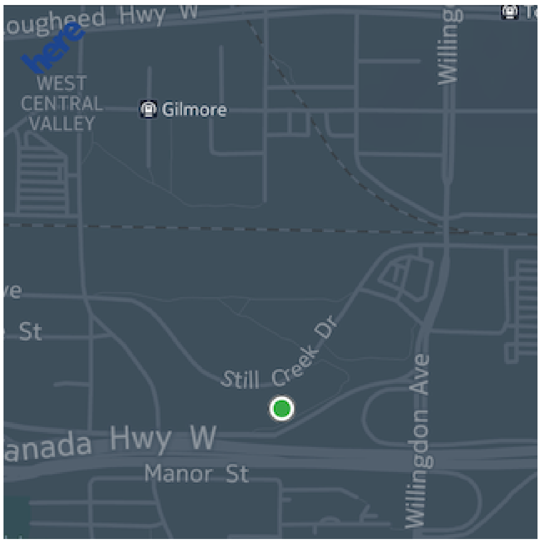


Figure 14: NMAMapSchemeReducedNight

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.

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
- `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.

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.

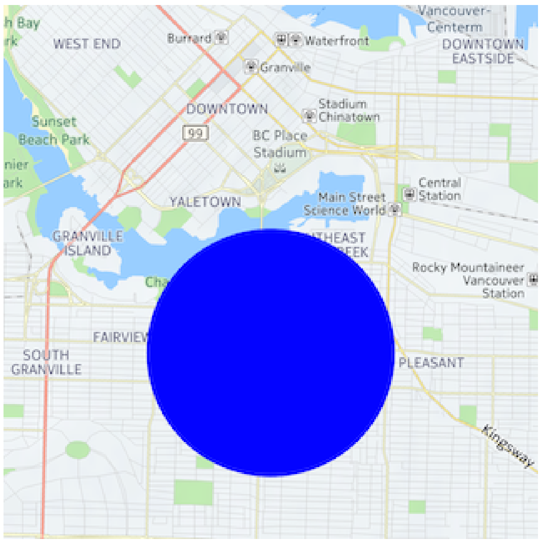
To use a map container, create one or more map objects and add them to the container using `addMapObject` method. To show the objects on a map, add the map container to the map with `addMapObject` method of `NMAMapView`.

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.

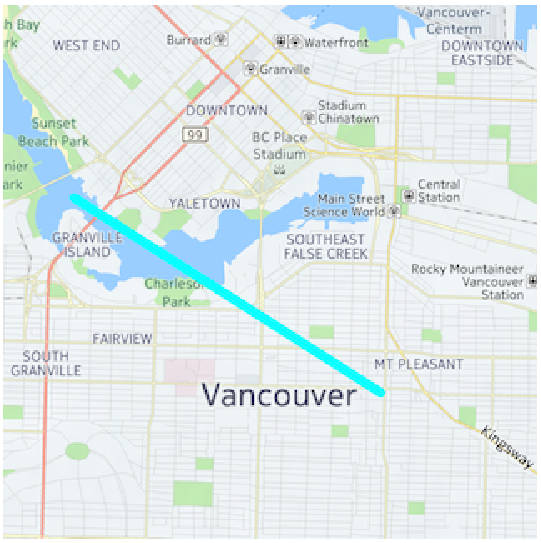
Figure 15: A `MapCircle` object



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.

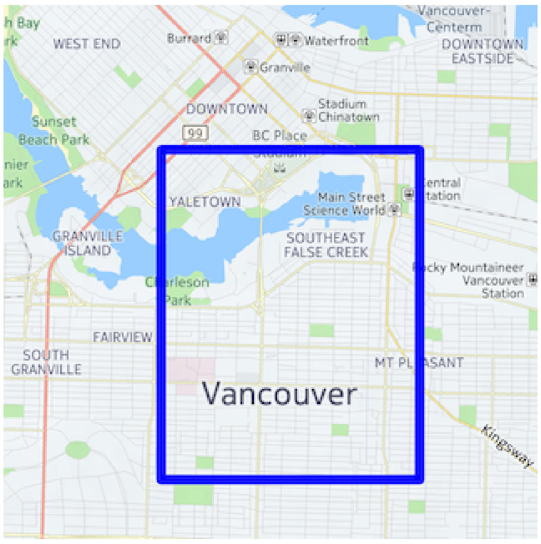
Figure 16: A `MapPolyline` object



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 17: A `MapPolygon` object with transparent fill and a border



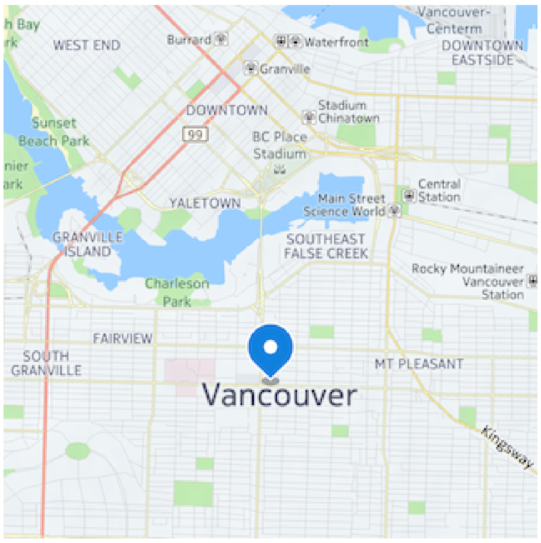
Note: Polygons that have a path that crosses over itelf are not supported. For example, it is not possible to create a "bowtie" shape using four line segments where one line segment crosses another. However, you can create the shape using two triangles.

NMAMapMarker class

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

Custom icons can be provided as a `UIImage`.

Figure 18: 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`.

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. `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.

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:` protocol method.

Dividing a Map and Using Tile Coordinates

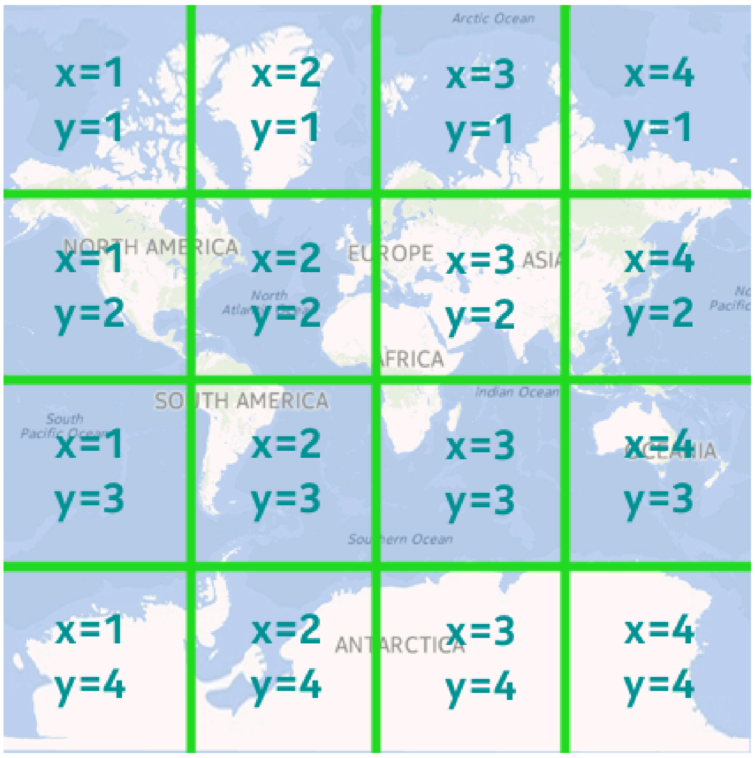
`NMAMapTileLayer` uses a scheme that divides the world map into tiles specified by x, y, and zoom level coordinates. This coordinate system is used by the `NMAMapTileLayerDataSource` protocol when it requests tiles.

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

- at level 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 19: 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 or 512 x 512 pixels.

   Note: As long as your tiles conform to 256 by 256 or 512 by 512 pixels, the `NMAMapView` can scale the image to fit the current map resolution. For example, if `useHighResolutionMap` property is set to YES (512 by 512 pixels) but your tiles are 256 by 256 pixels, the `NMAMapView` enlarges your image tiles.

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. 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
   withAnimation:NMAMapAnimationNone ];
}

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

  // 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;

  // 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:` method.
2. Create an `NMAMapTileLayer` object and set its properties to correspond to the tile data source. Set `dataSource` property.
3. Add the `NMAMapTileLayer` object to the `NMAMapView` by calling `addMapTileLayer:` method.

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. Ensure the properties you set on the tile layer match the data you are supplying via the `NMAMapTileLayerDataSource` protocol.
3. 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.
4. If requesting a specific tile is constantly failing, consider implementing `mapTileLayer:hasTileAtX:y:zoomLevel:` returning NO.
5. Use the provided disk caching mechanism.

Positioning

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.

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 21 for recommended strings.

To start receiving real time positioning updates, the application needs to call `NMAPositioningManager startPositioning` which uses the internal GPS as the 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 `rawPosition` property. `rawPosition` is a position value from the current data source that has not been modified by HERE SDK engine. 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: If you enable background location updates in the Xcode project, then `NMAPositioningManager` provides position updates even when your application is no longer in the foreground. To avoid this behavior, you can either stop the positioning manager before the app goes into the background, or you can disable this setting in Xcode.

Creating Position Logs

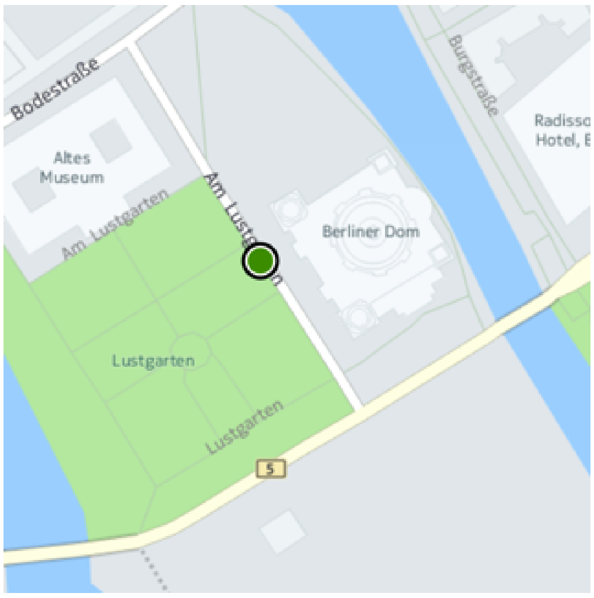
You can also use HERE SDK to create GPX logs that can be replayed by `NMALoggedPositionSource`. To do this, set `logType` property in `NMAPositionManager` to the value `NMAPositionLogTypeDataSource` indicating that the position received from the current data source should be logged. GPX logs are created 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 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` and `NMAMapCircle`.

Figure 20: An NMAPositionIndicator



Each `NMAMapView` instance has an `NMAPositionIndicator` instance which can be accessed from `NMAMapView positionIndicator` property.

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*.

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

NMARouteManager

The `NMARouteManager` class is responsible for calculating an `NMARoute` using a list of stops and an `NMARoutingMode`. It also provides an `NMARouteManagerDelegate` protocol for monitoring calculation progress and triggering appropriate callback methods upon completion. To calculate a route, the application needs to call `calculateRouteWithStops:routingMode:` method. `NMARouteManager` only supports one routing request at a time. Making another request before completion of the current request is not supported.

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
- `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.

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: method of NMARouteManager triggers a route calculation while NMARouteManagerDelegate implements callback methods 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
- tta - the NMARouteTta indicating the estimated time to arrival
- 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.tta.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 ttaForSubleg:0].duration;
}
```

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.

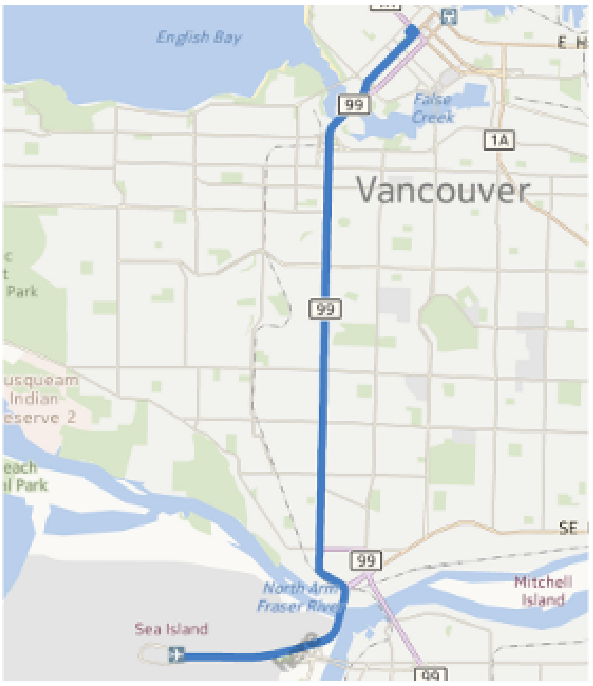
NMARouteElement and NMARoadElement

`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:`.

Figure 21: An `NMAMapRoute` added to an `NMAMapView`



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

1. Adopt `NMARouteManagerDelegate` protocol and create an `NMARouteManager`

```
@interface ClassName : NSObject <NMARouteManagerDelegate>
{
 // Setup your class
}

(void)setup
{
 // Create a NMARouteManager.
 NMARouteManager* routeManager = [NMARouteManager sharedRouteManager];
```

```
  // Setup delegate
  [routeManager setDelegate:self];
 }
```

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

```
NSMutableArray* stops = [[NSMutableArray alloc] initWithCapacity:4];
NMAGeoCoordinates* geoCoord1 = [[NMAGeoCoordinates alloc]
initWithLatitude:49.1966286 longitude:-123.0053635];
NMAGeoCoordinates* geoCoord2 = [[NMAGeoCoordinates alloc]
initWithLatitude:49.1947289 longitude:-123.1762924];
[stops addObject:geoCoord1];
[stops addObject:geoCoord2];
```

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

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

4. Calculate the route

```
[routeManager calculateRouteWithStops:stops routingMode:routingMode];
```

5. To receive the results of the route calculation, implement `NMARouteManagerDelegate` protocol method `routeManager:didCalculateRoutes:withError:violatedOptions:` in your delegate class.

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

```
-(void) routeManager: (NMARouteManager*)routeManager
    didCalculateRoutes:(NSArray*)routes
    withError:(NMARouteManagerError)error
    violatedOptions:(NSArray*)violatedOptions
{
    // If the route was calculated successfully
    if (!error && routes && routes.count > 0)
    {
        NMARoute* route = [routes objectAtIndex:0];
        // Render the route on the map
        NMAMapRoute *mapRoute = [NMAMapRoute mapRouteWithRoute:route];
        [mapView addMapObject:mapRoute];
    }
    else if (error)
    {
        // Display a message indicating route calculation failure
    }
}
```

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.
- NMARouteManagerError enum - represents values describing possible route calculation errors such as NMARouteManagerErrorNone or NMARouteManagerErrorViolatesOptions

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.

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
```

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.

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.

Request	NMAPlaces method	Purpose
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.

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
```

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)
- 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];
  //...
 }
}
```

Text AutoSuggestion Requests

HERE Places Search API also supports text autosuggestion requests. This type of request is used for retrieveing a list of suggested search terms (NMAAutoSuggestTypeQuery), 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".

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`. `NMAAutoSuggestTypeQuery` contains additional results through the `NMAAutoSuggestionRequest` object.

```
+(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;
   caseNMAAutoSuggestTypeQuery:
   {
    typeString = @"Query";
   }
    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:]
    // ...
   }
   else if (type == NMAAutoSuggestTypeQuery) {

    NMAAutoSuggestionRequest* autoSuggestionRequest =
  ((NMAAutoSuggestQuery*)item).autoSuggestionRequest;

    // Call autoSuggestionRequest startWithBlock:^(NMARequest, id, NSError) {}
    // ...
   }
  }
  return YES;
 }
```

Chapter 4
Supplemental Information

Topics:

This section provides supplemental information for using HERE iOS SDK.

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 12.

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 10 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

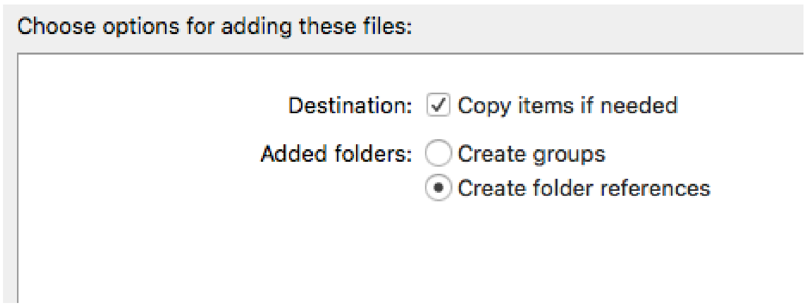
1. From Xcode menu, select **File** > **New** > **Project** to open the New project dialog (or press Shift + Command + N).
2. Select **iOS** > **Application** > **Single View Application** as the application type you want to create. Press Next.
3. In the next dialog, enter your **Product Name** (such 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 22: Add File to Target

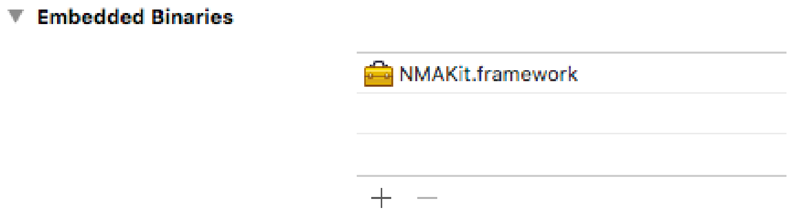

Choose options for adding these files:

Destination: [✓] Copy items if needed

Added folders: ○ Create groups
● Create folder references

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

Figure 23: Embedded Binaries


▼ Embedded Binaries

NMAKit.framework

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.

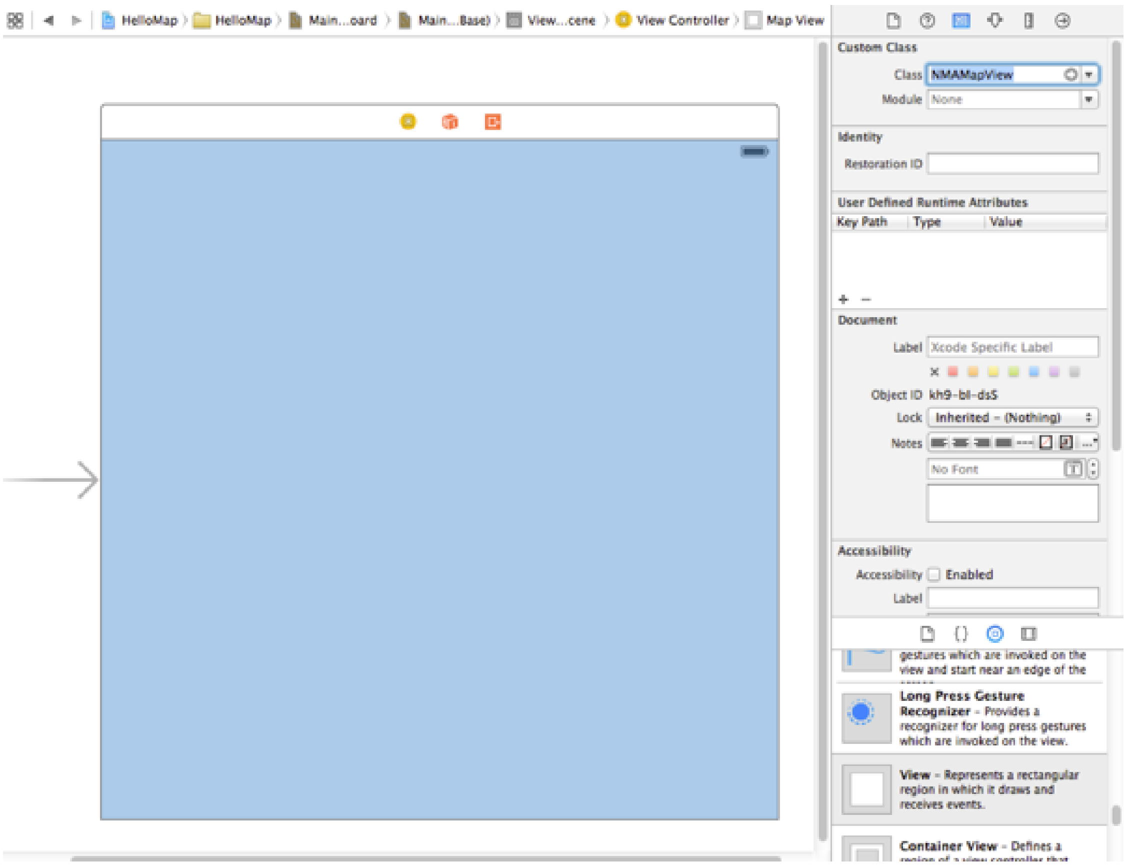
Create the Map View

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 24: MapView


HelloMap › HelloMap › Main...oard › Main...Base) › View...cene › View Controller › Map View

Custom Class
Class: NMAMapView
Module: None

Identity
Restoration ID

User Defined Runtime Attributes
Key Path | Type | Value

Document
Label: Xcode Specific Label
Object ID: kh9-bl-dsS
Lock: Inherited - (Nothing)
Notes: No Font

Accessibility
Accessibility: ☐ Enabled
Label

gestures which are invoked on the view and start near an edge of the

Long Press Gesture Recognizer - Provides a recognizer for long press gestures which are invoked on the view.

View - Represents a rectangular region in which it draws and receives events.

Container View - Defines a

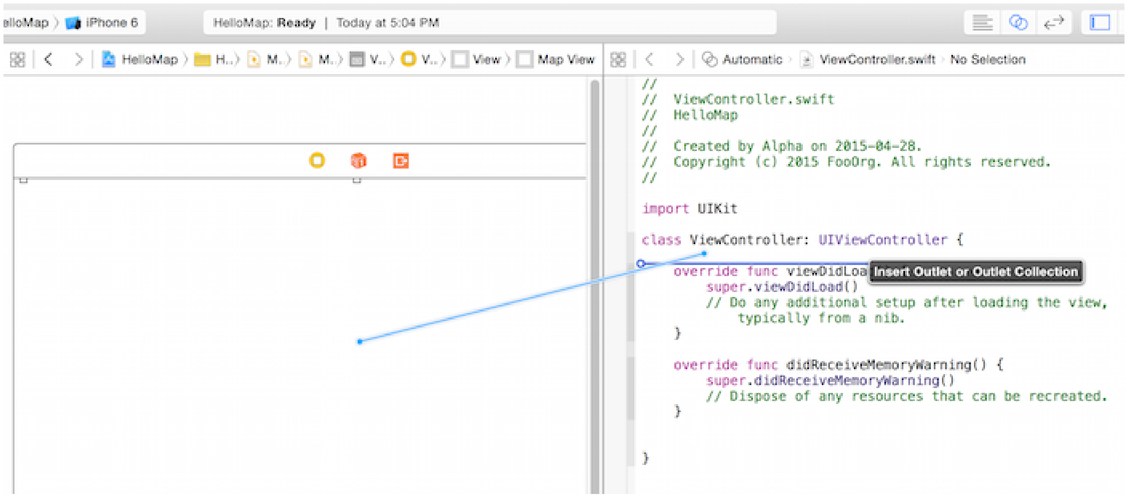
2. Create an outlet for `NMAMapView` in *ViewController*.
   a. Select `Main.storyboard` in the navigator.
   b. Press Command + Option + Return to open the Assistant Editor. It should show *ViewController.swift*.
   c. Add the following import statement to the top of this file:

   ```
   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 25: Create an Outlet


elloMap 〉 iPhone 6
HelloMap: Ready | Today at 5:04 PM
HelloMap 〉 H..〉 M..〉 M..〉 V..〉 V..〉 View 〉 Map View
Automatic › ViewController.swift › No Selection
Insert Outlet or Outlet Collection

```
//
//  ViewController.swift
//  HelloMap
//
//  Created by Alpha on 2015-04-28.
//  Copyright (c) 2015 FooOrg. All rights reserved.
//

import UIKit

class ViewController: UIViewController {

    override func viewDidLoa
        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.
    }

}
```

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)
 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.

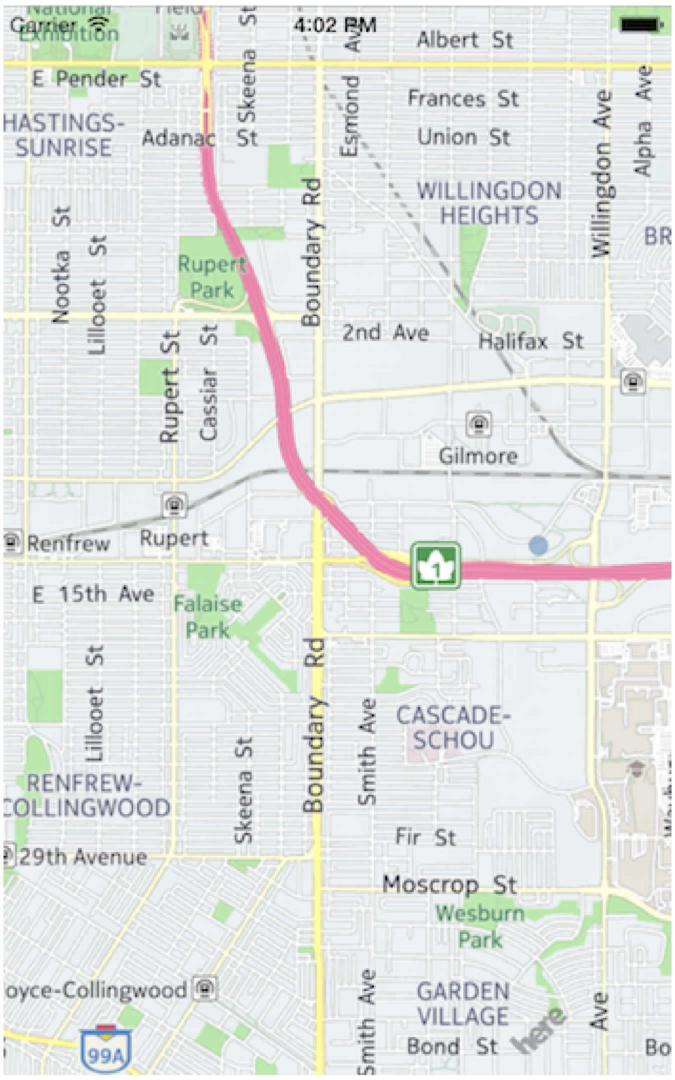
   ```
   import NMAKit
   ```

   b. Add the following in `application(_:didFinishLaunchingWithOptions)` function replacing `YOUR_APP_ID` and `YOUR_APP_CODE` with the credentials that you received from your *http://developer.here.com*.

   ```
   NMAApplicationContext.set(appId: "YOUR_APP_ID",
           appCode: "YOUR_APP_CODE")
   ```

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 26: 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 annotatations 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.

Chapter 5
Coverage Information

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*
- *Routing*
- *Point Addresses* (such as house numbers)
- *Online Geocoding / Reverse Geocoding*
- *Online Places and Search*