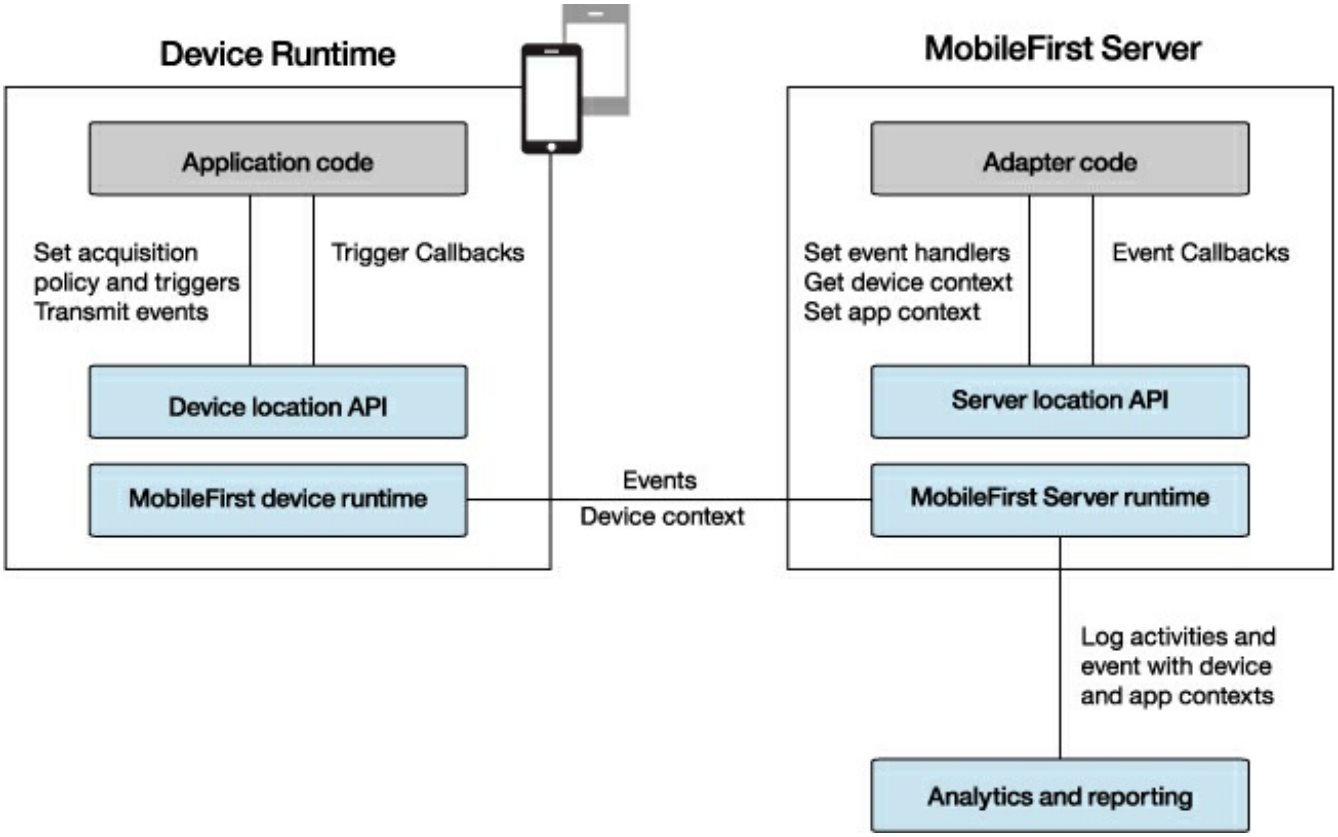
Location services in native iOS applications

This tutorial covers the following topics:

- Architecture
- Acquisition policy
- Start acquisition
- Triggers
- Events
- Sample application

Architecture



The application code on the mobile device, in the form of an acquisition policy, controls the collection of data from device sensors.

The collected data is referred to as the device context.

When a change occurs in the device context, such as a change in the geolocation of the device or the fact that it entered a Wi-Fi zone, triggers can be activated.

The triggers specify that an action occurs: either a callback function is called, or an event is sent to the server, based on the device context.

Events are created by triggers and application code, and include a snapshot of the device context at the time of their creation.

Events are buffered on the client and are transmitted to the server periodically.

The server might process the event later.

During the event transmission process, the device context is synchronized transparently to the server.

To handle the events, the server uses adapter application code.

This code sets up event handlers on the server. These handlers filter event data and pass matching events to a callback function.

The code also accesses the client device context (its location and Wi-Fi network information), and sets an application context.

Server activities and received events are logged, together with the device and application contexts, for future reporting and analytics.

Acquisition policy

An acquisition policy defines how acquisition takes place.

```
WLAcquisitionPolicy* policy = [[WLAcquisitionPolicy alloc] init];
```

Geo acquisition policy

Add a `UIRequiredDeviceCapabilities` node in `yourAppName-info.plist` with items:

- `location-services`
- `gps` (when `enableHighAccuracy=true`)

▼ Required device capabilities	Array	(3 items)
Item 0	String	location-services
Item 1	String	gps

Then in your code, you can use:

```
WLGeoAcquisitionPolicy* geoPolicy = [WLGeoAcquisitionPolicy getLiveTrackingProfile];
[policy setGeoPolicy:geoPolicy];
```

iOS 8

Add the following nodes to `yourAppName-info.plist`:

- `NSLocationWhenInUseUsageDescription`
- `NSLocationAlwaysUsageDescription`

NSLocationWhenInUseUsageDescription	▲▼	String
NSLocationAlwaysUsageDescription	▲▼	String

LiveTracking is a preset profile that uses the most accurate settings to track the device.
Additional configuration options:

- `RoughTracking`
- `PowerSaving`
- Custom settings

> For more information, see the topic about setting an acquisition policy, in the user documentation.

Wi-Fi acquisition policy

Add `wifi` to your `UIRequiredDeviceCapabilities` node in `yourAppName-info.plist`.

▼ Required device capabilities ▲▼	Array	(3 items)
Item 0	String	location-services
Item 1	String	gps
Item 2	String	wifi

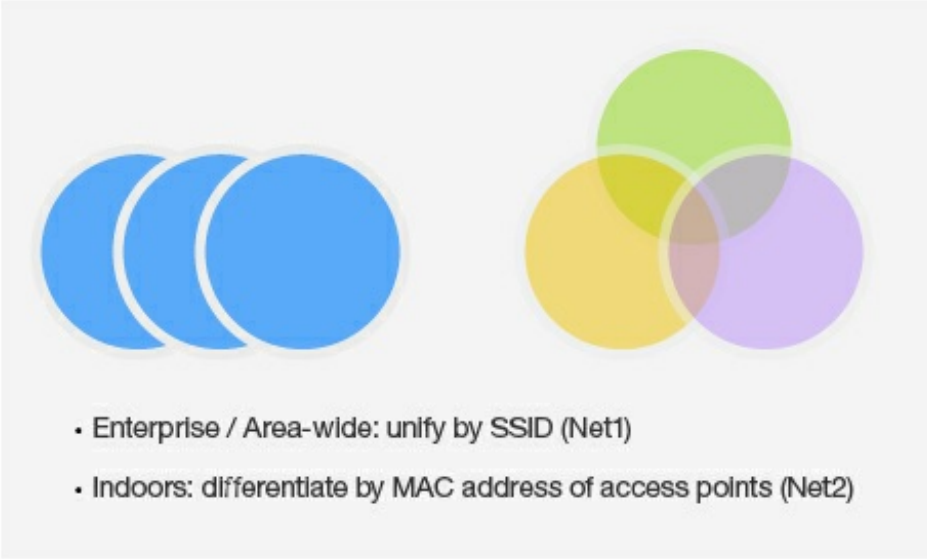
Then in your code, you can use:

```
WLWifiAcquisitionPolicy* wifiPolicy = [[WLWifiAcquisitionPolicy alloc] init];
WLWifiAccessPointFilter* filter1 = [[WLWifiAccessPointFilter alloc] init:@"Net1"];
WLWifiAccessPointFilter* filter2 = [[WLWifiAccessPointFilter alloc] initWithSSID:@"Net2" MAC:@"*"];
[wifiPolicy setInterval:10000];
[wifiPolicy setAccessPointFilters:[NSMutableArray arrayWithObjects:filter1, filter2, nil]];
[policy setWifiPolicy:wifiPolicy];
```

The `interval` parameter is the polling interval, in milliseconds. Wi-Fi polling is performed at each interval.

the `accessPointFilters` parameter lists access points of interest.
In the above example, the acquisition policy works as follows:

- Ignores everything except `Net1` and `Net2` – This policy is helpful in dynamic environments, such as mobile hotspots.
- Considers all `Net1` access points as one single access point.
- Differentiates `Net2` access points by MAC address.



> For more information, see the topic about setting an acquisition policy, in the user documentation.

Triggers

You can setup triggers:

- Geo/Wi-Fi fences: `Enter`, `Exit`, `Dwell Inside`, `Dwell Outside`
- Movement: Geo: `PositionChange`, Wi-Fi: `VisibleAccessPointsChange`
- Wi-Fi: `Connect` / `Disconnect`

```
//Init
WLTriggersConfiguration* triggers = [[WLTriggersConfiguration alloc] init];
//Center of circle
WLCoordinate* center = [[WLCoordinate alloc] initWithLatitude:40.689167 longitude:-74.044444];
//Circle
WLCircle* circle = [[WLCircle alloc] initWithCenter:center radius:100];
//Event
NSMutableDictionary* event = [NSMutableDictionary dictionaryWithObjectsAndKeys:@"me", @"bring",
@"huddledMasses", @"your", nil];
//Geo Trigger
WLGeoEnterTrigger* enterTrigger = [[WLGeoEnterTrigger alloc] init];
[enterTrigger setArea:circle];
[enterTrigger setCallback:nil]; //out of scope, see documentation
[enterTrigger setEvent:event];
[[triggers getGeoTriggers] setObject:enterTrigger forKey:@"trigger1"];
```

In this example, `trigger1` is an `Enter` trigger which is activated when the device enters the defined circle (longitude, latitude, and radius).

When a trigger activates, it can call a callback function and/or create an event to be sent to the server.

> For more information, see the topic about triggers, in the user documentation.

Start acquisition

Use the policy that you defined in section Acquisition policy and the triggers that you defined in section Triggers to start the acquisition.

```
WLLocationServicesConfiguration* config = [[WLLocationServicesConfiguration alloc] init];
[config setPolicy:policy];
[config setTriggers:triggers];
[config setFailureCallbacks:nil];//see documentation
[[[WLClient sharedInstance] getWLDevice] startAcquisition:config];
```

Events

Client side

Events are generated by triggers, as explained in section Triggers.

```
NSMutableDictionary* event = [NSMutableDictionary dictionaryWithObjectsAndKeys:@"me", @"bring",
@"huddledMasses", @"your", nil];
[enterTrigger setEvent:event];
```

Events can also be generated manually by calls to the API:

```
[[WLClient sharedInstance] transmitEvent:event immediately:YES];
<p>
```

Where `event` is a `NSDictionary` object and `immediately` is a `boolean` value.

Server side

In the adapter code, create event handlers:

```
WL.Server.createEventHandler(filter, handlerFunction);
```

The events that match the filter are passed to `handlerFunction`.
Filter examples:

- `{status: "platinum"}` – Handle platinum members only
- `{hotel: { country: "USA" } }` – Hotels in the USA
- `{} – All events`

Register the event handlers:

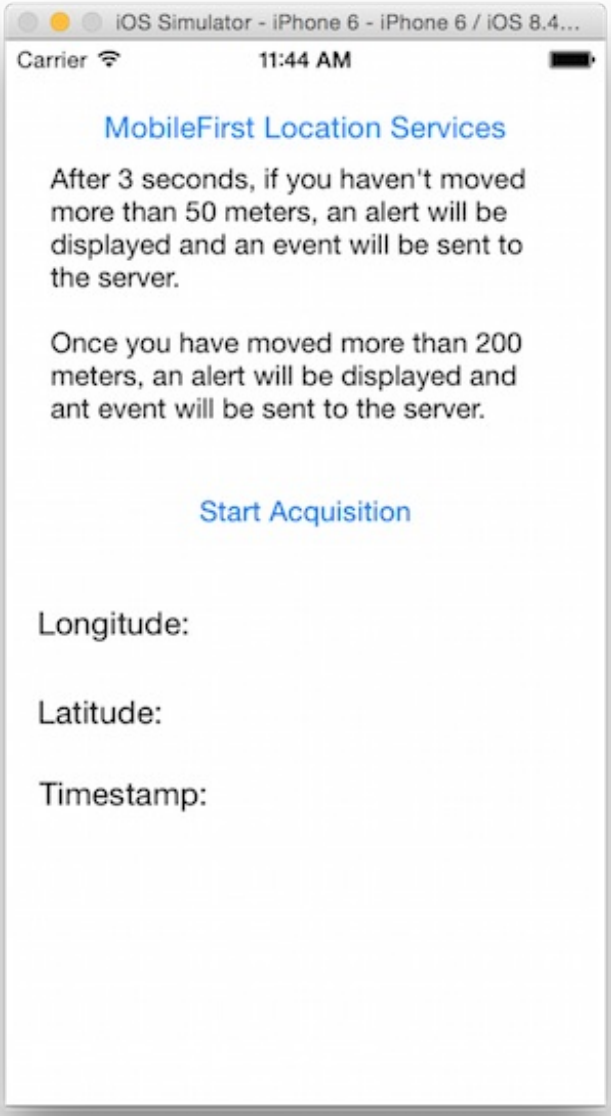
```
WL.Server.setEventHandlers([...]);
```

> For more information, see the topic about working with geofences and triggers, in the user documentation.

Sample application

Click to download (https://github.com/MobileFirst-Platform-Developer-Center/LocationServices/tree/release71) the MobileFirst project.

Click to download (https://github.com/MobileFirst-Platform-Developer-Center/LocationServicesObjC/tree/release71) the Native project.

The `LocationServices` sample demonstrates the following features:

- Acquiring an initial position
- Using a Geo profile
- Using Geo triggers for DwellInside, Exit area, and PositionChange
- Transmitting events to the server on DewllInside and Exit area
- Ongoing acquisition