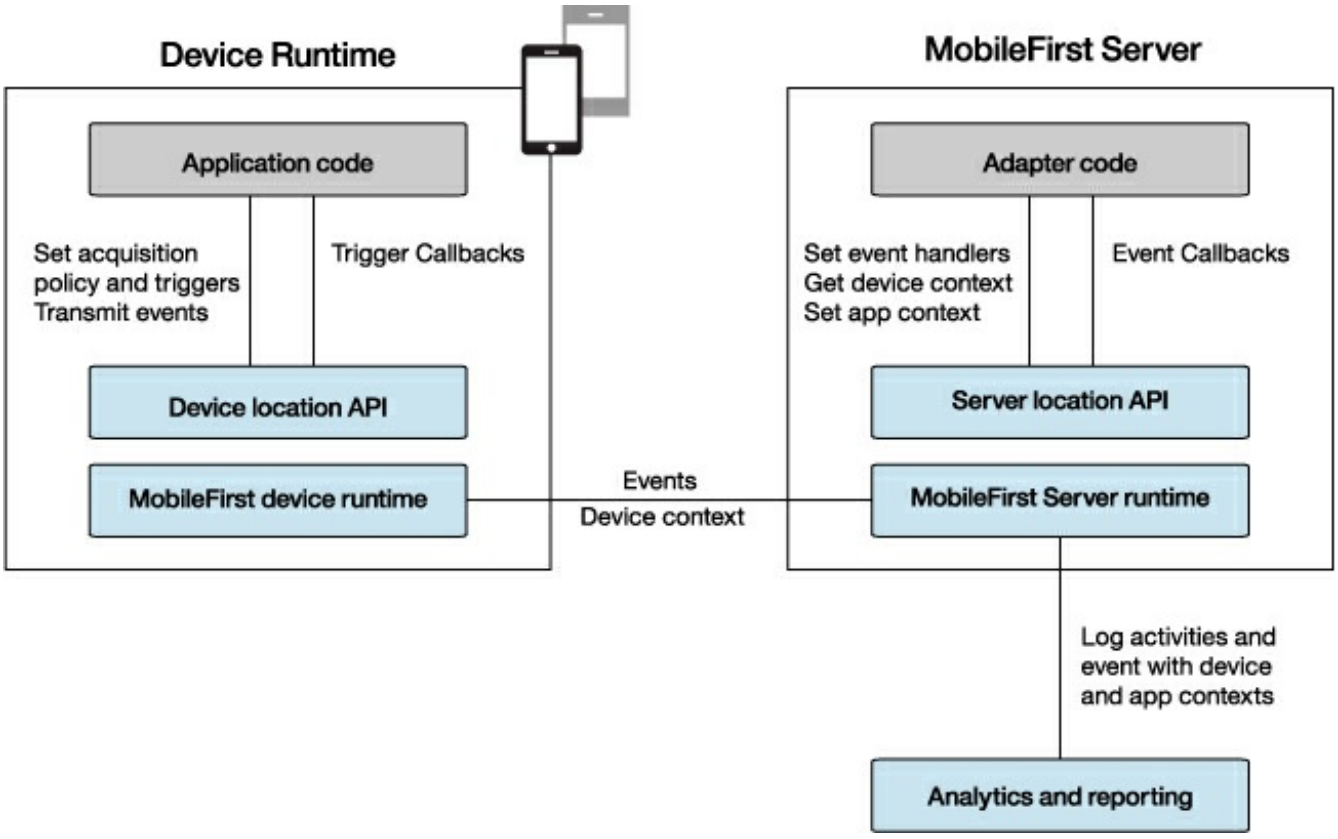
Location services in hybrid applications

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 WiFi zone, triggers can be activated.

The triggers specify that an action should occur: 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 WiFi 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.

Basic API

Client JavaScript

Note: The following functions should be called after the IBM MobileFirst Platform Foundation framework is initialized (within or after the `wlCommonInit()` function).

```
WL.Device.startAcquisition(policy, triggers, onFailure)
```

- **policy**: how do you acquire the sensor data?
- **triggers**: what do you act on and how?
- **onFailure**: how do you handle acquisition failures?

Server JavaScript

```
WL.Server.setEventHandlers(eventHandlers);
```

eventHandlers: what events do you act on and how?

Acquisition Policy

An acquisition policy defines how acquisition takes place.

```
var policy = {
  Geo: WL.Device.Geo.Profiles.LiveTracking()
,
  Wifi: {
    interval: 10000,
    accessPointFilters: {
      [{
        SSID: "Net1"
      }, {
        SSID: "Net2",
        MAC: "*"
      }]
    }
  }
};
```

Geo Acquisition Policy

```
  Geo: WL.Device.Geo.Profiles.LiveTracking(),
```

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

- `RoughTracking`
- `PowerSaving`
- Custom settings

> For more information, see the topic "Setting an acquisition policy" of the user documentation.

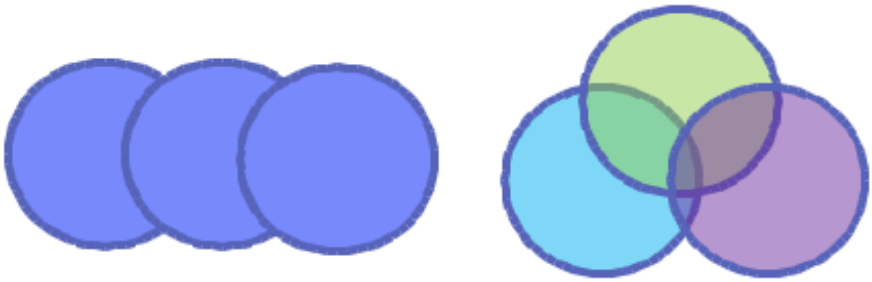
Wifi Acquisition Policy

```
 Wifi: {
   interval: 10000,
   accessPointFilters:
{
     [{
       SSID: "Net1"
     }, {
       SSID: "Net2",
       MAC: "*"
     }]
   }
 }
```

`interval` is the polling interval, in milliseconds; WiFi polling is performed each interval.

`accessPointFilters` are access points of interest. In the above example, the acquisition policy:

- Ignores everything except "Net1" and "Net2" – doing so assists in dynamic environments, such as when there are mobile hotspots.
- Considers all "Net1" access points as though they were one access point.
- Differentiates "Net2" access points by MAC address.



- Enterprise / Area-wide: unify by SSID (Net1)
- Indoors: differentiate by MAC address of access points (Net2)

> For more information, see the topic "Setting an acquisition policy" of the user documentation.

Triggers

```
var triggers = {
  Geo: {
    trigger1: {
      type: "Enter",
      circle: {
        longitude: -74.04444,
        latitude: 40.689167,
        radius: 100
      },
      callback: libertyAtLast,
      eventToTransmit: {
        event: {
          bring: "me",
          your: "huddledMasses
"
        }
      }
    }
  }
};
```

You can setup triggers for

- Geo/Wifi fences (`Enter`, `Exit`, `Dwell Inside`, `Dwell Outside`)
- Movement (Geo: `PositionChange`, WiFi: `VisibleAccessPointsChange`)
- WiFi `Connect` / `Disconnect`

In the above example *trigger1* is an **Enter** trigger which is activated when entering 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 "Triggers" of user documentation.

Events

Client Side

Events are generated by triggers (as explained above).

```
    eventToTransmit: {
      event: {
        bring: "me",
        your: "huddledMasses
"
      }
    }
```

Events can also be generated manually by calling the API:

```
WL.Client.transmitEvent(event,immediate);
```

Server-side

In the adapter code, create event handlers by using:

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

Events that match filter will be passed to `handlerFunction`. Filter examples:

- `{status: "platinum"}` – handle platinum members only
- `{hotel: { country: "USA" } }` – hotels in the USA
- `{}` – all events

Register the event handlers by using:

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

> For more information, see the topic "Working with geofences and triggers" of the user documentation.

Testing hybrid applications

In order to test your application you might need to test the various triggers and error handling logic that your application uses. The **Mobile Browser Simulator** provides capabilities to simulate sensor data and errors. This can be accessed by right-clicking an application environment, and selecting the **Preview** option under the **Run As** menu.

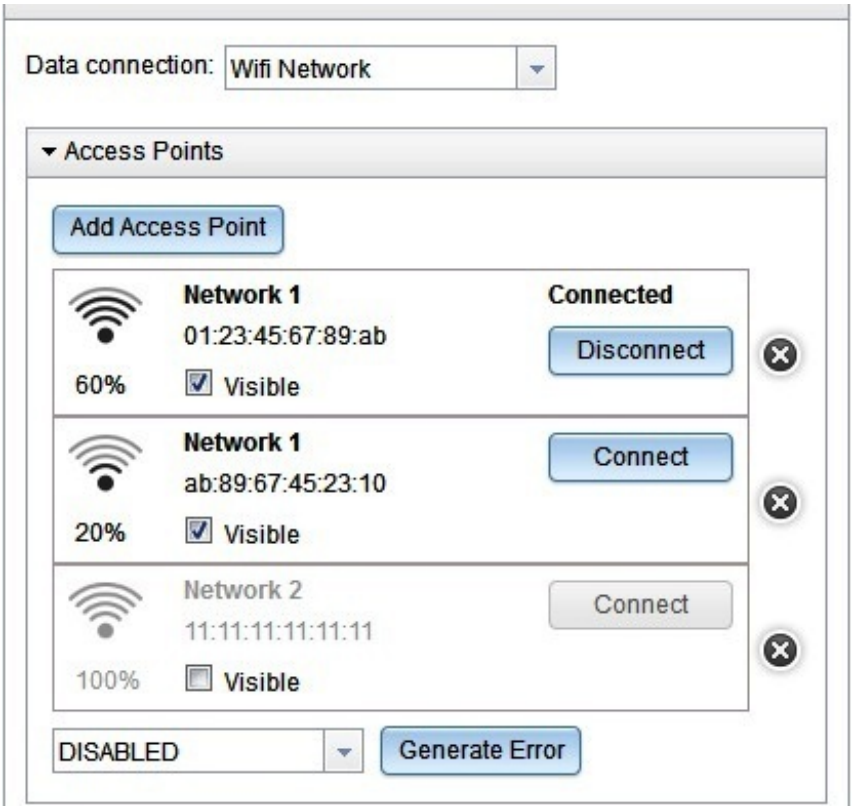
Geo testing

The Geolocation widget can be used to set a specific position through manual entry, or by clicking on the map.

A simple simulation mode is also provided by using the **Step** and **Play** buttons, which move the position in the simulated device at the given speed and in the direction of the given heading.

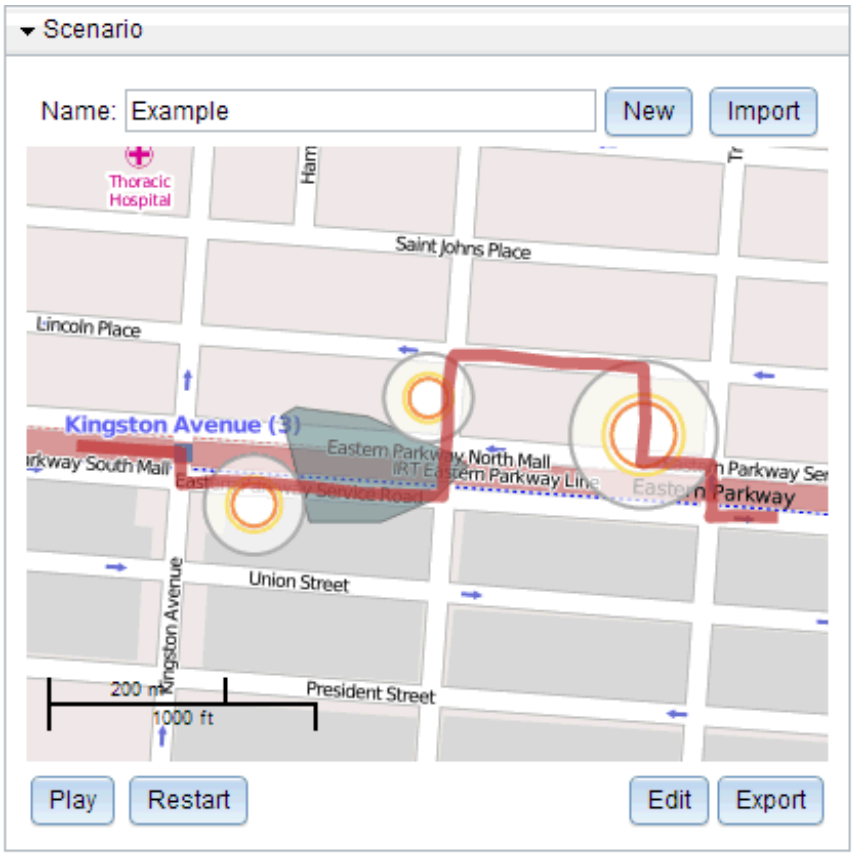
You can also simulate the generation of errors.

WiFi Testing



The Network widget can be used to define simulated access points, configure their signal strengths, and simulate the connection or disconnection to an access point. You can also simulate the generation of errors.

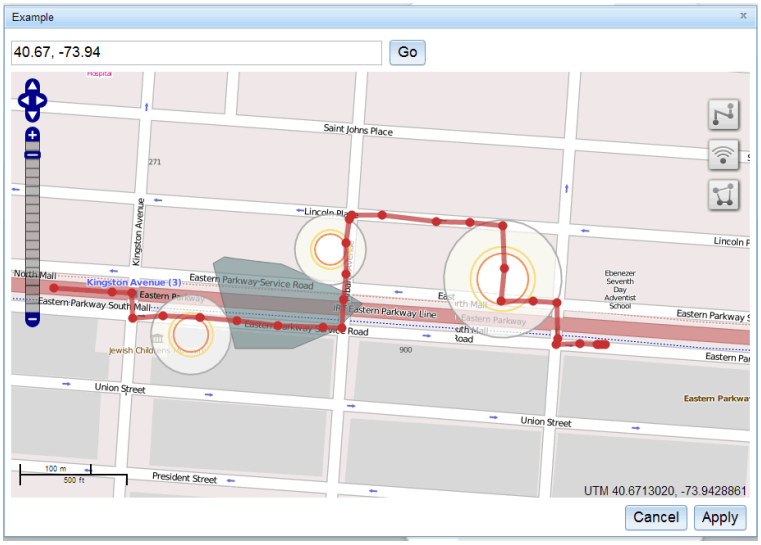
Scenarios



The Scenario widget can be used to automatically simulate a user moving through an environment in a complex way. A scenario consists of:

- The path of the user, and the point when the user reaches each path point
- WiFi access points
- No GPS coverage zones

To open the scenario editor, use the **Edit** button.



Enter a geolocation directly to move the map, or use the arrows to pan and +/- track to zoom.

Click  to define the path of the user.

Click on the map to add each path point; double-click to add the last point. You can drag points to new locations. Click on a point to set the user's arrival time to that point, or to delete the whole path.

Click  to add Wifi access points.

Click on the map and drag to set the area covered by the access point. Click an existing access point to change its SSID and MAC. After clicking, drag to move or resize.

Click  to add no-GPS zones.

Click on the map to add each vertex of the zone; double-click to add the last point. After clicking an existing zone, you can drag to move, resize, or rotate.

When playing a scenario, the position of the user is displayed on the map () and is automatically

updated. The position that is available to the device is shown in the Geolocation widget (and will not change when in a no-GPS zone). WiFi access point visibility and signal strengths are automatically updated. These updates can be seen in the Network widget. Scenarios can be imported and exported to support test reuse.

Sample

The LocationServices sample demonstrates:

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

Sample application

Click to download (http://public.dhe.ibm.com/software/products/en/MobileFirstPlatform/docs/v630/LocationServicesProject.zip) the Studio project.