Using Analytics API in client applications

Overview

MobileFirst Operational Analytics has a few APIs to help a user get started with collecting Analytics. In Cordova, applications start collecting analytics data out of the box. However, for native platforms, iOS and Android, there is some instrumentation that the developer has to implement.

Jump to:

Configuring Analytics on the Client Side

Before you can start collecting the out-of-the-box data that Operational Analytics provides, you first need to import the corresponding libraries and initialize analytics API.

Android

Import Library

```
import com.worklight.common.WLAnalytics;
```

Initialize Analytics

Inside the `onCreate` method of your main activity include:

```
WLAnalytics.init(this.getApplication());
```

iOS

Import Library

```
import "WLAnalytics.h"
```

Initialize Analytics

No initialization is needed for analytics on iOS.

Sending Analytics

Sending Analytics is a crucial step to see client side analytics on the Analytics Server. When collecting Analytics, the analytics logs are stored in a log file on the client device which is sent to the analytics server after using the `send` method of the Analytics API.

JavaScript

```
WL.Analytics.send();
```

Android

```
WLAnalytics.send();
```

Objective-C API

```
[[WLAnalytics sharedInstance] send];
```

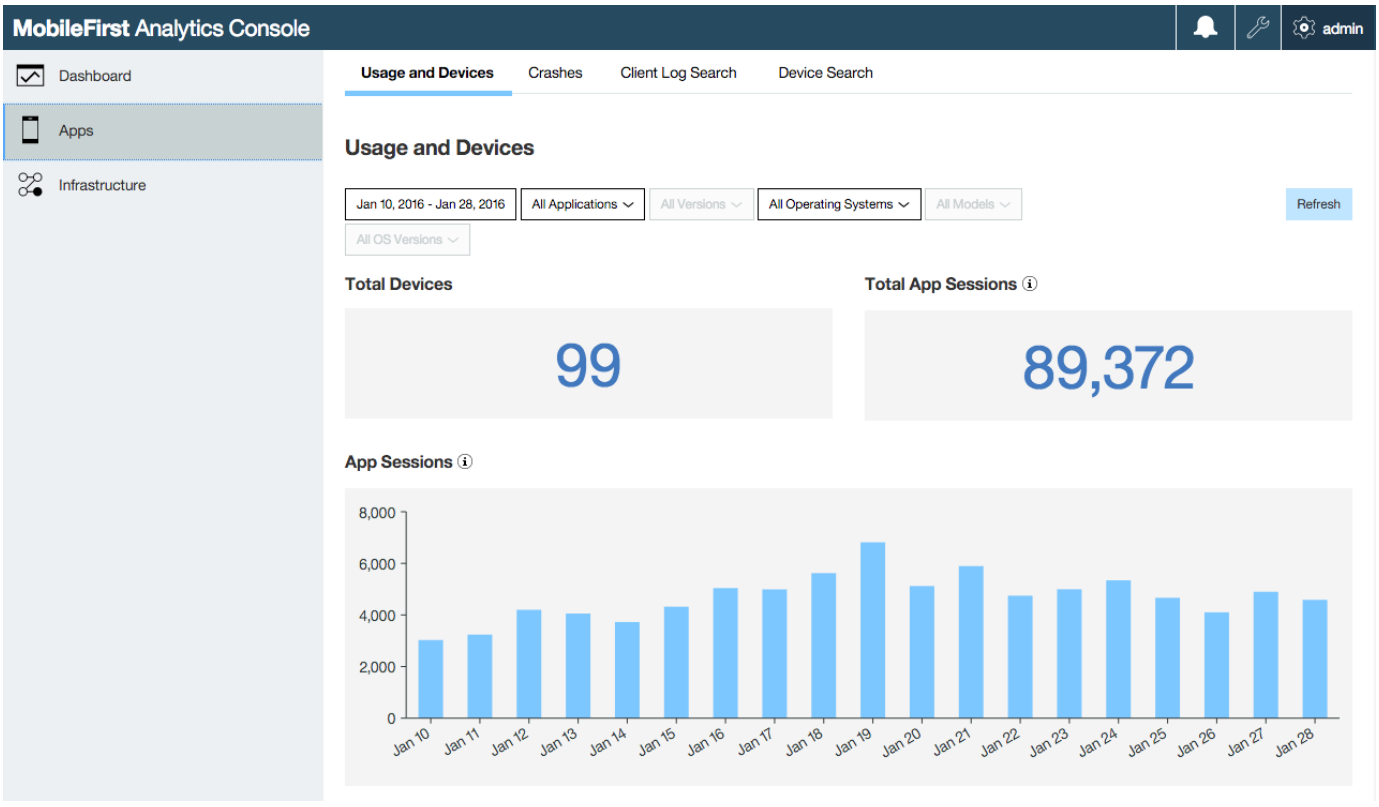
Enabling/Disabling Client Event Types

The Analytics API gives the developer the freedom to enable and disable collecting Analytics on the event they want to visualize on their analytics console.

Client Lifecycle Events

After configuring the Analytics SDK, app sessions will start to be recorded on the user's device. A session in MobileFirst Operational Analytics is recorded when the app is moved from the foreground then to the background, which creates a session on the analytics console.

As soon as the device is set up to record sessions and you send your data, you will see the analytics console populated with data like below.



You can enable or disable the collecting of app sessions with the API below:

Android:

```
//DeviceEvent.LIFECYCLE records app sessions
WLAnalytics.addDeviceEventListener(DeviceEvent.LIFECYCLE);
WLAnalytics.removeDeviceEventListener(DeviceEvent.LIFECYCLE);
```

Objective-C:

```
//DeviceEvent.LIFECYCLE records app sessions
[[WLAnalytics sharedInstance] addDeviceEventListener:LIFECYCLE];
[[WLAnalytics sharedInstance] removeDeviceEventListener:LIFECYCLE];
```

Client Network Activities

Collection on adapters and the network occur in two different locations -- on the client and on the server.

The client is going to collect information like roundtrip time and payload size when you start collecting on the device event `Network`.

The server is going to collect more backend information like server processing time, adapter usage, procedures, etc.

Since the client and the server are each collecting their own information this means that all the charts will not display data until the client is configured to do so. To configure your client you need to start collecting on the device event `NETWORK`.

To enable or disable network events on the client use the API below:

Android:

```
//DeviceEvent.Network records client information about adapters like 'Average Procedure Response Size'
WLAnalytics.addDeviceEventListener(DeviceEvent.NETWORK);
WLAnalytics.removeDeviceEventListener(DeviceEvent.NETWORK);
```

Objective-C:

```
//DeviceEvent.Network records client information about adapters like 'Average Procedure Response Size'
[[WLAnalytics sharedInstance] addDeviceEventListener:NETWORK];
[[WLAnalytics sharedInstance] removeDeviceEventListener:NETWORK];
```

Custom Events

JavaScript API

JavaScript API is used in Cordova applications.

Creating custom events in Cordova is simply just calling:

```
WL.Analytics.log({"key" : 'value'});
WL.Analytics.send();
```

Android API

After setting the first two configurations you can start to log data like in the example below.

```
JSONObject json = new JSONObject();
try {
    json.put("key", "value");
} catch (JSONException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}

WLAnalytics.log("Message", json);
WLAnalytics.send();
```

Objective-C API

Objective-C API is used in iOS applications.

After importing WLAnalytics you can now use the API to collect custom data like below:

```
NSDictionary *inventory = @{
    @"property" : @"value",
};

[[WLAnalytics sharedInstance] log:@"Custom event" withMetadata:inventory];
[[WLAnalytics sharedInstance] send];
```