Using Analytics API in client applications

Overview

MobileFirst Foundation's Operational Analytics provides client-side APIs to help a user get started with collecting Analytics data about the application. This tutorial provides information on how to setup analytics support on the client application and lists available APIs.

Jump to:

Configuring Analytics on the Client Side

Before you can start collecting the predefined data that MobileFirst Operational Analytics provides, you must first import the corresponding libraries to initialize the analytics support.

JavaScript (Cordova)

- In Cordova applications, no setup is required and initialization is built-in.

JavaScript (Web)

- In Web applications, the analytics JavaScript files must be referenced. Make sure you have first added the MobileFirst Web SDK. Review the Adding the MobileFirst SDK to Web applications (../../adding-the-mfpf-sdk/web) tutorial.
  Depending on how you've added the MobileFirst Web SDK, proceed in either of the following ways:
  - Reference Analytics in the `HEAD` element:

```
<head>
   ...
   <script type="text/javascript" src="node_modules/ibm-mfp-web-sdk/lib/analytics/ibmmfpfanalytics.js"></script>
   <script type="text/javascript" src="node_modules/ibm-mfp-web-sdk/ibmmfpf.js"></script>
</head>
```

  Or, if using RequireJS, write:

```
require.config({
   'paths': {
      'ibmmfpfanalytics': 'node_modules/ibm-mfp-web-sdk/lib/analytics/ibmmfpfanalytics',
      'mfp': 'node_modules/ibm-mfp-web-sdk/ibmmfpf'
   }
});

require(['ibmmfpfanalytics','mfp'], function(ibmmfpfanalytics, WL) {
   // application logic.
});
```

  Note that you can select your own namespace instead of "ibmmfpfanalytics".

- For Web applications no listeners are required. Analytics can be enabled and disabled through the `ibmmfpfanalytics.logger` class:

```
ibmmfpfanalytics.logger.config({analyticsCapture: true});
```

> **Important**: Some JavaScript API differences exist between the Cordova and Web SDKs. Please refer to the API Reference topic (http://www.ibm.com/support/knowledgecenter/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/topics/r_apiref.html) in the user documentation.

iOS

Import Analytics Library

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

Initialize Analytics

No setup required. Pre-initialized by default.

Android

Import AnalyticsLibrary

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

Initialize Analytics

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

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

Sending Analytics Data

Sending Analytics is a crucial step to see client-side analytics on the Analytics Server. When data is collected for Analytics, the analytics logs are stored in a log file on the client device. The data from the file is sent to the MobileFirst Analytics server after you use the `send` method of the Analytics API.

It should also be considered to send the captured logs periodically to the server. Sending data at regular intervals ensures that you will see up-to-date analytic data in the MobileFirst Analytics Console.

JavaScript (Cordova)

In a Cordova application, use the following JavaScript API method:

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

JavaScript (Web)

In a Web application, use the following JavaScript API method (depending on the namespace you've selected):

```
wlanalytics.send();
```

iOS

In an iOS application, use the following *Objective-C* API method:

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

or for *Swift* use the API method:

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

Android

In an Android application, use the following Java API method:

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

Enabling/Disabling Client Event Types

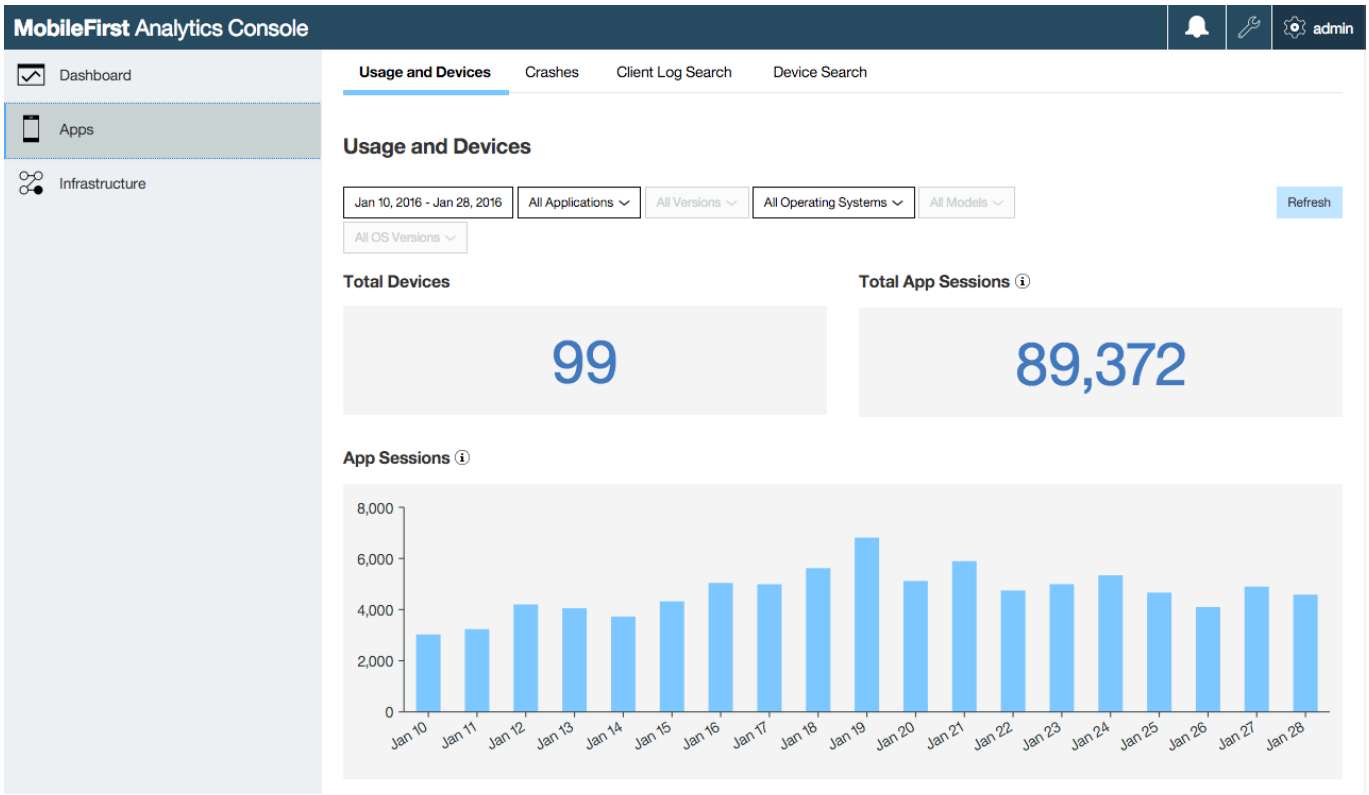
The Analytics API gives the developer the freedom to enable and disable collecting Analytics for the event they want to visualize on their Analytics Console.

To build Cordova applications, the Analytics API does not have methods to enable or disable collection on `LIFECYCLE` or `NETWORK` events. In other words, Cordova applications come with `LIFECYCLE` and `NETWORK` events pre-enabled by default. If you want to disable these events, follow the Client Lifecycle Events and Client Network Events on disabling events.

Client Lifecycle Events

After the Analytics SDK is configured, app sessions 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 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 can see the analytics console populated with data, as shown below.



You can enable or disable the collecting of app sessions by using the following API:

JavaScript

Web

To use client lifecycle events, initialize analytics:

```
ibmmfpfanalytics.logger.config({analyticsCapture: true});
```

Cordova

- For the iOS platform:
  - Open the **[Cordova application root folder] → platforms → ios → Classes → AppDelegate.m** file
  - Follow the iOS guide below to enable or disable `LIFECYCLE` activities.
  - Build the Cordova project by running the command: `cordova build`
- For the Android platform:
  - Open the **[Cordova appilcation root folder] → platforms → android → src → com → sample → [app-name] → MainActivity.java**
  - Look for the `onCreate` method and follow the Android guide below to enable or disable `LIFECYCLE` activities.
  - Build the Cordova project by running the command: `cordova build`

Android

To enable client lifecycle event logging:

```
WLAnalytics.addDeviceEventListener(DeviceEvent.LIFECYCLE);
```

To disable client lifecycle event logging:

```
WLAnalytics.removeDeviceEventListener(DeviceEvent.LIFECYCLE);
```

iOS

To enable client lifecycle event logging:

Objective-C:

```
[[WLAnalytics sharedInstance] addDeviceEventListener:LIFECYCLE];
```

Swift:

```
WLAnalytics.sharedInstance().addDeviceEventListener(LIFECYCLE);
```

To disable client lifecycle event logging:

Objective-C:

```
[[WLAnalytics sharedInstance] removeDeviceEventListener:LIFECYCLE];
```

Swift:

```
WLAnalytics.sharedInstance().removeDeviceEventListener(NETWORK);
```

Client Network Activities

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

- The client collects information such as roundtrip time and payload size when you start collecting on the `Network` device event.
- The server collects back-end information such as server processing time, adapter usage, used procedures.

Because the client and the server each collect their own information, charts do not display data until the client is configured to do so. To configure your client, you need to start collecting for the `NETWORK` device event.

JavaScript

Web
To use client network events, initialize analytics:

```
ibmmfpfanalytics.logger.config({analyticsCapture: true});
```

Cordova

- For the iOS platform:
  - Open the **[Cordova appilcation root folder] → platforms → ios → Classes → AppDelegate.m** file.
  - Follow the iOS guide below to enable or disable `NETWORK` activities.
  - Build the Cordova project by running the command: `cordova build`
- For the Android platform: navigate to the subactivity of the main activity to disable.
  - Open the **[Cordova appilcation root folder] → platforms → ios → src → com → sample → [app-name] → MainActivity.java** file.
  - Look for the `onCreate` method and follow the Android guide below to enable or disable `NETWORK` activities.
  - Build the Cordova project by running the command: `cordova build`

iOS

To enable client network-event logging:

Objective-C:

```
[[WLAnalytics sharedInstance] addDeviceEventListener:NETWORK];
```

Swift:

```
WLAnalytics.sharedInstance().addDeviceEventListener(NETWORK);
```

To disable client network-event logging:

Objective-C:

```
[[WLAnalytics sharedInstance] removeDeviceEventListener:NETWORK];
```

Swift:

```
WLAnalytics.sharedInstance().removeDeviceEventListener(NETWORK);
```

Android

To enable client network-event logging:

```
WLAnalytics.addDeviceEventListener(DeviceEvent.NETWORK);
```

To disable client network-event logging:

```
WLAnalytics.removeDeviceEventListener(DeviceEvent.NETWORK);
```

Custom Events

Use the following API methods to create custom events.

JavaScript (Cordova)

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

JavaScript (Web)

Depending on how you have referenced the Web SDK, you use either wlanalytics ```javascript

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

Android

After setting the first two configurations, you can start to log data as in this example:

```
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();
```

iOS

After importing WLAnalytics, you can now use the API to collect custom data, as follows:

Objective-C:

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

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

Swift:

```
let metadata: [NSObject: AnyObject] = ["foo": "bar"];
WLAnalytics.sharedInstance().log("hello", withMetadata: metadata);
WLAnalytics.sharedInstance().send();
```

Tracking Users

To track individual users, use the `setUserContext` method:

Cordova

Not supported

Web applications

```
ibmmfpfanalytics.setUserContext(user);
```

iOS

Objective-C

```
[[WLAnalytics sharedInstance] setUserContext:@"John Doe"];
```

Swift

```
WLAnalytics.sharedInstance().setUserContext("John Doe")
```

Android

```
WLAnalytics.setUserContext("John Doe");
```

To un-track individual users, use the `unsetUserContext` method:

Cordova

Not supported

Web applications

There is no `unsetUserContext` in the MobileFirst Web SDK. The user session ends after 30 minutes of inactivity, unless another call is made to `ibmmfpfanalytics.setUserContext(user)`.

iOS

Objective-C

```
[[WLAnalytics sharedInstance] unsetUserContext];
```

Swift

```
WLAnalytics.sharedInstance().unsetUserContext
```

Android

```
WLAnalytics.unsetUserContext();
```