Working offline

Working in offline mode

By using IBM MobileFirst Platform Foundation, it is possible to detect application connectivity failures and determine a course of action.
Application going offline and online can be detected in two ways:

- Explicitly, on invoking server-based procedures
- Implicitly, by using JavaScript event listeners

It is possible to define custom application behavior for offline and online status.
The developer is responsible for maintaining the offline or online state within the application, and ensuring that the application can recover from failed attempts to connect to the server.

For example, before the application logs in a new user or accesses the server under a new user, the application must ensure that a successful logout was received by the server.

Active detection

Using methods

Connectivity loss can be detected in two locations in the application code:

- Application initialization – `WL.Client.init()` method, typically called from `initOptions.js` file
- Adapter procedure invocation – `WL.Client.invokeProcedure()` method

To add connectivity failure detection in either location, add the `onConnectionFailure` property and specify a callback function to invoked if connectivity fails.

initOptions.js

```
var wlInitOptions = {
    onConnectionFailure: function (data)
{
    connectionFailure(data);
}
```

Implementation JS file

```
WL.Client.invokeProcedure(invocationData, {
    onSuccess: successHandlerFunction,
    onConnectionFailure: connectionFailure,
    timeout: 1000
});
```

Passive detection

Offline and online events

Each time the MobileFirst framework attempts to access the MobileFirst Server, it might detect that the application switched from offline to online status or vice versa.

In both cases, JavaScript events are fired:

- `WL.Events.WORKLIGHT_IS_DISCONNECTED` event is fired when connectivity to the MobileFirst Server fails
- `WL.Events.WORKLIGHT_IS_CONNECTED` event is fired when connectivity to the MobileFirst Server is restored

The developer can also add event listeners to the above events and specify the callback functions to handle them.

```
document.addEventListener(WL.Events.WORKLIGHT_IS_CONNECTED, connectDetected, false);
document.addEventListener(WL.Events.WORKLIGHT_IS_DISCONNECTED, disconnectDetected, fals
e);
```

Note: `WL.Events.WORKLIGHT_IS_DISCONNECTED` and `WL.Events.WORKLIGHT_IS_CONNECTED` are namespace constants, not strings.

Additional methods

More methods are provided by the MobileFirst framework to simplify online and offline development:

- `WL.Client.connect (options)` – attempt to establish a connection to the MobileFirst Server and return to online mode. options is an object that contains the following keys:
  - `onSuccess` – Callback function to invoke when server connection is established
  - `onFailure` – Callback function to invoke when server connection fails
  - `timeout` – Number of milliseconds to wait for the server response before failing with request timeout
- `WL.Device.getNetworkInfo()` - this method is available for the Android and iOS environments. A callback must be specified as a function parameter. The callback receives an object with the following properties:
  - `isAirplaneMode` – true/false
  - `carrierName` – string (for example, AT&T or VERIZON)
  - `telephonyNetworkType` – string (for example, UMTS or GPRS)
  - `isRoaming` – true/false
  - `networkConnectionType` – mobile/WiFi
  - `ipAddress` – string
  - `isNetworkConnected` – true/false

Foreground event

When a MobileFirst application returns to the foreground, the Cordova `resume` event is fired.
The developer can add a listener for this event and specify the callback function that handles it. For example:

```
document.addEventListener("resume", function() {
    WL.Device.getNetworkInfo( function(networkInfo)
{
        if (networkInfo.isNetworkConnected) {
            // Perform client or server actions.
        }
    });
}, false);
```

Heartbeat

The heartbeat pings the server at specified intervals to verify connectivity.
The heartbeat can used to periodically make sure that the application remains connected to the server.

Both `WL.Events.WORKLIGHT_IS_CONNECTED` and `WL.Events.WORKLIGHT_IS_DISCONNECTED` events can be fired by the heartbeat in designated cases.

A developer can specify the heartbeat interval by using the `WL.Client.setHeartBeatInterval(intervalSeconds)` API method.

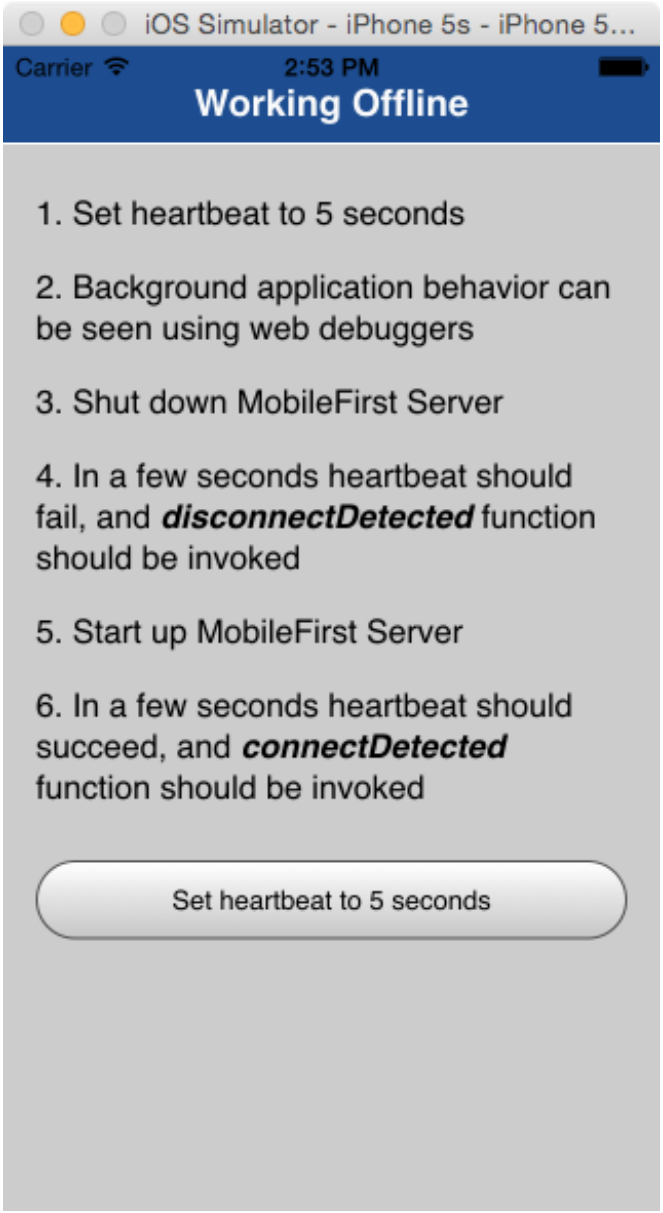
The following sample shows an offline and online detection mechanism

```
document.addEventListener(WL.Events.WORKLIGHT_IS_DISCONNECTED,
MyApp.connectionFailure, false);
MyApp.connectionFailure = function() {
    WL.Client.connect({
        onSuccess: function() {
            WL.Logger.debug("online");
            MyApp.onlineRestored();
        },<
        onFailure: function() {
            WL.Logger.debug("Still offline... Trying to connect again in 5
            seconds.");
            window.setTimeout(MyApp.connectionFailure, 5000);
        }
    });
};
```

1. An event listener for a `WL.Events.WORKLIGHT_IS_DISCONNECTED` event is added to the document. `MyApp.connectionFailure()` is invoked when the event fires
2. `WL.Client.connect()` tries to establish a server connection
3. If connection is successfully established, `MyApp.onlineRestored()` is invoked
4. If connection fails, a timeout is set for 5 seconds to invoke `MyApp.connectionFailure()` again

Sample application

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


iOS Simulator - iPhone 5s - iPhone 5...
Carrier
2:53 PM
Working Offline
1. Set heartbeat to 5 seconds
2. Background application behavior can be seen using web debuggers
3. Shut down MobileFirst Server
4. In a few seconds heartbeat should fail, and disconnectDetected function should be invoked
5. Start up MobileFirst Server
6. In a few seconds heartbeat should succeed, and connectDetected function should be invoked
Set heartbeat to 5 seconds