Migrating push notifications from event source-based notifications

Overview

From IBM MobileFirst Foundation v8.0, the event source-based model is not supported, and push notifications capability is enabled entirely by the push service model. For existing event source-based applications on earlier versions of MobileFirst to be moved to v8.0, they must be migrated to the new push service model.

During migration, keep in mind that it is not about using one API instead of another, but more about using one model/approach versus another.

For example, in the event source-based model, if you were to segment your mobile application users to send notifications to specific segments, you would model every segment as a distinct event source. In the push service model, you would achieve the same by defining tags that represents segments and have users subscribe to the respective tags. Tag-based notifications is a replacement to event source-based notifications.

Jump to

The table below provides you with a comparison between the two models.

User requirement	Event source model	Push service model
To enable your application with push notifications	• Create an Event Source Adapter and within it create an EventSource. • Configure or setup your application with push credentials.	Configure or setup your application with push credentials.
To enable your mobile client application with push notifications	• Create WLClient • Connect to the MobileFirst Server • Get an instance of push client • Subscribe to the Event source	• Instantiate push client • Initialize push client • Register the mobile device
To enable your mobile client application for notifications based on specific tags	Not supported.	Subscribe to the tag (that uses tag name) that is of interest.
To receive and handle notifications in your mobile client applications	Register a listener implementation.	Register a listener implementation.

To send push notifications to mobile client applications	• Implement adapter procedures that internally call the WL.Server APIs to send push notifications.
	• WL Server APIs provide means to send notifications:
	  ◦ By user
	  ◦ By device
	  ◦
	  ◦ Broadcasts (all devices)
	• Backend server applications can then invoke the adapter procedures to trigger push notification as part of their application logic.
		• Backend server applications can directly call the messages REST API. However, these applications must register as confidential client with the MobileFirst Server and obtain a valid OAuth access token that must be passed in the Authorization header of the REST API.
		• The REST API provides options to send notifications:
		  ◦ By user
		  ◦ By device
		  ◦ By platform
		  ◦ By tags
		  ◦ Broadcasts (all devices)

To trigger push notifications as regular time periods (polling intervals)	Implement the function to send push notifications within the event-source adapter and this as part of the createEventSource function call.	Not supported.
To register a hook with the name, URL, and the even types.	Implement hooks on the path of a device subscribing or unsubscribing to push notifications.	Not supported.

Migration Scenarios

Starting from IBM MobileFirst Foundation v8.0, the event source-based model will not be supported and push notifications capability will be enabled on IBM MobileFirst Platform Foundation entirely by the push service model, which is a more simple and agile alternative to event source model.

Existing event source-based applications on earlier versions of IBM MobileFirst Platform Foundation need to be migrated to v8.0, to the new push service model.

Jump to

- Native Android applications
- Native iOS applications
- Native Windows Universal applications

Hybrid applications

Examples of migration scenarios cover applications that use a single event sources or multiple sources, broadcast or Unicast notification, or tag notification.

Scenario 1: Existing applications using single event source in their application

Applications have used single event source over the earlier versions of MobileFirst as it supported push only through event source-based model.

Client

To migrate this in V8.0.0, convert this model to Unicast notification.

1. Initialize the MobileFirst push client instance in your application and in the success callback register the callback method that should receive the notification.

```
MFPPush.initialize(function(successResponse){
MFPPush.registerNotificationsCallback(notificationReceived); },
function(failureResponse){alert("Failed to initialize");
                    }
);
```

2. Implement the notification callback method.

```
var notificationReceived = function(message) {
   alert(JSON.stringify(message));
};
```

3. Register the mobile device with the push notification service.

```
MFPPush.registerDevice(function(successResponse) {
   alert("Successfully registered");
   },
 function(failureResponse) {
   alert("Failed to register");
   }
);
```

4. (Optional) Un-register the mobile device from the push notification service.

```
MFPPush.unregisterDevice(function(successResponse) {
   alert("Successfully unregistered");
   },
 function(failureResponse) {
   alert("Failed to unregister");
   }
);
```

5. Remove WL.Client.Push.isPushSupported() (if used) and use.

```
MFPPush.isPushSupported (function(successResponse) {
   alert(successResponse);
  },
  function(failureResponse) {
     alert("Failed to get the push suport status");
  }
);
```

6. Remove the following `WL.Client.Push` APIs, since there will be no event source to subscribe to and register notification callbacks.
   - `registerEventSourceCallback()`
   - `subscribe()`
   - `unsubscribe()`
   - `isSubscribed()`
   - `onReadyToSubscribe()`

Server

1. Remove the following WL.Server APIs (if used), in your adapter:
   - `notifyAllDevices()`
   - `notifyDevice()`
   - `notifyDeviceSubscription()`
   - `createEventSource()`
2. Complete the following steps for every application that was using the same event source:
   1. Set up the credentials by using the MobileFirst Operations Console. See Configuring push notification settings (../../notifications/sending-notifications). You can also set up the credentials by using Update GCM settings (PUT) (http://www.ibm.com/support/knowledgecenter/en/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/apiref/r_restapi_update_gcm_settings_put.html?view=kc#Update-GCM-settings--PUT-) REST API, for Android applications or Update APNs settings (PUT) (http://www.ibm.com/support/knowledgecenter/en/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/apiref/r_restapi_update_apns_settings_put.html?view=kc#Update-APNs-settings--PUT-) REST API, for iOS applications.

2. Add the scope `push.mobileclient` in **Scope Elements Mapping**.
3. Create tags to enable push notifications to be sent to subscribers. See Defining tags (../../notifications/sending-notifications/#defining-tags) for push notification.
4. You can use either of the following methods to send notifications:
    - The MobileFirst Operations Console. See Sending push notifications to subscribers (../../notifications/sending-notifications/#sending-notifications).
    - The Push Message (POST) (http://www.ibm.com/support/knowledgecenter/en/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/rest_runtime/r_restapi_push_message_post.html?view=kc#Push-Message--POST-) REST API with `userId`/`deviceId`.

Scenario 2: Existing applications using multiple event sources in their application

Applications using multiple event sources requires segmentation of users based on subscriptions.

Client

This maps to tags which segments the users/devices based on topic of interest. To migrate this, this model can be converted to tag-based notification.

1. Initialize the MFPPush client instance in your application and in the success callback register the callback method that should receive the notification.

```
MFPPush.initialize(function(successResponse){
   MFPPush.registerNotificationsCallback(notificationReceived);                    },
   function(failureResponse){
      alert("Failed to initialize");
   }
);
```

2. Implement the notification callback method.

```
var notificationReceived = function(message) {
   alert(JSON.stringify(message));
};
```

3. Register the mobile device with the push notification service.

```
MFPPush.registerDevice(function(successResponse) {
   alert("Successfully registered");
   },
 function(failureResponse) {
   alert("Failed to register");
   }
);
```

4. (Optional) Unregister the mobile device from the push notification service.

```
MFPPush.unregisterDevice(function(successResponse) {
   alert("Successfully unregistered");
   },
 function(failureResponse) {
   alert("Failed to unregister");
   }
);
```

5. Remove `WL.Client.Push.isPushSupported()` (if used) and use.

```
MFPPush.isPushSupported (function(successResponse) {
   alert(successResponse);
   },
 function(failureResponse) {
   alert("Failed to get the push suport status");
   }
);
```

6. Remove the following `WL.Client.Push` APIs since there will be no event source to subscribe to and register notification callbacks.
    - `registerEventSourceCallback()`
    - `subscribe()`
    - `unsubscribe()`
    - `isSubscribed()`
    - `onReadyToSubscribe()`
7. Subscribe to tags.

```
var tags = ['sample-tag1','sample-tag2'];
MFPPush.subscribe(tags, function(successResponse) {
   alert("Successfully subscribed");
   },
 function(failureResponse) {
   alert("Failed to subscribe");
   }
);
```

8. (Optional) Unsubscribe from tags.

```
MFPPush.unsubscribe(tags, function(successResponse) {
   alert("Successfully unsubscribed");
   },
 function(failureResponse) {
   alert("Failed to unsubscribe");
   }
);
```

Server

Remove the following `WL.Server` APIs (if used) in your adapter:

- `notifyAllDevices()`
- `notifyDevice()`
- `notifyDeviceSubscription()`
- `createEventSource()`

Complete the following steps for every application that was using the same event source:

1. Set up the credentials by using the MobileFirst Operations Console. See Configuring push notification settings (../../notifications/sending-notifications). You can also set up the credentials by using Update GCM settings (PUT) (http://www.ibm.com/support/knowledgecenter/en/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/apiref/r_restapi_update_gcm_settings_put.html?view=kc#Update-GCM-settings--PUT-) REST API, for Android applications or Update APNs settings (PUT) (http://www.ibm.com/support/knowledgecenter/en/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/apiref/r_restapi_update_apns_settings_put.html?view=kc#Update-APNs-settings--PUT-) REST API, for iOS applications.
2. Add the scope `push.mobileclient` in **Scope Elements Mapping**.
3. Create tags to enable push notifications to be sent to subscribers. See Defining tags (../../notifications/sending-notifications/#defining-tags) for push notification.
4. You can use either of the following methods to send notifications:
   - The MobileFirst Operations Console. See Sending push notifications to subscribers (../../notifications/sending-notifications/#sending-notifications).
   - The Push Message (POST) (http://www.ibm.com/support/knowledgecenter/en/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/rest_runtime/r_restapi_push_message_post.html?view=kc#Push-Message--POST-) REST API with `userId`/`deviceId`.

Scenario 3: Existing applications using broadcast/Unicast notification in their application

Client

1. Initialize the MFPPush client instance in your application and in the success callback register the callback method that should receive the notification.

```
MFPPush.initialize(function(successResponse){
   MFPPush.registerNotificationsCallback(notificationReceived);                    },
   function(failureResponse){
      alert("Failed to initialize");
   }
);
```

2. Implement the notification callback method.

```
var notificationReceived = function(message) {
   alert(JSON.stringify(message));
};
```

3. Register the mobile device with the push notification service.

```
MFPPush.registerDevice(function(successResponse) {
   alert("Successfully registered");
   },
 function(failureResponse) {
   alert("Failed to register");
   }
);
```

4. (Optional) Unregister the mobile device from the push notification service.

```
MFPPush.unregisterDevice(function(successResponse) {
   alert("Successfully unregistered");
   },
 function(failureResponse) {
   alert("Failed to unregister");
   }
);
```

5. Remove WL.Client.Push.isPushSupported() (if used) and use.

```
MFPPush.isPushSupported (function(successResponse) {
   alert(successResponse);
   },
 function(failureResponse) {
   alert("Failed to get the push suport status");
   }
);
```

6. Remove the following `WL.Client.Push` APIs:
   - `onReadyToSubscribe()`
   - `onMessage()`

Server

Remove `WL.Server.sendMessage()` (if used) in your adapter.
Complete the following steps for every application that was using the same event source:

1. Set up the credentials by using the MobileFirst Operations Console. See Configuring push notification settings (../../notifications/sending-notifications). You can also set up the credentials by using Update GCM settings (PUT) (http://www.ibm.com/support/knowledgecenter/en/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/apiref/r_restapi_update_gcm_settings_put.html?view=kc#Update-GCM-settings--PUT-) REST API, for Android applications or Update APNs settings (PUT) (http://www.ibm.com/support/knowledgecenter/en/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/apiref/r_restapi_update_apns_settings_put.html?view=kc#Update-APNs-settings--PUT-) REST API, for iOS applications.
2. Add the scope `push.mobileclient` in **Scope Elements Mapping**.
3. Create tags to enable push notifications to be sent to subscribers. See Defining tags (../../notifications/sending-notifications/#defining-tags) for push notification.
4. You can use either of the following methods to send notifications:
   - The MobileFirst Operations Console. See Sending push notifications to subscribers (../../notifications/sending-notifications/#sending-notifications).
   - The Push Message (POST) (http://www.ibm.com/support/knowledgecenter/en/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/rest_runtime/r_restapi_push_message_post.html?view=kc#Push-Message--POST-) REST API with `userId`/`deviceId`.

Scenario 4: Existing applications using tag notifications in their application

Client

1. Initialize the MFPPush client instance in your application and in the success callback register the callback method that should receive the notification.

```
MFPPush.initialize(function(successResponse){
   MFPPush.registerNotificationsCallback(notificationReceived);                    },
   function(failureResponse){
      alert("Failed to initialize");
   }
);
```

2. Implement the notification callback method.

```
var notificationReceived = function(message) {
   alert(JSON.stringify(message));
};
```

3. Register the mobile device with the push notification service.

```
MFPPush.registerDevice(function(successResponse) {
   alert("Successfully registered");
   },
 function(failureResponse) {
   alert("Failed to register");
   }
);
```

4. (Optional) Un-register the mobile device from push notification service.

```
MFPPush.unregisterDevice(function(successResponse) {
   alert("Successfully unregistered");
   },
 function(failureResponse) {
   alert("Failed to unregister");
   }
);
```

5. Remove `WL.Client.Push.isPushSupported()` (if used) and use:

```
MFPPush.isPushSupported (function(successResponse) {
   alert(successResponse);
   },
 function(failureResponse) {
   alert("Failed to get the push suport status");
   }
);
```

6. Remove the following `WL.Client.Push` APIs:
    - `subscribeTag()`
    - `unsubscribeTag()`
    - `isTagSubscribed()`
    - `onReadyToSubscribe()`
    - `onMessage()`
7. Subscribe to tags:

```
var tags = ['sample-tag1','sample-tag2'];
MFPPush.subscribe(tags, function(successResponse) {
   alert("Successfully subscribed");
   },
 function(failureResponse) {
   alert("Failed to subscribe");
   }
);
```

8. (Optional) Unsubscribe from tags:

```
MFPPush.unsubscribe(tags, function(successResponse) {
   alert("Successfully unsubscribed");
   },
 function(failureResponse) {
   alert("Failed to unsubscribe");
   }
);
```

Server

Remove `WL.Server.sendMessage()` (if used) in your adapter.
Complete the following steps for every application that was using the same event source:

1. Set up the credentials by using the MobileFirst Operations Console. See Configuring push notification settings (../../notifications/sending-notifications). You can also set up the credentials by using Update GCM settings (PUT) (http://www.ibm.com/support/knowledgecenter/en/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/apiref/r_restapi_update_gcm_settings_put.html?view=kc#Update-GCM-settings--PUT-) REST API, for Android applications or Update APNs settings (PUT) (http://www.ibm.com/support/knowledgecenter/en/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/apiref/r_restapi_update_apns_settings_put.html?view=kc#Update-APNs-settings--PUT-) REST API, for iOS applications.
2. Add the scope `push.mobileclient` in **Scope Elements Mapping**.
3. Create tags to enable push notifications to be sent to subscribers. See Defining tags (../../notifications/sending-notifications/#defining-tags) for push notification.
4. You can use either of the following methods to send notifications:
    - The MobileFirst Operations Console. See Sending push notifications to subscribers (../../notifications/sending-notifications/#sending-notifications).
    - The Push Message (POST) (http://www.ibm.com/support/knowledgecenter/en/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/rest_runtime/r_restapi_push_message_post.html?view=kc#Push-Message--POST-) REST API with `userId`/`deviceId`.

Native Android applications

Examples of migration scenarios cover applications that use a single event sources or multiple sources, broadcast or Unicast notification, or tag notification.

Scenario 1: Existing applications using single event source in their application

Applications have used single event source over the earlier versions of MobileFirst as it supported push only through event source-based model.

Client

To migrate this in v8.0, convert this model to Unicast notification.

1. Initialize the MFPPush client instance in your application.

```
MFPPush push = MFPPush.getInstance();
   push.initialize(_this);
```

2. Implement the interface MFPPushNotificationListener and define onReceive().

```
@Override
public void onReceive(MFPSimplePushNotification message) {
   Log.i("Push Notifications", message.getAlert());
}
```

3. Register the mobile device with the push notification service.

```
push.registerDevice(new MFPPushResponseListener<String>(){
   @Override
   public void onFailure(MFPPushException arg0) {
      Log.i("Push Notifications", "Failed to register");
   }
   @Override
   public void onSuccess(String arg0) {
     Log.i("Push Notifications", "Registered successfully");

   }
});
```

4. (Optional) Un-register the mobile device from the push notification service.

```
push.unregisterDevice(new MFPPushResponseListener<String>(){
   @Override
   public void onFailure(MFPPushException arg0) {
      Log.i("Push Notifications", "Failed to unregister");

   }
   @Override
   public void onSuccess(String arg0) {
       Log.i("Push Notifications", "Unregistered successfully");
   }
});
```

5. Remove `WLClient.Push.isPushSupported()` (if used) and use `push.isPushSupported();`.
6. Remove the following `WLClient.Push` APIs since there will be no event source to subscribe to and register notification callbacks:
   - `registerEventSourceCallback()`
   - `subscribe()`
   - `unsubscribe()`
   - `isSubscribed()`
   - `WLOnReadyToSubscribeListener` and `WLNotificationListener` implementation

Server

Remove the following `WL.Server` APIs (if used) in your adapter:

- `notifyAllDevices()`
- `notifyDevice()`
- `notifyDeviceSubscription()`
- `createEventSource()`

Complete the following steps for every application that was using the same event source:

1. Set up the credentials by using the MobileFirst Operations Console. See Configuring push notification settings (../../notifications/sending-notifications). You can also set up the credentials by using Update GCM settings (PUT) (http://www.ibm.com/support/knowledgecenter/en/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/apiref/r_restapi_update_gcm_settings_put.html?view=kc#Update-GCM-settings--PUT-) REST API, for Android applications or Update APNs settings (PUT) (http://www.ibm.com/support/knowledgecenter/en/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/apiref/r_restapi_update_apns_settings_put.html?view=kc#Update-APNs-settings--PUT-) REST API, for iOS applications.
2. Add the scope `push.mobileclient` in **Scope Elements Mapping**.
3. Create tags to enable push notifications to be sent to subscribers. See Defining tags (../../notifications/sending-notifications/#defining-tags) for push notification.
4. You can use either of the following methods to send notifications:
   - The MobileFirst Operations Console. See Sending push notifications to subscribers (../../notifications/sending-notifications/#sending-notifications).
   - The Push Message (POST) (http://www.ibm.com/support/knowledgecenter/en/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/rest_runtime/r_restapi_push_message_post.html?view=kc#Push-Message--POST-) REST API with `userId`/`deviceId`.

Scenario 2: Existing applications using multiple event sources in their application

Applications using multiple event sources requires the segmentation of users based on the subscriptions.

Client

This maps to tags which segments the users/devices based on topic of interest. To migrate this in MobileFirst V8.0.0, convert this model to tag based notification.

1. Initialize the `MFPPush` client instance in your application:

```
MFPPush push = MFPPush.getInstance();
push.initialize(_this);
```

2. Implement the interface MFPPushNotificationListener and define onReceive().

```
@Override
public void onReceive(MFPSimplePushNotification message) {
   Log.i("Push Notifications", message.getAlert());

}
```

3. Register the mobile device with the push notification service.

```
push.registerDevice(new MFPPushResponseListener<String>(){
   @Override
   public void onFailure(MFPPushException arg0) {
      Log.i("Push Notifications", "Failed to register");
   }
   @Override
   public void onSuccess(String arg0) {
      Log.i("Push Notifications", "Registered successfully");
   }
});
```

4. (Optional) Un-register the mobile device from the push notification service:

```
push.unregisterDevice(new MFPPushResponseListener<String>(){
  @Override
   public void onFailure(MFPPushException arg0) {
      Log.i("Push Notifications", "Failed to unregister");

   }
   @Override
   public void onSuccess(String arg0) {
      Log.i( "Push Notifications", "Unregistered successfully");
   }
});
```

5. Remove `WLClient.Push.isPushSupported()` (if used) and use `push.isPushSupported();`.
6. Remove the following `WLClient.Push` APIs since there will be no event source to subscribe to and register notification callbacks:
   - `registerEventSourceCallback()`
   - `subscribe()`
   - `unsubscribe()`
   - `isSubscribed()`
7. `WLOnReadyToSubscribeListener` and `WLNotificationListener` Implementation
8. Subscribe to tags:

```
String[] tags = new String[2];
tags[0] ="sample-tag1";
tags[1] ="sample-tag2";
push.subscribe(tags, new MFPPushResponseListener<String[]>(){

   @Override
   public void onFailure(MFPPushException arg0) {
      Log.i("Failed to subscribe");
   }

   @Override
   public void onSuccess(String[] arg0) {
      Log.i( "Subscribed successfully");
   }
});
```

9. (Optional) Unsubscribe from tags:

```
String[] tags = new String[2];
tags[0] ="sample-tag1";
tags[1] ="sample-tag2";
push.unsubscribe(tags, new MFPPushResponseListener<String[]>(){

   @Override
   public void onFailure(MFPPushException arg0) {
      Log.i("Push Notifications", "Failed to unsubscribe");
   }

   @Override
   public void onSuccess(String[] arg0) {
      Log.i("Push Notifications", "Unsubscribed successfully");
   }
});
```

Server

Remove the following `WL.Server` APIs (if used) in your adapter:

- `notifyAllDevices()`
- `notifyDevice()`
- `notifyDeviceSubscription()`
- `createEventSource()`

Complete the following steps for every application that was using the same event source:

1. Set up the credentials by using the MobileFirst Operations Console. See Configuring push notification settings (../../notifications/sending-notifications).

You can also set up the credentials by using Update GCM settings (PUT) (http://www.ibm.com/support/knowledgecenter/en/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/apiref/r_restapi_update_gcm_settings_put.html?view=kc#Update-GCM-settings--PUT-) REST API, for Android applications or Update APNs settings (PUT) (http://www.ibm.com/support/knowledgecenter/en/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/apiref/r_restapi_update_apns_settings_put.html?view=kc#Update-APNs-settings--PUT-) REST API, for iOS applications.

2. Add the scope `push.mobileclient` in **Scope Elements Mapping**.
3. Create tags to enable push notifications to be sent to subscribers. See Defining tags (../../notifications/sending-notifications/#defining-tags) for push notification.
4. You can use either of the following methods to send notifications:
   - The MobileFirst Operations Console. See Sending push notifications to subscribers (../../notifications/sending-notifications/#sending-notifications).
   - The Push Message (POST) (http://www.ibm.com/support/knowledgecenter/en/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/rest_runtime/r_restapi_push_message_post.html?view=kc#Push-Message--POST-) REST API with `userId`/`deviceId`.

Scenario 3: Existing applications using broadcast/Unicast notification in their application

Client

1. Initialize the `MFPPush` client instance in your application:

```
MFPPush push = MFPPush.getInstance();
push.initialize(_this);
```

2. Implement the interface `MFPPushNotificationListener` and define `onReceive()`.

```
@Override
public void onReceive(MFPSimplePushNotification message) {
    Log.i("Push Notifications", message.getAlert());
}
```

3. Register the mobile device with push notification service.

```
push.registerDevice(new MFPPushResponseListener<String>(){
    @Override
    public void onFailure(MFPPushException arg0) {
        Log.i("Push Notifications", "Failed to register");
    }
    @Override
    public void onSuccess(String arg0) {
        Log.i("Push Notifications", "Registered successfully");

    }
});
```

4. (Optional) Un-register the mobile device from push notification service.

```
push.unregisterDevice(new MFPPushResponseListener<String>(){
   @Override
    public void onFailure(MFPPushException arg0) {
        Log.i("Push Notifications", "Failed to unregister");

    }
    @Override
    public void onSuccess(String arg0) {
        Log.i( "Push Notifications", "Unregistered successfully");
    }
});
```

5. Remove `WLClient.Push.isPushSupported()` (if used) and use `push.isPushSupported();`.
6. Remove the following WLClient.Push APIs:
   - `registerEventSourceCallback()`
   - `WLOnReadyToSubscribeListener` and `WLNotificationListener` Implementation

Server

Remove WL.Server.sendMessage()` APIs (if used) in your adapter:

Complete the following steps for every application that was using the same event source:

1. Set up the credentials by using the MobileFirst Operations Console. See Configuring push notification settings (../../notifications/sending-notifications). You can also set up the credentials by using Update GCM settings (PUT) (http://www.ibm.com/support/knowledgecenter/en/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/apiref/r_restapi_update_gcm_settings_put.html?view=kc#Update-GCM-settings--PUT-) REST API, for Android applications or Update APNs settings (PUT) (http://www.ibm.com/support/knowledgecenter/en/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/apiref/r_restapi_update_apns_settings_put.html?view=kc#Update-APNs-settings--PUT-) REST API, for iOS applications.
2. Add the scope `push.mobileclient` in **Scope Elements Mapping**.
3. Create tags to enable push notifications to be sent to subscribers. See Defining tags (../../notifications/sending-notifications/#defining-tags) for push notification.
4. You can use either of the following methods to send notifications:

- The MobileFirst Operations Console. See Sending push notifications to subscribers (../../notifications/sending-notifications/#sending-notifications).
- The Push Message (POST) (http://www.ibm.com/support/knowledgecenter/en/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/rest_runtime/r_restapi_push_message_post.html?view=kc#Push-Message--POST-) REST API with `userId`/`deviceId`.

Scenario 4: Existing applications using tag notifications in their application

Client

1. Initialize the `MFPPush` client instance in your application:

```
MFPPush push = MFPPush.getInstance();
push.initialize(_this);
```

2. Implement the interface MFPPushNotificationListener and define onReceive().

```
@Override
public void onReceive(MFPSimplePushNotification message) {
    Log.i("Push Notifications", message.getAlert());
}
```

3. Register the mobile device with the push notification service.

```
push.registerDevice(new MFPPushResponseListener<String>(){
    @Override
    public void onFailure(MFPPushException arg0) {
        Log.i("Push Notifications", "Failed to register");
    }
    @Override
    public void onSuccess(String arg0) {
        Log.i("Push Notifications", "Registered successfully");
    }
});
```

4. (Optional) Un-register the mobile device from the push notification service.

```
push.unregisterDevice(new MFPPushResponseListener<String>(){
    @Override
    public void onFailure(MFPPushException arg0) {
        Log.i("Push Notifications", "Failed to unregister");

    }
    @Override
    public void onSuccess(String arg0) {
        Log.i( "Push Notifications", "Unregistered successfully");
    }
});
```

5. Remove `WLClient.Push.isPushSupported()` (if used) and use `push.isPushSupported()`;
6. Remove the following `WLClient.Push` API's:
    - `subscribeTag()`
    - `unsubscribeTag()`
    - `isTagSubscribed()`
    - `WLOnReadyToSubscribeListener` and `WLNotificationListener` Implementation
7. Subscribe to tags:

```
String[] tags = new String[2];
tags[0] ="sample-tag1";
tags[1] ="sample-tag2";
push.subscribe(tags, new MFPPushResponseListener<String[]>(){
    @Override
    public void onFailure(MFPPushException arg0) {
        Log.i("Failed to subscribe");
    }

    @Override
    public void onSuccess(String[] arg0) {
        Log.i( "Subscribed successfully");
    }
});
```

8. (Optional) Unsubscribe from tags:

```
String[] tags = new String[2];
tags[0] ="sample-tag1";
tags[1] ="sample-tag2";
push.unsubscribe(tags, new MFPPushResponseListener<String[]>(){
    @Override
    public void onFailure(MFPPushException arg0) {
        Log.i("Push Notifications", "Failed to unsubscribe");
    }

    @Override
    public void onSuccess(String[] arg0) {
        Log.i("Push Notifications", "Unsubscribed successfully");
    }
});
```

Server

Remove `WL.Server.sendMessage()` (if used) in your adapter.

Complete the following steps for every application that was using the same event source:

1. Set up the credentials by using the MobileFirst Operations Console. See Configuring push notification settings (../../notifications/sending-notifications). You can also set up the credentials by using Update GCM settings (PUT) (http://www.ibm.com/support/knowledgecenter/en/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/apiref/r_restapi_update_gcm_settings_put.html?view=kc#Update-GCM-settings--PUT-) REST API, for Android applications or Update APNs settings (PUT) (http://www.ibm.com/support/knowledgecenter/en/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/apiref/r_restapi_update_apns_settings_put.html?view=kc#Update-APNs-settings--PUT-) REST API, for iOS applications.
2. Add the scope `push.mobileclient` in **Scope Elements Mapping**.
3. Create tags to enable push notifications to be sent to subscribers. See Defining tags (../../notifications/sending-notifications/#defining-tags) for push notification.
4. You can use either of the following methods to send notifications:
    - The MobileFirst Operations Console. See Sending push notifications to subscribers (../../notifications/sending-notifications/#sending-notifications).
    - The Push Message (POST) (http://www.ibm.com/support/knowledgecenter/en/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/rest_runtime/r_restapi_push_message_post.html?view=kc#Push-Message--POST-) REST API with `userId`/`deviceId`.

Native iOS applications

Examples of migration scenarios cover applications that use a single event sources or multiple sources, broadcast or Unicast notification, or tag notification.

Scenario 1: Existing applications using single event source in their application

Applications have used single event source over the earlier versions of MobileFirst as it supported push only through event source-based model.

Client

To migrate this in v8.0, convert this model to Unicast notification.

1. Initialize the `MFPPush` client instance in your application.

```
[[MFPPush sharedInstance] initialize];
```

2. Implement the notification processing in the `didReceiveRemoteNotification()`.
3. Register the mobile device with the push notification service.

```
[[MFPPush sharedInstance] registerDevice:^(WLResponse *response, NSError *error) {
    if(error){
        NSLog(@"Failed to register");
    } else {
        NSLog(@"Successfullyregistered");
    }
}];
```

4. (Optional) Un-register the mobile device from the push notification service.

```
[MFPPush sharedInstance] unregisterDevice:^(WLResponse *response, NSError *error) {
    if(error){
        NSLog(@"Failed to unregister");
    } else {
        NSLog(@"Successfully unregistered");
    }
}];
```

5. Remove `WLClient.Push.isPushSupported()` (if used) and use:

```
[[MFPPush sharedInstance] isPushSupported]
```

6. Remove the following `WLClient.Push` API's since there will be no event source to subscribe to and register notification callbacks:
    - `registerEventSourceCallback()`
    - `subscribe()`

- `unsubscribe()`
- `isSubscribed()`
- `WLOnReadyToSubscribeListener` implementation

7. Call `sendDeviceToken()` in `didRegisterForRemoteNotificationsWithDeviceToken`.

```
[[MFPPush sharedInstance] sendDeviceToken:deviceToken];
```

Server

Remove the following WL.Server API's (if used) in your adapter:

- `notifyAllDevices()`
- `notifyDevice()`
- `notifyDeviceSubscription()`
- `createEventSource()`

Complete the following steps for every application that was using the same event source:

1. Set up the credentials by using the MobileFirst Operations Console. See Configuring push notification settings (../../notifications/sending-notifications). You can also set up the credentials by using Update GCM settings (PUT) (http://www.ibm.com/support/knowledgecenter/en/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/apiref/r_restapi_update_gcm_settings_put.html?view=kc#Update-GCM-settings--PUT-) REST API, for Android applications or Update APNs settings (PUT) (http://www.ibm.com/support/knowledgecenter/en/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/apiref/r_restapi_update_apns_settings_put.html?view=kc#Update-APNs-settings--PUT-) REST API, for iOS applications.
2. Add the scope `push.mobileclient` in **Scope Elements Mapping**.
3. Create tags to enable push notifications to be sent to subscribers. See Defining tags (../../notifications/sending-notifications/#defining-tags) for push notification.
4. You can use either of the following methods to send notifications:
   - The MobileFirst Operations Console. See Sending push notifications to subscribers (../../notifications/sending-notifications/#sending-notifications).
   - The Push Message (POST) (http://www.ibm.com/support/knowledgecenter/en/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/rest_runtime/r_restapi_push_message_post.html?view=kc#Push-Message--POST-) REST API with `userId`/`deviceId`.

Scenario 2: Existing applications using multiple event sources in their application

Applications using multiple event sources requires segmentation of users based on subscriptions.

Client

This maps to tags which segments the users/devices based on topic of interest. To migrate this to MobileFirstV8.0.0, convert this model to tag based notification.

1. Initialize the `MFPPush` client instance in your application.

```
[[MFPPush sharedInstance] initialize];
```

2. Implement the notification processing in the `didReceiveRemoteNotification()`.
3. Register the mobile device with the push notification service:

```
[[MFPPush sharedInstance] registerDevice:^(WLResponse *response, NSError *error) {
   if(error){
      NSLog(@"Failed to register");
   }else{
      NSLog(@"Successfullyregistered");
   }
}];
```

4. (Optional) Un-register the mobile device from the push notification service:

```
[MFPPush sharedInstance] unregisterDevice:^(WLResponse *response, NSError *error) {
   if(error){
      NSLog(@"Failed to unregister");
   }else{
      NSLog(@"Successfully unregistered");
   }
}];
```

5. Remove `WLClient.Push.isPushSupported()` (if used) and use:

```
[[MFPPush sharedInstance] isPushSupported]
```

6. Remove the following `WLClient.Push` API's since there will be no event source to subscribe to and register notification callbacks:
   - `registerEventSourceCallback()`
   - `subscribe()`
   - `unsubscribe()`
   - `isSubscribed()`
   - `WLOnReadyToSubscribeListener` Implementation
7. Call `sendDeviceToken()` in `didRegisterForRemoteNotificationsWithDeviceToken`.
8. Subscribe to tags:

```
NSMutableArray *tags = [[NSMutableArray alloc]init];
[tags addObject:@"sample-tag1"];
[tags addObject:@"sample-tag2"];
[MFPPush sharedInstance] subscribe:tags completionHandler:^(WLResponse *response, NSError *error) {
   if(error){
      NSLog(@"Failed to unregister");
   }else{
      NSLog(@"Successfully unregistered");
   }
}];
```

9. (Optional) Unsubscribe from tags:

```
NSMutableArray *tags = [[NSMutableArray alloc]init];
[tags addObject:@"sample-tag1"];
[tags addObject:@"sample-tag2"];
[MFPPush sharedInstance] unsubscribe:tags completionHandler:^(WLResponse *response, NSError *error) {
   if(error){
     NSLog(@"Failed to unregister");
   }else{
     NSLog(@"Successfully unregistered");
   }
}];
```

Server

Remove `WL.Server` (if used) in your adapter.

- `notifyAllDevices()`
- `notifyDevice()`
- `notifyDeviceSubscription()`
- `createEventSource()`

Complete the following steps for every application that was using the same event source:

1. Set up the credentials by using the MobileFirst Operations Console. See Configuring push notification settings (../../notifications/sending-notifications). You can also set up the credentials by using Update GCM settings (PUT) (http://www.ibm.com/support/knowledgecenter/en/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/apiref/r_restapi_update_gcm_settings_put.html?view=kc#Update-GCM-settings--PUT-) REST API, for Android applications or Update APNs settings (PUT) (http://www.ibm.com/support/knowledgecenter/en/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/apiref/r_restapi_update_apns_settings_put.html?view=kc#Update-APNs-settings--PUT-) REST API, for iOS applications.
2. Add the scope `push.mobileclient` in **Scope Elements Mapping**.
3. Create tags to enable push notifications to be sent to subscribers. See Defining tags (../../notifications/sending-notifications/#defining-tags) for push notification.
4. You can use either of the following methods to send notifications:
   - The MobileFirst Operations Console. See Sending push notifications to subscribers (../../notifications/sending-notifications/#sending-notifications).
   - The Push Message (POST) (http://www.ibm.com/support/knowledgecenter/en/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/rest_runtime/r_restapi_push_message_post.html?view=kc#Push-Message--POST-) REST API with `userId`/`deviceId`.

Scenario 3: Existing applications using broadcast/Unicast notification in their application

Client

1. Initialize the MFPPush client instance in your application:

```
[[MFPPush sharedInstance] initialize];
```

2. Implement the notification processing in the `didReceiveRemoteNotification()`.
3. Register the mobile device with the push notification service:

```
[[MFPPush sharedInstance] registerDevice:^(WLResponse *response, NSError *error) {
   if(error){
      NSLog(@"Failed to register");
   }else{
      NSLog(@"Successfullyregistered");
   }
}];
```

4. (Optional) Un-register the mobile device from the push notification service.

```
[MFPPush sharedInstance] unregisterDevice:^(WLResponse *response, NSError *error) {
   if(error){
      NSLog(@"Failed to unregister");
   }else{
      NSLog(@"Successfully unregistered");
   }
}];
```

5. Remove `WLClient.Push.isPushSupported()` (if used) and use:

```
[[MFPPush sharedInstance] isPushSupported]
```

6. Remove the following `WLClient.Push` API's:
   - `registerEventSourceCallback()`
   - `WLOnReadyToSubscribeListener` Implementation

Server

Remove `WL.Server.sendMessage` (if used) in your adapter.

Complete the following steps for every application that was using the same event source:

1. Set up the credentials by using the MobileFirst Operations Console. See Configuring push notification settings (../../notifications/sending-notifications). You can also set up the credentials by using Update GCM settings (PUT) (http://www.ibm.com/support/knowledgecenter/en/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/apiref/r_restapi_update_gcm_settings_put.html?view=kc#Update-GCM-settings--PUT-) REST API, for Android applications or Update APNs settings (PUT) (http://www.ibm.com/support/knowledgecenter/en/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/apiref/r_restapi_update_apns_settings_put.html?view=kc#Update-APNs-settings--PUT-) REST API, for iOS applications.
2. Add the scope `push.mobileclient` in **Scope Elements Mapping**.
3. Create tags to enable push notifications to be sent to subscribers. See Defining tags (../../notifications/sending-notifications/#defining-tags) for push notification.
4. You can use either of the following methods to send notifications:
   - The MobileFirst Operations Console. See Sending push notifications to subscribers (../../notifications/sending-notifications/#sending-notifications).
   - The Push Message (POST) (http://www.ibm.com/support/knowledgecenter/en/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/rest_runtime/r_restapi_push_message_post.html?view=kc#Push-Message--POST-) REST API with `userId`/`deviceId`.

Scenario 4: Existing applications using tag notifications in their application

Client

1. Initialize the MFPPush client instance in your application:

```
[[MFPPush sharedInstance] initialize];
```

2. Implement the notification processing in the `didReceiveRemoteNotification()`.
3. Register the mobile device with the push notification service:

```
[[MFPPush sharedInstance] registerDevice:^(WLResponse *response, NSError *error) {
   if(error){
      NSLog(@"Failed to register");
   }else{
      NSLog(@"Successfullyregistered");
   }
}];
```

4. (Optional) Un-register the mobile device from the push notification service:

```
[MFPPush sharedInstance] unregisterDevice:^(WLResponse *response, NSError *error) {
   if(error){
     NSLog(@"Failed to unregister");
   }else{
     NSLog(@"Successfully unregistered");
   }
}];
```

5. Remove `WLClient.Push.isPushSupported()` (if used) and use `[[MFPPush sharedInstance] isPushSupported]`.
6. Remove the following `WLClient.Push` API's since there will be no Event source to subscribe to and register notification callbacks:
   - `registerEventSourceCallback()`
   - `subscribeTag()`
   - `unsubscribeTag()`
   - `isTagSubscribed()`
   - `WLOnReadyToSubscribeListener` Implementation
7. Call `sendDeviceToken()` in `didRegisterForRemoteNotificationsWithDeviceToken`.
8. Subscribe to tags:

```
NSMutableArray *tags = [[NSMutableArray alloc]init];
[tags addObject:@"sample-tag1"];
[tags addObject:@"sample-tag2"];
[MFPPush sharedInstance] subscribe:tags completionHandler:^(WLResponse *response, NSError *error) {
   if(error){
     NSLog(@"Failed to unregister");
   }else{
     NSLog(@"Successfully unregistered");
   }
}];
```

9. (Optional) Unsubscribe from tags:

```
NSMutableArray *tags = [[NSMutableArray alloc]init];
[tags addObject:@"sample-tag1"];
[tags addObject:@"sample-tag2"];
[MFPPush sharedInstance] unsubscribe:tags completionHandler:^(WLResponse *response, NSError *error) {
   if(error){
      NSLog(@"Failed to unregister");
   }else{
      NSLog(@"Successfully unregistered");
   }
}];
```

Server

Remove the `WL.Server.sendMessage` (if used), in your adapter.

Complete the following steps for every application that was using the same event source:

1. Set up the credentials by using the MobileFirst Operations Console. See Configuring push notification settings (../../notifications/sending-notifications). You can also set up the credentials by using Update GCM settings (PUT) (http://www.ibm.com/support/knowledgecenter/en/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/apiref/r_restapi_update_gcm_settings_put.html?view=kc#Update-GCM-settings--PUT-) REST API, for Android applications or Update APNs settings (PUT) (http://www.ibm.com/support/knowledgecenter/en/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/apiref/r_restapi_update_apns_settings_put.html?view=kc#Update-APNs-settings--PUT-) REST API, for iOS applications.
2. Add the scope `push.mobileclient` in **Scope Elements Mapping**.
3. Create tags to enable push notifications to be sent to subscribers. See Defining tags (../../notifications/sending-notifications/#defining-tags) for push notification.
4. You can use either of the following methods to send notifications:
   - The MobileFirst Operations Console. See Sending push notifications to subscribers (../../notifications/sending-notifications/#sending-notifications).
   - The Push Message (POST) (http://www.ibm.com/support/knowledgecenter/en/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/rest_runtime/r_restapi_push_message_post.html?view=kc#Push-Message--POST-) REST API with `userId`/`deviceId`.

Native Windows Universal applications

Examples of migration scenarios cover applications that use a single event sources or multiple sources, broadcast or Unicast notification, or tag notification.

Scenario 1: Existing applications using single event source in their application

To migrate this in v8.0, convert this model to Unicast notification.

1. Initialize the `MFPPush` client instance in your application.

```
MFPPush push = MFPPush.GetInstance();
push.Initialize();
Implement the interface MFPPushNotificationListener and define onReceive().
class Pushlistener : MFPPushNotificationListener
{
   public void onReceive(String properties, String payload)
   {
         Debug.WriteLine("Push Notifications\n properties:" + properties + "\n payload:" + payload);
   }
}
```

2. Register the mobile device with the push notification service.

```
MFPPushMessageResponse Response = await push.RegisterDevice(null);
if (Response.Success == true)
{
   Debug.WriteLine("Push Notifications Registered successfully");
}
else
{
   Debug.WriteLine("Push Notifications Failed to register");
}
```

3. (Optional) Un-register the mobile device from the push notification service.

```
MFPPushMessageResponse Response = await push.UnregisterDevice();
if (Response.Success == true)
{
   Debug.WriteLine("Push Notifications Failed to unregister");
}
else
{
   Debug.WriteLine("Push Notifications Unregistered successfully");
}
```

4. Remove `WLClient.Push.IsPushSupported()` (if used) and use `push.IsPushSupported();`.
5. Remove the following `WLClient.Push` APIs since there will be no event source to subscribe to and register notification callbacks:

- `registerEventSourceCallback()`
- `subscribe()`
- `unsubscribe()`
- `isSubscribed()`
- `WLOnReadyToSubscribeListener` and `WLNotificationListener` implementation

Server

Remove the following `WL.Server` APIs (if used) in your adapter:

- `notifyAllDevices()`
- `notifyDevice()`
- `notifyDeviceSubscription()`
- `createEventSource()`

Complete the following steps for every application that was using the same event source:

1. Set up the WNS credentials in the **Push Settings** page of MobileFirst Operations Console or use WNS Settings REST API.
2. Add the scope `push.mobileclient` in **Map Scope Elements to security checks** section in the Security tab of MobileFirst Operations Console.
3. You can also use the Push Message (POST) (http://www.ibm.com/support/knowledgecenter/en/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/rest_runtime/r_restapi_push_message_post.html?view=kc#Push-Message--POST-) REST API with `userId`/`deviceId`, to send message.

Scenario 2: Existing applications using multiple event sources in their application

Applications using multiple event sources requires segmentation of users based on subscriptions.

Client

This maps to tags which segments the users/devices based on topic of interest. To migrate this in MobileFirst V8.0.0, convert this model to tag based notification.

1. Initialize the `MFPPush` client instance in your application:

```
MFPPush push = MFPPush.GetInstance();
push.Initialize();
Implement the interface MFPPushNotificationListener and define onReceive().
class Pushlistener : MFPPushNotificationListener
{
   public void onReceive(String properties, String payload)
   {
         Debug.WriteLine("Push Notifications\n properties:" + properties + "\n payload:" + payload);
   }
}
```

2. Register the mobile device with the IMFPUSH service.

```
MFPPushMessageResponse Response = await push.RegisterDevice(null);
if (Response.Success == true)
{
   Debug.WriteLine("Push Notifications Registered successfully");
}
else
{
   Debug.WriteLine("Push Notifications Failed to register");
}
```

3. (Optional) Un-register the mobile device from the IMFPUSH service:

```
MFPPushMessageResponse Response = await push.UnregisterDevice();
if (Response.Success == true)
{
   Debug.WriteLine("Push Notifications Failed to unregister");
}
else
{
   Debug.WriteLine("Push Notifications Unregistered successfully");
}
```

4. Remove `WLClient.Push.IsPushSupported()` (if used) and use `push.IsPushSupported();`.
5. Remove the following `WLClient.Push` APIs since there will be no Event Source to subscribe to and register notification callbacks:
   - `registerEventSourceCallback()`
   - `subscribe()`
   - `unsubscribe()`
   - `isSubscribed()`
   - `WLOnReadyToSubscribeListener` and `WLNotificationListener` implementation
6. Subscribe to tags:

```
String[] Tag = { "sample-tag1", "sample-tag2" };
MFPPushMessageResponse Response = await push.Subscribe(Tag);
if (Response.Success == true)
{
   Debug.WriteLine("Subscribed successfully");
}
else
{
   Debug.WriteLine("Failed to subscribe");
}
```

7. (Optional) Unsubscribe from tags:

```
String[] Tag = { "sample-tag1", "sample-tag2" };
MFPPushMessageResponse Response = await push.Unsubscribe(Tag);
if (Response.Success == true)
{
   Debug.WriteLine("Unsubscribed successfully");
}
else
{
   Debug.WriteLine("Failed to unsubscribe");
}
```

Server

Remove the following `WL.Server` APIs (if used) in your adapter:

- `notifyAllDevices()`
- `notifyDevice()`
- `notifyDeviceSubscription()`
- `createEventSource()`

Complete the following steps for every application that was using the same event source:

1. Set up the WNS credentials in the **Push Settings** page of MobileFirst Operations Console or use WNS Settings REST API.
2. Add the scope `push.mobileclient` in **Map Scope Elements to security checks** section in the **Security** tab of MobileFirst Operations Console.
3. Create Push tags in the **Tags** page of MobileFirst Operations Console.
4. You can also use the Push Message (POST) (http://www.ibm.com/support/knowledgecenter/en/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/rest_runtime/r_restapi_push_message_post.html?view=kc#Push-Message--POST-) REST API with `userId`/`deviceId`/`tagNames` as target, to send notifications.

Scenario 3: Existing applications using broadcast/Unicast notification in their application

Client

1. Initialize the `MFPPush` client instance in your application:

```
MFPPush push = MFPPush.GetInstance();
push.Initialize();
Implement the interface MFPPushNotificationListener and define onReceive().
class Pushlistener : MFPPushNotificationListener
{
   public void onReceive(String properties, String payload)
   {
         Debug.WriteLine("Push Notifications\n properties:" + properties + "\n payload:" + payload);
   }
}
```

2. Register the mobile device with the push notification service.

```
MFPPushMessageResponse Response = await push.RegisterDevice(null);
if (Response.Success == true)
{
   Debug.WriteLine("Push Notifications Registered successfully");
}
else
{
   Debug.WriteLine("Push Notifications Failed to register");
}
```

3. (Optional) Un-register the mobile device from the push notification service.

```
MFPPushMessageResponse Response = await push.UnregisterDevice();
if (Response.Success == true)
{
   Debug.WriteLine("Push Notifications Failed to unregister");
}
else
{
   Debug.WriteLine("Push Notifications Unregistered successfully");
}
```

4. Remove `WLClient.Push.isPushSupported()` (if used) and use `push.IsPushSupported();`.

5. Remove the following `WLClient.Push` APIs:
    - `registerEventSourceCallback()`
    - `WLOnReadyToSubscribeListener` and `WLNotificationListener` implementation

Server

Remove `WL.Server.sendMessage()` (if used) in your adapter.

Complete the following steps for every application that was using the same event source:

1. Set up the WNS credentials in the **Push Settings** page of MobileFirst Operations Console or use WNS Settings REST API.
2. Add the scope `push.mobileclient` in **Map Scope Elements to security checks** section in the **Security** tab of MobileFirst Operations Console.
3. Create Push tags in the **Tags** page of MobileFirst Operations Console.
4. You can also use the Push Message (POST) (http://www.ibm.com/support/knowledgecenter/en/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/rest_runtime/r_restapi_push_message_post.html?view=kc#Push-Message--POST-) REST API with `userId`/`deviceId`/`tagNames` as target, to send notifications.

Scenario 4: Existing applications using tag notifications in their application

Client

1. Initialize the `MFPPush` client instance in your application:

```
MFPPush push = MFPPush.GetInstance();
push.Initialize();
```

2. Implement the interface MFPPushNotificationListener and define onReceive().

```
class Pushlistener : MFPPushNotificationListener
{
   public void onReceive(String properties, String payload)
   {
         Debug.WriteLine("Push Notifications\n properties:" + properties + "\n payload:" + payload);
   }
}
```

3. Register the mobile device with the push notification service.

```
MFPPushMessageResponse Response = await push.RegisterDevice(null);
if (Response.Success == true)
{
   Debug.WriteLine("Push Notifications Registered successfully");
}
else
{
   Debug.WriteLine("Push Notifications Failed to register");
}
```

4. (Optional) Un-register the mobile device from push notification service.

```
MFPPushMessageResponse Response = await push.UnregisterDevice();
if (Response.Success == true)
{
   Debug.WriteLine("Push Notifications Failed to unregister");
}
else
{
   Debug.WriteLine("Push Notifications Unregistered successfully");
}
```

5. Remove `WLClient.Push.IsPushSupported()` (if used) and use `push.IsPushSupported()`;
6. Remove the following `WLClient.Push` API's:
    - `subscribeTag()`
    - `unsubscribeTag()`
    - `isTagSubscribed()`
    - `WLOnReadyToSubscribeListener` and `WLNotificationListener` implementation
7. Subscribe to tags:

```
String[] Tag = { "sample-tag1", "sample-tag2" };
MFPPushMessageResponse Response = await push.Subscribe(Tag);
if (Response.Success == true)
{
   Debug.WriteLine("Subscribed successfully");
}
else
{
   Debug.WriteLine("Failed to subscribe");
}
```

8. (Optional) Unsubscribe from tags:

```
String[] Tag = { "sample-tag1", "sample-tag2" };
MFPPushMessageResponse Response = await push.Unsubscribe(Tag);
if (Response.Success == true)
{
   Debug.WriteLine("Unsubscribed successfully");
}
else
{
   Debug.WriteLine("Failed to unsubscribe");
}
```

Server

Remove `WL.Server.sendMessage()` (if used) in your adapter.

Complete the following steps for every application that was using the same event source:

1. Set up the WNS credentials in the **Push Settings** page of MobileFirst Operations Console or use WNS Settings REST API.
2. Add the scope `push.mobileclient` in **Map Scope Elements to security checks** section in the **Security** tab of MobileFirst Operations Console.
3. Create Push tags in the **Tags** page of MobileFirst Operations Console.
4. You can also use the Push Message (POST) (http://www.ibm.com/support/knowledgecenter/en/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/rest_runtime/r_restapi_push_message_post.html?view=kc#Push-Message--POST-) REST API with `userId`/`deviceId`/`tagNames` as target, to send notifications.

Last modified on