Interactive and Silent Notifications in Native iOS Applications

Prerequisite: Make sure that you read the Push notifications in native iOS applications (../) tutorial first.

Silent Notifications

iOS 7 and above.

Silent notifications is a feature allowing to send notifications without disturbing the user. Notifications are not shown in the notification center or notification bar.
Callback methods are executed even when the application is running in the background.

> For more information, refer to the "silent notifications" topics in the MobileFirst Platform user documentation, and in Apple's user documentation.

Server API for silent notification

To send silent notification, set a string to indicate the type. Create a notification object by using the `WL.Server.createDefaultNotification` API and set the type as below:

```
notification.APNS.type = "DEFAULT" | "SILENT" | "MIXED";
```

- `DEFAULT` means normal notification, which shows the alert and is kept in the notification center if the application is running in the background.
- `SILENT` means silent notification, which does not show any alert or the message is not placed in the notification center. In the silent type, the aps tag of push notification contains only content-available.
- `MIXED` means a combination of the above: This option invokes the callback method silently and shows the alert.

Client-side API for silent notification

To handle silent notification on the client-side:

- Enable the application capability to perform background tasks on receiving the remote notifications
  To enable background processing, select the project in XCode and in the capabilities tab, select the appropriate background modes, like Remote notifications and Background fetch.
- Implement a new callback method in the `AppDelegate (application: didReceiveRemoteNotification:fetchCompletionHandler:)` to receive silent notifications when the application is running in the background.
- In the callback, check whether the notification is silent by checking that the key content-available is set to 1.
- Call the `fetchCompletionHandler` block method at the end of the notification handler.

Interactive Notifications

iOS 8 and above.

Interactive notifications enable users to take actions when a notification is received without the application being open.
When an interactive notification is received, the device shows action buttons along with the notification message.

> For more information, refer to the "interactive notifications" topics in the MobileFirst Platform user documentation, and in Apple's user documentation.

Server API for interactive notification

To send interactive notification, set a string to indicate the category.
Categories describe a custom type of notification that your application sends and contains actions that a user can perform in response.

- For event-source notifications, create a notification object and set the category as below:

```
notification.APNS.category = "poll";
```

- For broadcast/tag-based notifications, create a notification object and set the category as below:

```
notification.settings.apns.category = "poll";
```

- The category name must be same as the one used on the client side.

Client-side steps for interactive notification

To handle interactive notification on the client-side:

- Enable the application capability to perform background tasks on receiving the remote notifications. This step is required if some of the actions are background-enabled.
  To enable background processing, select the project in XCode and in the capabilities tab, select the appropriate background modes like Remote notifications and Background fetch.
- Set categories before setting `deviceToken` on `WLPush` object in `didRegisterForRemoteNotificationsWithDeviceToken` method in the `AppDelegate` class.

```
-(void)application:(UIApplication *)application
didRegisterForRemoteNotificationsWithDeviceToken:(NSData *)deviceToken{
    NSLog(@"APNS Token : %@", deviceToken);
    if([application respondsToSelector:@selector(registerUserNotificationSettings:)]){
        UIUserNotificationType userNotificationTypes = UIUserNotificationTypeNone | UIUserNotific
ationTypeSound | UIUserNotificationTypeAlert | UIUserNotificationTypeBadge;
        UIMutableUserNotificationAction *acceptAction = [[UIMutableUserNotificationAction alloc] in
it];
        acceptAction.identifier = @"OK";
        acceptAction.title = @"OK";
        UIMutableUserNotificationAction *rejetAction = [[UIMutableUserNotificationAction alloc] init]
;
        rejetAction.identifier = @"Cancel";
        rejetAction.title = @"Cancel";
        [rejetAction setActivationMode:UIUserNotificationActivationModeBackground];</p>
        UIMutableUserNotificationCategory *cateogory = [[UIMutableUserNotificationCategory alloc]
init];
        cateogory.identifier = @"poll";
        [cateogory setActions:@[acceptAction,rejetAction] forContext:UIUserNotificationActionCon
textDefault];
        [cateogory setActions:@[acceptAction,rejetAction] forContext:UIUserNotificationActionCon
textMinimal];</p>
        NSSet *catgories = [NSSet setWithObject:cateogory];
        [application registerUserNotificationSettings:[UIUserNotificationSettings settingsForType
s:userNotificationTypes categories:catgories]];
    }
    [[WLPush sharedInstance] setTokenFromClient:deviceToken.description];
}
```

- To handle the actions the user selects, use the `handleActionWithIdentifier` method of the `UIApplicationDelegate` protocol:

```
- (void)application:(UIApplication *)application handleActionWithIdentifier:(NSString *)identifier
forRemoteNotification:(NSDictionary *)userInfo completionHandler:(void (^)
())completionHandler {
    if ([identifier isEqualToString:@"OK"]) {
        NSLog(@"OK was tapped");
    }
    else if ([identifier isEqualToString:@"Cancel"]) {
        NSLog(@"Cancel was tapped");
    }
    if (completionHandler) {
        completionHandler();
    }
}
```

Make sure to call the `completionHandler`.