



Hewlett Packard
Enterprise

AppPulse Mobile

Adding AppPulse Mobile to Your iOS App

March 2016



Legal Notices

Warranty

The only warranties for Hewlett Packard Enterprise products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HPE shall not be liable for technical or editorial errors or omissions contained herein.

The information contained herein is subject to change without notice.

Restricted Rights Legend

Confidential computer software. Valid license from HPE required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notice

© Copyright 2015-2016 Hewlett Packard Enterprise Development LP

Trademark Notices

Apple is a trademark of Apple Computer, Inc., registered in the U.S. and other countries.

Google is a trademark of Google Inc.

Contents

How to Set Up iOS Apps	4
Advanced Options	8
Crash Stack Limit	8
Protecting Sensitive Data	8
Protecting Sensitive URL Parameters	9
Enabling Users to Opt-In/Opt-Out of Data Reporting	9
Symbolication For Crash Reporting	11
Invoking During Xcode Build	11
Using a Command Line to Upload Debug Symbols	12
Post-Build Wrapping Option	13
Prerequisites	13
To Add AppPulse Mobile to Your App	13
Troubleshooting	15
Compilation Error: Duplicate Symbols Detected	15
Linking Problems	15
Unsupported Frameworks and Devices	15
iOS SDK	16
Handled Exceptions and Crashes API	16
Handled Exceptions	16
Handled Crashes	17
Breadcrumbs API	18
User Action Customization APIs	18
Control Name	19
Control Type	20
Screen Name	21
Screen Name by Control Container	22
Enumeration of Control Types	22
Protecting Sensitive Data Collected by the SDK	23

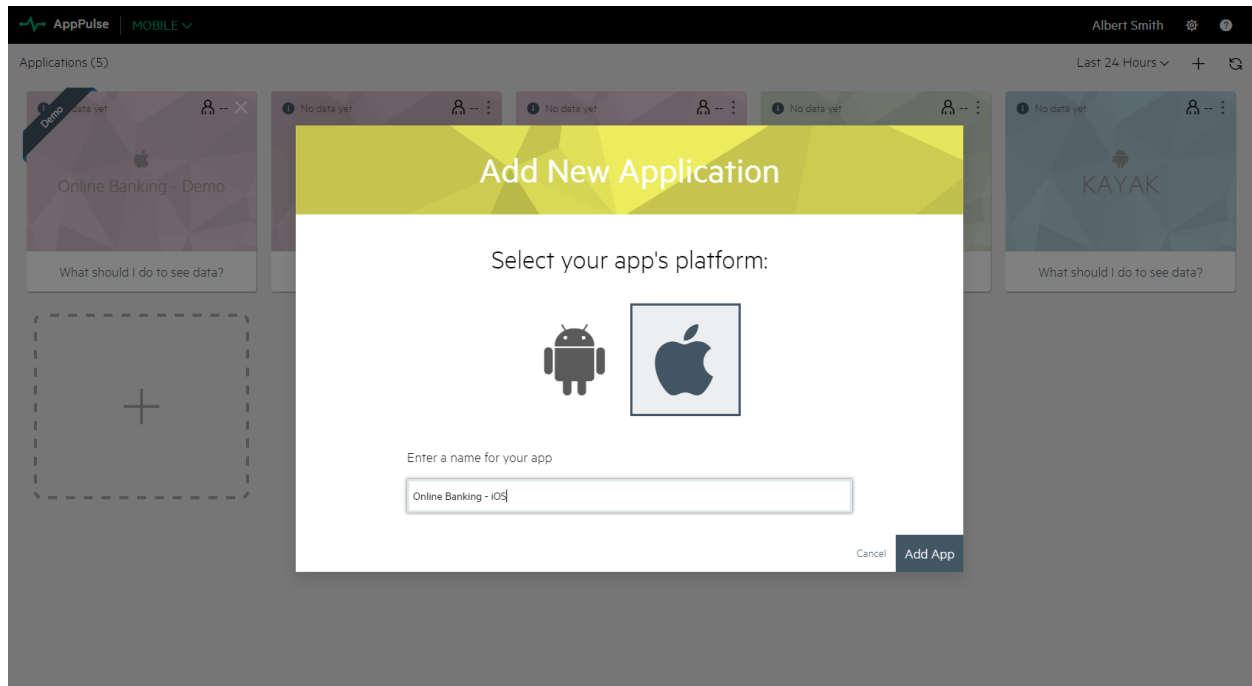
How to Set Up iOS Apps

This section describes how to add AppPulse Mobile to your iOS app.

For a quick video showing this process, [click here](#).

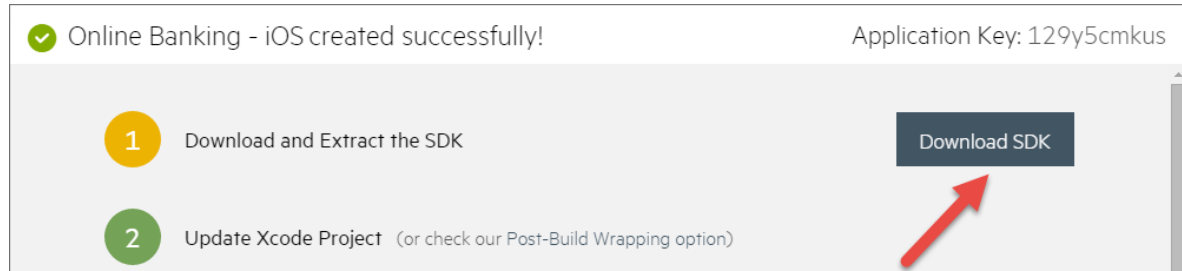
Before you Begin

Within AppPulse Mobile, add an application.



Step 1: Download the SDK

From the window that appears, download and extract the AppPulse Mobile SDK.

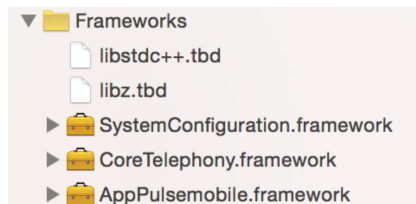


Step 2: Update the Xcode Project

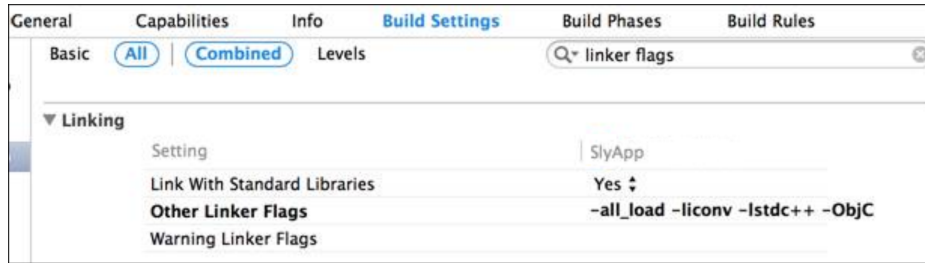
You need to have the development environment for your application set up within Xcode.

(If you do not have access to Xcode and you want to test AppPulse Mobile, see ["Post-Build Wrapping Option" on page 13.](#))

1. Drag AppPulsemobile.framework from the unzipped SDK directory to your project tree. Choose to copy the item to your project.



2. Add the following native libraries to the project (Build Phases > Link Binary with Libraries):
 - SystemConfiguration.framework
 - CoreTelephony.framework
 - libstdc++.dylib in Xcode6 / libstdc++.tbd in Xcode7 and higher
 - libz.dylib in Xcode6 / libz.tbd in Xcode7 and higher
3. In the build settings, under Other Linker Flags, add the **-ObjC** linker flag.



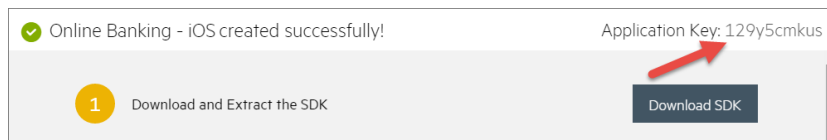
4. Copy the hprummonitor.plist and HPFilter.xml files from the unzipped SDK and add them to the project.



5. In the hprummonitor.plist file, set the **AppId** parameter. Enter the application key from AppPulse Mobile. For example, for the application below you would enter the following:

```
<key>AppId</key>
```

```
<string>129y5cmkus</string>
```



If you already closed the window, you can access the application's settings within AppPulse Mobile and copy the application key from there.

Step 3: Build and Run Your App

1. Build the application, and use your app.
2. Now open AppPulse Mobile, and explore your app's user experience!

What's Next?

1. Test the application to ensure that it functions properly. At this point you will already be able to see data in AppPulse Mobile from the testing devices.
2. When you are ready, submit it to the App Store. As soon as users interact with your app, AppPulse Mobile shows your users' experience in detail.

Note:

- The above steps are mandatory. For other options, see ["Advanced Options" on the next page](#).
- AppPulse Mobile supports apps with iOS 7.0 and higher.

Advanced Options

When you add AppPulse Mobile to your app, you can define advanced options, as described in the following sections:

- ["Crash Stack Limit" below](#)
- ["Protecting Sensitive Data" below](#)
- ["Protecting Sensitive URL Parameters" on the next page](#)
- ["Enabling Users to Opt-In/Opt-Out of Data Reporting" on the next page](#)

Crash Stack Limit

By default, crash reports' stacktrace have a size limit of 2 KB. To modify this limit, open the `hprummonitor.plist` file and set the **StackTraceSizeLimit** parameter (possible range: 0-10).

Protecting Sensitive Data

To prevent reporting sensitive data such as a button or URL showing an account number, AppPulse Mobile automatically shows *** instead of any sequence of 4 or more of the following elements: **0-9**, **.**, **-** (digit, comma, period, hyphen). For example, if your app reports `Paid by account 413-57`, AppPulse Mobile shows `Paid by account ***`.

Before building your project, you can customize data masking as follows:

- **No data masking.** If you do not want these strings replaced, open the `HPFilter.xml` file for editing, and delete its contents. Save the empty file, and build the project.
- **Change data masking rules.** To define specific rules relevant for your particular app, open the file `HPFilter.xml`. This file contains the default rule, and a template for creating additional rules. Each rule has two parts: `<detection string>` `<replacement string>`. The detection uses regular expressions. You can add as many rules as needed; each rule is a separate `Item` node.

Add or edit rules as needed, save the file, and build the project.

Protecting Sensitive URL Parameters

By default, AppPulse Mobile also masks sensitive URL parameters. For example, `http://www.example.com/customers/orders?username=dave&os=android` is reported as `http://www.example.com/customers/orders?username=*&os=*`.

If you need custom masking (for example if a REST URLs contain usernames such as `http://www.example.com/customers/546789876/orders/`), you can configure a mask in the file `HPFilter.xml`.

Example:

```
...
</Items>>
<MaskUrls>
    <MaskUrl>http://www.example.com/customers/(.*)/orders/
    (.*?)/there</MaskUrl>
    <MaskUrl>http://www.example.com/customers/(.*)
    /orders</MaskUrl>
    <MaskUrl>http://www.example.com/customers/(.*)
    /orders/.*</MaskUrl>
</MaskUrls>
```

The URLs are configured as regular expressions, where the capture groups (parenthesized parts) are the parts that will be masked. In the example above, the URL `http://www.example.com/customers/546789876/orders` will be masked to `http://www.example.com/customers/*/orders`.

Enabling Users to Opt-In/Opt-Out of Data Reporting

Some apps give each user the ability to avoid sending data from their device, for example using a Settings menu item (opt-out). Other apps ask users to actively approve data collection, for example when the app is first launched (opt-in).

By default, data is automatically sent from mobile apps to AppPulse Mobile. To modify this, in the `hprummonitor.plist` file set the **sendingEnabled** parameter to **NO**. In this case, data is only collected from users who opt in. (The default value is **YES**, which means data is sent unless users opt out.) Note that the relevant user interface and interaction within the app are up to the application developer.

- Include the header file as follows: `#import <AppPulsemobile/HPUserMonitoringSDK.h>`

- To test for the current reporting status: `BOOL b = [HPUserMonitoringSDK getOptStatus];`
A value of **YES** means that the device is allowed to report data, **NO** means not allowed.
- To change the value: `[HPUserMonitoringSDK setOptStatus:YES];`
Enter **YES** to enable data reporting. The change will take effect the next time the user opens the app.


Symbolication For Crash Reporting

In order to see meaningful information in the crash reports you need to provide us with your app's debug symbols file. This can be done in two ways: via Xcode during the build process, or using a command line.

Note: Symbolication requires the Open API, which is only available for users with a Premium license. To generate Open API credentials you need Admin user permissions.

Invoking During Xcode Build

The preferred method for uploading your symbol files is during XCode build, so that AppPulse Mobile has the symbol file before crashes on the new version occur.


1. From the settings  icon, access the Open API and generate a Client Secret. Copy the Client ID and Client Secret that are generated, as well as the tenant ID.
2. In Xcode, select your project in the navigator and click your application target.
3. Select the Build Phases tab in the settings editor.
4. Click the + icon and choose New Run Script Phase.
5. Add the following two lines of code to the new phase:

```
SCRIPT=`/usr/bin/find ${FRAMEWORK_SEARCH_PATHS} -name apppulse_
upload_dsym\*.sh 2>/dev/null | head -n 1`
$SCRIPT -tenant <tenant ID> -client <client ID> -secret <client
secret>
```

For example:

```
SCRIPT=`/usr/bin/find ${FRAMEWORK_SEARCH_PATHS} -name apppulse_
upload_dsym\*.sh 2>/dev/null | head -n 1`
$SCRIPT -tenant 300 -client 300#C1 -secret 6fe5d2ad-0f35-427c-
8046-3af6dcd8e6db
```

Using a Command Line to Upload Debug Symbols

1. From the settings  icon, access the Open API and generate a Client Secret. Copy the Client ID and Client Secret that are generated, as well as the tenant ID.
2. Find the script `apppulse_upload_dsym.sh` in the SDK, under `AppPulsemobile.framework`.

To upload your debug symbols file so that crash reports are symbolicated, run the following in a command line:

```
apppulse_upload_dsym.sh -tenant <tenant ID> -client <client ID> -secret <client secret> -appkey <application key> -dsym <path to dSYM file> -infoplist <path to the app info.plist file>
```

For example:

```
apppulse_upload_dsym.sh -tenant 300 -secret 6fe5d2ad-0f35-427c-8046-3af6dcd8e6db -client 300#C1 -appkey 142mlunc5h -dsym /Users/hpuser/BuildArtifacts/RUM_Probe_test.app.dSYM -infoplist /Users/hpuser/Documents/DEV/demo-apps/RUM_Probe_test/RUM_Probe_test/RUM_Probe_test-Info.plist
```

Note that on a Mac you need to enclose your client ID in apostrophes; for example `'300#C1'`.

3. (Optional) To see the status of the debug symbols files that have been uploaded for your app, run the following:

```
apppulse_upload_dsym.sh -tenant <tenant ID> -client <client ID> -secret <client secret> -appkey <application key> -status
```

When you are ready to upload your app to the store, make sure you provide us with the mapping file for the specific build you are uploading. We recommend that you embed the above procedure to occur automatically as part of your build process, as described in the previous procedure.

Post-Build Wrapping Option

The previous sections describe how to add AppPulse Mobile to your app in the pre-build stage using Xcode. If you do not have access to Xcode and you want to test AppPulse Mobile, you can use the following post-build procedure.

Note that this procedure is only relevant for testing purposes - you cannot upload this instrumented app to the App Store.

Prerequisites

- The post-build procedure will run on a Mac computer.
- Obtain the application .ipa file from the developer and not from the App Store, because apps in the App Store are encrypted and cannot be re-signed.
- You must have a certificate to re-sign the application. For details on signing, see <https://developer.apple.com/library/mac/documentation/Security/Conceptual/CodeSigningGuide/Procedures/Procedures.html>.
- You should have already added an application in AppPulse Mobile, copied the application key, and downloaded the SDK.

To Add AppPulse Mobile to Your App

The standard AppPulse Mobile SDK contains a folder named **PostBuildWrapper**, which contains the **AppPulse_mobile** tool for post-build wrapping.

1. Run the following command line wrapper tool:

```
Usage: <Unzipped SDK directory>\AppPulse_mobile -appKey  
<application key> [-codesign <certificate name> [-p provision]]  
<path and name of your ipa>
```

The tool generates a file named **<original app name>-AppPulseMobile.ipa**. located in the same directory as the original ipa.

The following is an example of the command:

```
./AppPulse_mobile -appKey n604wzdeuu -codesign 94E6D8G6KE -p  
~/Desktop/MyPP.mobileprovision -o ~/temp  
/Users/johndoe/temp/Cracked/MyApp.ipa
```

Note:

- You can use the parameter `[-o <File path>]` to specify a non-default output file.
- You can update the HPFilter.xml file for privacy masking as described in ["Advanced Options" on page 8](#). If you do so, add the parameter `[-sdfile <File path>]` to specify the non-default privacy masking file.
- If you are missing required developer tools, you will receive a popup message to help install them (i.e. no developer tools were found at `'/Applications/Xcode.app'`, requesting install). Install them to proceed.

2. If you did not enter `-codesign <certificate name>` in the command, you must re-sign the new ipa. You can do this using your certificate and the **AppPulse_mobile** tool, or using your own signing process. For example:
 - Unzip the IPA: `> unzip MyApp-AppPulseMobile.ipa`
 - Re-sign: `> /usr/bin/codesign -f -s "Certificate Name" "Payload/MyApp-AppPulseMobile.app"`
 - Re-package: `> zip -qr "MyApp-AppPulseMobile.resigned.ipa" Payload`
3. Install the application on your device and test it. At this point you will already be able to see data in AppPulse Mobile from the testing devices.
4. When you are ready, instrument it as described in ["How to Set Up iOS Apps" on page 4](#), and submit it to the App Store.

Troubleshooting

Compilation Error: Duplicate Symbols Detected

If compilation of the project fails at the linking stage, and you receive an error message about duplicate symbols being detected, this occurs when your project is linked with the **PLCrashReporter** library (either directly or by including a third-party framework that links with it).

In order to use the AppPulse Mobile framework, you need to remove the conflicting library from your project.

Linking Problems

If building of the project fails at the linking stage, this may be due to a problem with using the -ObjC flag with another library you are using.

Replace the -ObjC flag with the following flag that is specific to AppPulse Mobile: –
**force_load <path to the AppPulse Mobile
SDK>/AppPulsemobile.framework/AppPulsemobile.**

Unsupported Frameworks and Devices

AppPulse Mobile does not support applications developed using the Xamarin or Kony frameworks.

Post-build instrumentation is not supported on 64-bit devices.

iOS SDK

This section includes the following APIs:

- ["Handled Exceptions and Crashes API" below](#)
- ["Breadcrumbs API" on page 18](#)
- ["User Action Customization APIs" on page 18](#)

In some cases, you may not want to show specific information collected via the AppPulse Mobile APIs. For details on setting this up, see ["Protecting Sensitive Data Collected by the SDK" on page 23](#).

Handled Exceptions and Crashes API

AppPulse Mobile automatically identifies and reports crashes. However, there are certain situation where exceptions occur and are caught by the application code, meaning they do not lead to a crash.

The following APIs provide you with the ability to report these kind of exceptions with a severity level that will cause them to be displayed in AppPulse Mobile as either crashes or errors, helping you improve application quality and stability. You can use the following APIs to report the exception or crash you would like to monitor.

Handled Exceptions

A non-fatal exception will be displayed as an event in the Stability > Errors area. AppPulse Mobile will display the exception type, the developer-added description, and a link to show the full stack trace.

HPAppPulse.reportException: (NSEException*) exception Description: (NSString*) description

HPAppPulse.reportException: (NSEException*) exception

- `exception` is an object containing all the relevant information of the problem (stack trace, user message).
- `description` is an optional field in case you want some text displayed in the errors page. If no description is used, the text will be taken from the exception message.

Examples:

reportHandledException:withDescription

```
- (IBAction)loadValue:(id)sender {
    @try{
        [MyClass load:@"MyKey"];
    }
    @catch (NSEException *ex){
        [HPAppPulse reportHandledException:ex
withDescription:@"Caught exception in loading"];
    }
}
```

reportHandledException:exception

```
- (IBAction)loadValue:(id)sender {
    @try{
        [MyClass load:@"MyKey"];
    }
    @catch (NSEException *ex){
        [HPAppPulse reportHandledException:ex];
    }
}
```

Handled Crashes

Use this API when you catch a crash or exception that leads to exit the application. A fatal exception will be displayed as a crash in the Stability > Errors area, and will have the same data as a crash.

HPAppPulse.reportCrash: (NSEException*) exception

`exception` is an object containing all the relevant information of the problem (stack trace, user message).

Example:

reportCrash:exception

```
- (IBAction)onClickLogin:(id)sender {
    @try{
        [MyClass login];
    }
    @catch (NSEException *ex){
        [HPAppPulse reportCrash: ex];
        exit(1);
    }
}
```

Breadcrumbs API

This API gives you the ability to add custom breadcrumbs in critical places in your application, containing information on the application's inner state. AppPulse Mobile displays these custom breadcrumbs in reports on crashes, making it easier for you to investigate the source of the problem.

+ (void) addBreadcrumb:(NSString *)text;

Where `text` is a short logical name which describes the breadcrumb context. This will be displayed in the AppPulse Mobile UI.

Example:

```
- (IBAction)tappedButton:(id)sender {
    [HPAppPulse addBreadcrumb:@"BREADCRUMB at button tapped"];
}
```

User Action Customization APIs

AppPulse Mobile automatically assigns names to your application's screens and user actions (UI controls) based on a number of methods. If you want to customize these screen and control names as they appear in AppPulse Mobile, you can use the following APIs to define naming and grouping as relevant to your needs.

In all classes that use the SDK, include the header file as follows:

```
#import <AppPulsemobile/HPAppPulse.h>
```

This includes the following:

- ["Control Name" below](#)
- ["Control Type" on the next page](#)
- ["Screen Name" on page 21](#)
- ["Screen Name by Control Container " on page 22](#)
- ["Enumeration of Control Types" on page 22](#)

Control Name

Sets the name and type for a given control.

A control is usually a UIView object that is defined as a "sender" in some of the application's actions.

If you would like to consider a few separate controls as an identical control (for example two separate lists that really do the same thing), you can group them together by setting the same name and type. For example, suppose you have a list containing three items: Flights to x, Flights to y, and Flights to z, and AppPulse Mobile identifies this as a menu with three distinct actions. You can set each of the options with the same name/ID, grouped under the category "Flights to a location."

```
+ (void) setControlName:(UIView*)control controlName:(NSString*)controlName;
```

Example:

```
@interface ViewController ()
@property (weak, nonatomic) IBOutlet UIButton *button;
@end

- (void)viewDidLoad {
    [super viewDidLoad];
    [HPAppPulse setControlName:self.button screenName: @"My
Button"];
}
.
. <rest of your code>
.
@end
```

Control Type

Sets the type for a given control. (Note that **HPControlType enum** is defined below.)

For example, you can use this if you want a menu to be considered a list, so it will be the same as your other lists and not a separate action. Another example would be if you think the type of control is not what it appears to be, like if you have a control which looks like a button, but is in fact a list item. The automatic naming will report it as a list item, but you can change that to a button..

+ (void) setControlType:(UIView*)control controlType:(HPControlType)type

Example:

```
@interface ViewController ()
@property (weak, nonatomic) IBOutlet UIButton *button;
@end

- (void)viewDidLoad {
    [super viewDidLoad];
    [HPAppPulse setControlType:self.button
controlType:HPLListItemControl];
}
.
. <rest of your code>
.
@end
```

Screen Name

You can set the name for a specific screen by calling the following API on the existing View Controller.

This should be added as a part of the viewDidLoad method of the UIViewController object. You can also set the screen name for a specific control.

+ (void) setScreenName:(UIViewController*)vc screenName:(NSString*)name;

Example:

```
@implementation ViewController

- (void)viewDidLoad {
    [super viewDidLoad];
    [HPAppPulse setScreenName:self screenName: @"My Screen"];
}
.
. <rest of your code>
.
@end
```

Screen Name by Control Container

Sets the screen name based on a given control container.

For example, if a list item is clicked and you want the screen name to be based on the list container, the user will call this method on the list rather than on the list item.

+ (void) setControlName:(UIView*)control controlName:(NSString*)controlName withScreenName: (NSString*)screenName;

Example:

```
@interface ViewController ()
@property (weak, nonatomic) IBOutlet UIButton *button;
@end

- (void)viewDidLoad {
    [super viewDidLoad];
    [HPAppPulse setControlName:self.sendButton
controlName:@"MyButton" withScreenName:@"My Screen Name"];
}
.
. <rest of your code>
.
@end
```

Enumeration of Control Types

```
typedef enum HPControlType{
    HPButtonControl,
    HPTabControl,
    HPSwitchControl,
    HPStepperControl,
    HPSliderControl,
    HPSearchBarControl,
    HPNavigationBarControl,
    HPListItemControl,
    HPCollectionViewControl

} HPControlType;
```

Protecting Sensitive Data Collected by the SDK

In some cases, you may not want to show specific information collected via the AppPulse Mobile APIs, such as a screen name that includes a sensitive parameter. To mask such data, open the file HPFilter.xml. Uncomment and edit the <SDK> block as needed.

This block contains a template for creating masking rules. Each rule has two parts: <detection string> <replacement string>. The detection uses regular expressions. You can add as many rules as needed; each rule is a separate SDKItem node.

