Getting Started with the iOS SDK

Version 4

Introduction

The Ballista Framework (formerly called AppRelay) for iOS makes it easy for developers to collect user engagement data and custom reports from their apps. The Ballista service also provides an easy way to target and provide rich campaigns to your user.

This document provides a guide to getting started with the service and setup your application to enable metrics tracking.

Before You Begin

Before begin integrating the Framework, make sure you have the following:

- The latest Ballista iOS Framework Library
  the latest library is available on a public git repository:
  https://github.com/wherecloud/ballista-sdk-ios-precompiled

- The CrashReporter iOS Framework Library V1.1-rc2 or later available at
  https://code.google.com/p/plcrashreporter/

- Your application bundle identifier (in the form `com.company.xyz`) which you need to provide us via email at: info@goballista.com

  - Note: Please send it from the email account you want us to use as your username.

- Upon reception of your bundle id, we will return you an access to the Ballista dashboard accessible here: https://client.goballista.com

The minimum deployment target version when using the Ballista iOS Framework is limited to iOS 5.0 and higher.

Getting Started

There are two steps to getting started with the framework: (1) adding it to you Xcode project and (2) initializing the tracker.

Adding the Framework / Resources and Configuring Your Project

Make sure you follow all the steps below to ensure the integration is performed correctly.

• Drag & drop `AppRelay.framework` and `CrashReporter.framework` from the finder to your project.
You can either copy it in your source tree, which make it easier to work with other developers in your team, or just reference it relatively to your project.

Warning: If you're already using a third party framework for crash reports that statically integrates the PLCrashReporter framework (such as Crittercism), you should not add the `CrashReporter.framework` in your project or you will experience errors when the application gets build (linking error: duplicate symbols)

• Drag & drop the framework's `AppRelay.framework/Resources` folder from the finder to your project.
Their are some mandatory resources files in it like the AppRelay.plist file.

• In the "Build Settings" of your project, ensure you have "`-ObjC -all_load`" defined in the Other Linker Flags.
This is mandatory for the compiler to embed objective-C code defined in categories.

• In the "Build Phases" of your project, expands the "Link Binary With Library" section and add the following frameworks and libraries :

```
CoreTelephony.framework
Security.framework
AdSupport.framework (weak link)
CFNetwork.framework
SystemConfiguration.framework
QuartzCore.framework
PassKit.framework (weak link)
libz.dylib, libstdc++.dylib, libicucore.dylib
```

Initializing the Ballista SDK

To initialize the Ballista SDK, import the `<AppRelay/Apprelay.h>` file in your application delegate and add this code to the `application:didFinishLaunchingWithOptions:` method:

```
- (BOOL)application:(UIApplication *)application
        didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    NSString *bundleIdentifier = [[NSBundle mainBundle] bundleIdentifier];

    [[ARService sharedService] startTrackerWithAPIKey:@"YOUR_API_KEY"
                                              bundleId:bundleIdentifier];

    // Initialize your application here!
    return YES;
}
```

With that single line, the SDK will be monitoring “sessions” of the application running in foreground and background and sending events to the platform accordingly.

Note: Please ensure your application is initializing the Ballista SDK with the `startTracker-WithAPIKey` method. The other methods available are needed when a deeper integration with the application is being performed.

Warning: If you’re already using a third party framework for crash reports (like Crittercism), you should disable the crash reports from Ballista framework to avoid unknown exceptions thrown by PLCrashReporter. Add the following line of code just before the `startTrackerWithAPIKey:bundleId:` call to ARService.

You will still access the quality of service details in the goballista website but you will not be able to download the `.crash` files for each crash that will be reported.

```
[[ARService sharedService]setCrashReportsEnabled:NO];
```

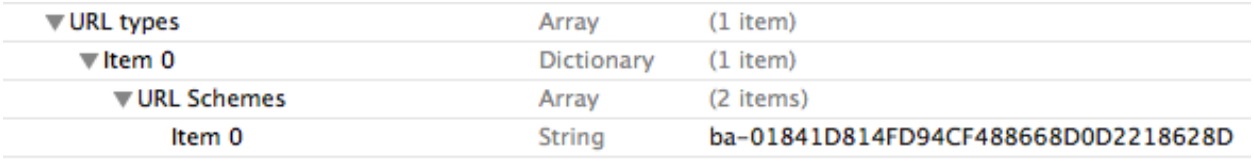
Enabling Ballista Color ID Support

Ballista allows you to easily test event propagation and campaign distribution using a revolutionary color id technology.

This technology allows you to easily pinpoint specific devices for testing and validating events propagation and campaign reception

You can *optionally* support the Ballista Color ID in your mobile applications by integrating a specific URL Scheme.

This is done by enabling support in the `info.plist` of your application:



The syntax is to prefix the value of your API key with the string **`"ba-"`**.

You also have to forward URL open requests to the Ballista SDK, if those URLs are not specific to your application.

Add the following code to your `AppDelegate.m` file :

```
- (BOOL)application:(UIApplication *)application
           openURL:(NSURL *)url
  sourceApplication:(NSString *)sourceApplication
         annotation:(id)annotation{

    //Do Your Stuff here :
    //return YES if you handled an URL of your concern

    //Ballista Color Id integration
    if([[ARService sharedService]handleApplication:application
                                           openURL:url
                                 sourceApplication:sourceApplication]){
        return YES;
    }

    return NO;
}
```

Testing the SDK

You can now test that your application is properly integrated:

To test proper integration, log into the Ballista Dashboard using the provided credentials. Within that dashboard, select the Activity Log button.

Now start your Ballista-enabled application, you should see events being display that would look something like below:


```
{
  "appid": "com.wci.turntape",
  "appversion": "1.1.1",
  "appbuildnumber": "457",
  "devicemodel": "iPhone4,1",
  "devicetimezone": "-25200",
  "devicelocale": "en",
  "deviceresolution": "640x960",
  "systemname": "iOS",
  "systemversion": "6.1.3",
  "networktype": "-1",
  "networkcarrier": "310VZW",
  "networkcarriername": "Verizon",
  "inetaddress": "99.48.88.9",
  "geo_latitude": 33.92689895629883,
  "geo_longitude": -117.86119842529297,
  "geo_city": "Brea",
  "geo_region": "CA",
  "geo_country": "US",
  "userid": "4D0B4C4C-09C4-4A5A-AAD7-55D45A9BA84F",
  "vendorid": "D0B5D5E9-E7CC-4142-908C-83560DA7F84C",
  "advertisingid": "1D35603F-CB63-448C-9B4C-3E77B6686E00",
  "advertisingtrackingenabled": "1",
  "sdkversion": "1.3.0 (138)",
  "sdkcampaignsenabled": "1",
  "sessionid": "E986D600-7BC3-444C-8524-FC9CB3AE056C",
  "eventid": "16D95AC7-42CA-49E1-9511-952B2B2D07ED",
  "timestamp": "1377718229.328880",
  "event": "app.session.begin",
  "value": {}
}
```

Tracking Triggers and Timed Events

The Ballista iOS Framework supports the tracking of two types of events: *triggers* and *timers*. Triggers are designed to track a single action, this is the most basic form of tracking (i.e., deleting a note). Timers on the other hand are events with a duration, these are useful to track the time the user spent doing an activity (i.e., playing a game level).

Both types of events can carry custom properties (in the form of a dictionary of key/value pairs) to be use for analytics or segmentation.

For example, tracking a story favorited by the user in a news application:

```
[[ARService sharedService] trackEvent:@"Story Favorited"
                           properties:@{ @"story": @"<story-tag>" }];
```

Any kind of metadata can be appended to an event.

Tracking a *timer* is done in the same manner. In the following example, we use a timer to start tracking the time spent by a user reading a specific section of a news application:

```
[[ARService sharedService] trackTimedEvent:@"Section Read"
                                properties:@{ @"section" : @"sports" }];
```

When the user leaves the section, we simply call:

```
[[ARService sharedService] stopTimedEvent:@"Section Read"];
```

Timed Events Life Cycle

The Ballista framework takes care of the timed events life cycle so you don't have to. Meaning that by default, a timed events will be:

- "`paused`" when the application is inactive (i.e., application is in foreground by doesn't have the focus, like when the Notification Center is covering the application) and "resumed" when the application goes back to being active;
- "`stopped`" when the application goes in background;
- "`restarted`" when the application goes back in foreground;

This default behavior can be disabled if you want to manage the life cycle of timed events manually for specific needs in your application.

Timed Events Real-Time Reporting

Another benefits of using timers (besides the automatic reporting of its duration) is their real-time reporting in the Ballista platform. Every timer can be displayed in a capsule on the dashboard using a live counter displaying the total live number of users who are currently being "timed."

This gives you as a glimpse the current usage of a features in your application.