



Adobe® Experience Cloud
AppMeasurement 3.x for iOS

Contents

AppMeasurement 3.x for iOS.....	4
Download the Library.....	5
Analyst Quick Start.....	6
Developer Quick Start.....	7
Video Measurement Quick Start.....	11
Lifecycle Metrics.....	14
Context Data.....	15
Configuration Variables.....	16
Tracking Variables.....	18
Methods.....	21
Offline Tracking.....	29
Target.....	30
Audience Management.....	34
PhoneGap Plug-in.....	36
PhoneGap Plug-in Methods.....	37
iOS Version 2.x to 3.x Migration Guide.....	49

iOS Device Versions.....51

Using Bloodhound to Test Mobile Applications.....53

Sample Code.....55

Contact and Legal Information.....56

AppMeasurement 3.x for iOS

Adobe® AppMeasurement for iOS lets you measure native Apple* iPhone* and iPad* applications in the Adobe Experience Cloud.

For a list of the 4.x Experience Cloud SDKs, see the *SDK Documentation* section in [Adobe Mobile Services](#).

This guide is divided into two sections; one section for the Web Analyst that has Analytics experience, and one section for the iOS developer with mobile app development experience.

Analyst Quick Start	This section walks you through selecting and configuring the events, eVars, and props that you'll use to collect iOS data. Steps are also included to create Processing Rules to copy the context data sent by the iOS libraries to these variables.
Developer Quick Start	This section walks you through implementing the iOS library and adding the code required for a standard implementation. Steps are included to show you how to send custom events and other data.

Supported Versions

iOS 4.3 or later.

Release History

[iOS](#)

Documentation Revision History

Revision Date	Description
August 05, 2013	Added support for Audience Management .
November 08, 2012	Added the <code>lifecycleSessionTimeout</code> variable that is new in version 3.1.3.
July 26, 2012	Updated the description for <code>a.DeviceName</code> to include details about the revision number. See Lifecycle Metrics .
July 19, 2012	Initial release.

Download the Library

Download instructions and links for all AppMeasurement mobile platforms are available at the [Measuring and Optimizing Mobile Applications](#) page on Developer Connection.

You must have a free Developer Connection account or a reports and analytics login to download the libraries. The download links do not appear until you have logged in.

Analyst Quick Start

As the analyst, you need to enable the Mobile Application Reports for your report suite. If you have additional metrics to capture, you should provide your developer a description of the context data variables that should be sent by the application.

For example, to collect a username after login, you could have your developer set the username into a context data variable called `myco.username`.

Enable Mobile Application Reports in Analytics

Analytics provides an interface to enable Mobile App Lifecycle Tracking. This mapping lets Analytics automatically generate the **Mobile Application Reports**.

1. Open **Admin Tools > Report Suites > Edit Settings > Mobile Management > Mobile Application Reporting**.
2. Click **Enable Mobile App Lifecycle Tracking**.
3. If you are using Google Play Campaign Tracking for Android, click **Enable Google Play Campaign Tracking**.

The lifecycle metrics are now captured, and **Mobile Application Reports** appear in the **Reports** menu in the marketing reports interface.

Capture Additional Metrics using Processing Rules

If your implementation sends additional events and dimensions using context data, you must configure processing rules to capture that data.

Before creating Processing Rules, someone in your organization must become certified. For additional information, see the [Processing Rules Help](#).

After Processing Rules are enabled, the [Copy a Context Data Variable](#) example demonstrates the process required to map context data.

(Optional) Enable Offline Tracking

If you plan to store hits when the device is offline, your report suite must be timestamp-enabled.

After you enable offline tracking, all hits must be time-stamped or they are not collected. If you are currently reporting AppMeasurement data to a report suite that also collects data from JavaScript, you might need to set up a separate report suite for mobile data to avoid data loss, or include a custom timestamp on JavaScript hits using the `s.timestamp` variable.

If you are unsure if your report suite is timestamp-enabled, contact Customer Care.

Developer Quick Start

This guide walks you through the steps to implement the iOS library and start sending measurement data.

- [Get the Library](#)
- [Add the Library to your Project](#)
- [A Quick Word on the TrackingHelper](#)
- [Implementation](#)

Get the Library

Visit https://developer.omniture.com/en_US/content_module/mobile to download the iOS library.

After unzipping the download file, you'll have the following software components:

- `ADMS_Measurement.h`: The Objective-C header file used for iOS AppMeasurement.
- `admsAppLibrary.a`: a fat binary containing the library builds for devices (armv7 and armv7x) as well as the simulator (i386/x86_64). Requires iOS 4.3 or later.
- `Examples/TrackingHelper.h`, `Examples/TrackingHelper.m`: Optional class that is designed to keep tracking code separate from your application logic.

Add the Library to your Project

1. Launch the Xcode IDE.
2. Copy the `ADMS_AppLibrary` folder from the `ADMS_AppLibrary-*.DMG` you downloaded to your project folder or another location on your drive.
3. Drag and Drop the `ADMS_Measurement` folder on your Xcode project, and confirm the following settings:
 - Select **Copy items into destination group's folder (if needed)**.
 - Select **Create groups for any added folders**.
 - Select the targets where you want to use AppMeasurement code.
4. Click **Finish**.

A Quick Word on the TrackingHelper

The SDK includes an additional, optional class, called `TrackingHelper`. `TrackingHelper` provides a way to separate your measurement code from your application logic.

Consider the following example: In your application, you want to send an event that tracks each login. You could add the following code directly after your login code to send this event (don't worry about the code details, we'll get to those later):

```
ADMS_Measurement *measure = [ADMS_Measurement sharedInstance];
NSMutableDictionary *contextData = [NSMutableDictionary dictionary];
[contextData setObject:@"username_value" forKey:@"username"];
//add additional key:value pairs to this dictionary...

[measure trackEvents:@"event1" withContextData:contextData];
```

This direct approach is OK, but wouldn't you rather put this code somewhere else so it is out of your way while you are developing your application? The `TrackingHelper` is that "somewhere else".

Inside of `TrackingHelper`, you could place this code in a method called `trackLogonEvent`:

```
+ (void)trackLogonEvent:(NSString *)username_value{
NSMutableDictionary *contextData = [NSMutableDictionary dictionary];
[contextData setObject:username_value forKey:@"username"];
//add additional key:value pairs to this dictionary...

[measure trackEvents:@"event1" withContextData:contextData];
```

Now, in your application code, you can track a logon event with a simple call to `trackLogonEvent`:

```
[TrackingHelper trackLogonEvent:@"username_value"];
```

The measurement call in your application code is down to a single line. Plus, if the underlying measurement logic needs to change, you'll need to update only the `TrackingHelper`. If another developer adds to your code, he or she can use the simplified methods you've defined without taking a crash course in the `AppMeasurement` library.

We think you'll prefer using the `TrackingHelper` for new implementations. The remaining steps in this guide walk you through the steps to set up `TrackingHelper` and start sending measurement data.

If you'd rather implement without `TrackingHelper`, you can safely remove `TrackingHelper.h` and `TrackingHelper.m` from your project.

Implementation

1. Open `TrackingHelper.m` and update the report suite ID and tracking server:

```
NSString *const TRACKING_RSID = @"YOUR_RSID_HERE" ;  
NSString *const TRACKING_SERVER = @"YOUR_SERVER_HERE" ;
```

These values are required, and must be correct or measurement won't work. If you are unsure about these values, ask your Analytics expert.

2. Find the `+(void)configureAppMeasurement` method. This is where you enable SSL, enable offline tracking, and make other global changes to your configuration. For now, un-comment the line to enable `debugLogging` until things are working the way you'd expect.
3. Open your `AppDelegate.m` file and add the following line to import the `TrackingHelper` header file:

```
#import "TrackingHelper.h"
```

4. Add a call to `TrackingHelper configureAppMeasurement` to your `application:didFinishLaunchingWithOptions` method:

```
-(BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{  
[TrackingHelper configureAppMeasurement];  
}
```

That's it! You've now implemented basic app tracking in your iOS app. Your iOS app will now send lifecycle metrics, including launches, upgrades, crashes, and daily and monthly users.

Where to go from here:

- [\(Recommended\) Track Lifecycle Events](#). We recommend this so much, we turned it on by default.
- [\(Optional\) Track Navigation States](#). The measurement library can also automatically track navigation events, based on the views you define in your application. This can be enabled with a few lines of code.
- [\(Optional\) Track Custom Events](#). Add custom methods to `TrackingHelper` to track all of the events you want to measure in your application. You might add methods to track logons, in-app purchases, and so on.
- [\(Optional\) Manually Track Custom States](#). An application state is similar to a page view. This section shows you how to add a custom method to `TrackingHelper` to track an app state.
- [Video Measurement Quick Start](#). Add code to track video metrics including video views, total time played, and most popular videos.

(Recommended) Track Lifecycle Events

By default, each time your app is launched, a *single hit* is sent to track installs, upgrades, engaged days, and the other [Lifecycle Metrics](#).

We can't think of a reason you'd want to disable this, but we added an option to do it anyway. To disable lifecycle tracking, in `TrackingHelper.m`, find the line that contains `setAutoTrackingOptions` and uncomment the following line:

```
[measure setAutoTrackingOptions:ADMS_AutoTrackOptionsNone];
```

(Optional) Track Navigation States

In addition to tracking lifecycle events, the library can be configured to automatically track navigation events. When navigation auto tracking is enabled, a hit is sent each time a view is loaded within a `UINavigationController` or a `UITabBarController`.

To enable navigation auto tracking, open `TrackingHelper.m` and find the line that contains `setAutoTrackingOptions`. Uncomment the following line:

```
[measure setAutoTrackingOptions:ADMS_AutoTrackOptionsLifecycle |  
ADMS_AutoTrackOptionsNavigation];
```

This line enables both lifecycle and navigation auto tracking.

App State Name (Auto Tracking)

In mobile applications, an app state is synonymous with a page view on a website. The library populates the app state value using the `title` attribute of each view controller.

To set a meaningful title for each of your view controllers:

Find the `viewDidLoad` method:

```
- (void)viewDidLoad  
{  
  ...  
}
```

Set the title attribute anywhere in this method:

```
- (void)viewDidLoad  
{  
  ...  
  self.title = @"ViewControllerName";  
}
```

The value provided for title appears as the page name in Analytics reports, and a page view is counted for each app state.

(Optional) Track Custom Events

Custom events are success metrics that are unique to your application. You might send a custom event when a user logs in, initiates an in-app purchase, or clicks a link to your Facebook page. The tracking helper class contains an example that shows how to track a custom event. You can use this method as a template to add additional event tracking methods.

In `TrackingHelper.m`, define a custom event track method:

```
+ (void)trackCustomEvents {  
  NSMutableDictionary *contextData = [NSMutableDictionary dictionary];  
  [contextData setObject:@"value" forKey:@"contextKey"];  
  
  [[ADMS_Measurement sharedInstance] trackEvents:events withContextData:contextData];  
}
```

After you update this function to send your events, make sure the signature appears in `TrackingHelper.h`:

```
+ (void)trackCustomEvents;
```

Throughout your code, you can call this method to track a custom event:

```
[TrackingHelper trackCustomEvents];
```

Use this process to add as many event tracking methods as you need. [A Quick Word on the TrackingHelper](#) contains an example method to track a login event.

(Optional) Manually Track Custom States

The tracking helper class also contains an example that shows how to track an application state. Tracking a custom state is very similar to tracking an event, the only difference is you use the `trackAppState` method instead of `trackEvents`.

```
[measure trackAppState:@"name" withContextData:contextData];
```

From a reporting standpoint, `trackAppState` increments page views, while `trackEvents` does not.

Video Measurement Quick Start

Video measurement is described in the [Measuring Video in Analytics](#) guide. The general process to measure video is very similar across all AppMeasurement platforms. This quick start section provides a basic overview of the developer tasks along with code samples.

The video measurement methods are defined in a separate header file, called `ADMS_MediaMeasurement.h`. The same library, `admsAppLibrary.a`, is used for application and video tracking. The documentation refers to these methods as the media module for consistency across platforms.

You need to have a measurement instance configured before you can use the media module.

To implement video tracking on iOS, complete the following tasks:

- [Map Player Events to Analytics Variables](#)
- [Configure Milestones, Segments, and Call Frequency](#)
- [Track Player Events using Auto Track](#)

Map Player Events to Analytics Variables

To configure video measurement, you need to map the Analytics events and eVars selected for video tracking to context data variables. For more information, see [Analytics Video Configuration](#).

Your Web Analyst should provide you with a [Video Implementation Worksheet](#) that contains a list of the Analytics variables that they configured to receive video data:

- Video Name (eVar): eVar2
- Video Name (Prop): prop2
- Segments (eVar): eVar3
- Content Type (eVar): eVar1
- Video Time (Event): event3
- Video Views (Event): event1
- Video Completes (Event): event7
- Video Segment Views (Event): event2

Add the following code, replacing the Analytics variable you selected with the example in the code:

1. Launch the Xcode IDE.
2. In `TrackingHelper.m`, update the `contextDataMapping` object with the Analytics variables you selected for your implementation.
 - a. Add the following code after the code you added in [Implementation](#):

```
//get sharedInstance
ADMS_MediaMeasurement *mediaMeasure = [ADMS_MediaMeasurement sharedInstance];

mediaMeasure.contextDataMapping = [NSDictionary dictionaryWithObjectsAndKeys:
    @"eVar2,prop2",@"a.media.name",
    @"eVar3",@"a.media.segment",
    @"eVar1",@"a.contentType", //note that this is not in the .media namespace
    @"event3",@"a.media.timePlayed",
    @"event1",@"a.media.view",
    @"event2",@"a.media.segmentView",
    @"event7",@"a.media.complete",
    nil];
```

3. [Configure Milestones, Segments, and Call Frequency](#).

4. [Track Player Events using Auto Track.](#)

- When configuration is complete, add a call to `TrackingHelper configureMediaMeasurement` to your application:`didFinishLaunchingWithOptions` method:

```
-(BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{
    [TrackingHelper configureMediaMeasurement];
}
```

Configure Milestones, Segments, and Call Frequency

Milestones let you send an event when a specific point in the video is reached. Segments let you divide your video into sections for more granular tracking. Call frequency lets you send measurement calls with time viewed at specific intervals.

For more information, see [Video Metrics](#).

Map Milestones

You can define milestones based on a percentage of total length or based on seconds elapsed.

This example defines milestones as a percentage of total length. For a 2 minute video, the following code sends milestone events at 30 seconds, 60 seconds, and 90 seconds.

```
//this is assigned in context data - see contextDataMapping example
NSDictionary *_milestoneMappingDict = [NSDictionary dictionaryWithObjectsAndKeys:
    @"event4", @"25",
    @"event5", @"50",
    @"event6", @"75",
    nil];
```

Now add the `_milestoneMappingDict` to your `contextDataMapping` variable:

```
_milestoneMappingDict, @"a.media.milestones"
```

Map Offset Milestones

This example defines milestones by seconds elapsed from the beginning of the video. For a 2 minute video, the following code send milestone events at 20 seconds, 40 seconds, and 60 seconds regardless of the total video length.

```
//this is assigned in context data - see contextDataMapping example
NSDictionary *_offsetMilestoneMappingDict = [NSDictionary dictionaryWithObjectsAndKeys:
    @"event8", @"20",
    @"event9", @"40",
    @"event10", @"60",
    nil];
```

Now add the `_offsetMilestoneMappingDict` to your `contextDataMapping` variable:

```
_offsetMilestoneMappingDict, @"a.media.offsetMilestones"
```

Examples

```
//get sharedInstance
ADMS_MediaMeasurement *mediaMeasure = [ADMS_Measurement sharedInstance];

//Track By Milestones:
mediaMeasure.trackMilestones = @"25,50,75";

// track Milestones & segmentByMilestones:
mediaMeasure.trackMilestones = @"25,50,75";
mediaMeasure.segmentByMilestones = YES;

// track by seconds:
mediaMeasure.trackSeconds = 15;

// track Milestones & segmentByMilestones & seconds:
mediaMeasure.trackMilestones = @"25,50,75";
```

```

mediaMeasure.segmentByMilestones = YES;
mediaMeasure.trackSeconds = 15;

// track offsetMilestones & segmentByOffsetMilestones:
mediaMeasure.trackOffsetMilestones = @"15,30,45,60,75,90";
mediaMeasure.segmentByOffsetMilestones = YES;

// track offsetMilestones:
mediaMeasure.trackOffsetMilestones = @"5,15,45";

```

Track Player Events using Auto Track

Auto track automatically tracks native iOS video player events such as start, stop, and pause, in the native iOS player (MPMoviePlayer). This prevents you from needing to manually track these events and call the open, play, stop, and close methods directly.

See [What is Autotrack?](#)

Video Name

The video name is set to the filename.

Enable Auto Track

The following example enables video auto tracking:

```

ADMS_MediaMeasurement *mediaMeasure = [ADMS_Measurement sharedInstance];
[mediaMeasure setAutoTrackingOptions: ADMS_MediaAutoTrackOptionsMPMoviePlayer];

```

Track Player Events using Manual Video Tracking

This lets you call open, play, stop, and close at the appropriate times. For example:

Load: Call open and play

Pause: Call stop. For example, if a user pauses a video after 15 seconds, call stop("Video1", 15)

Buffer: Call stop while the video buffers. Call play when playback resumes.

Resume: Call play. For example, when a user resumes a video after initially playing 15 seconds of the video, call s.play("Video1", 15).

Scrub (slider): When the user drags the video slider, call stop. When the user releases the video slider, call play.

End: Call stop, then close. For example, at the end of a 100-second video, call stop("Video1", 100), then close("Video1").

To accomplish this, you can define four custom functions that you can call from the media player event handlers. The various parameters passed into open, play, stop, and close come from the player. The following pseudocode demonstrates how this might be done:

```

[mediaMeasure open:mediaName length:mediaLength playerName:mediaPlayerName];
[mediaMeasure play:mediaName offset:mediaOffset];
[mediaMeasure stop:mediaName offset:mediaOffset];
[mediaMeasure close:mediaName];

```

Lifecycle Metrics

This worksheet lists the metrics and dimensions that can be measured automatically by the mobile library.

Context Data

Context data is the preferred method to send application data to collection servers.

Instead of explicitly assigning values to props and eVars in your code, you can send data in context data variables that are mapped in Analytics using Processing Rules. Processing Rules provides a powerful graphical interface to make changes to data as it is received. Based on the values sent in context data, you can set events, copy values to eVars and props, and execute additional conditional statements.

Using context data helps prevent you from making code updates to support different report suite configurations.

There are two ways to send context data with the tracking methods.

- **Persistent:** Set name-value pairs directly on the `PersistentContextData` object to send the value in with each server call. These values persist for all calls until deleted.
- **Single-Call:** Pass context data name-value pairs in a dictionary to be sent with that call only.

Persistent Context Data

Values placed in Persistent Context Data are sent with each server call. In the following example, "key" and "value" are sent with all `trackAppState` calls:

```
ADMS_Measurement *measure = [ADMS_Measurement sharedInstance];
NSMutableDictionary *contextData = [NSMutableDictionary dictionary];
[contextData setObject:@"value" forKey:@"key"];
//add additional key:value pairs to this dictionary...
[measure setPersistentContextData:contextData];

[measure trackAppState:@"state1"];
[measure trackAppState:@"state2"];
[measure trackAppState:@"state3"];
```

Single-Call Context Data

To send context data for a single call, send Context Data as a parameter to the tracking call (`trackAppState`, `trackEvents`, and so on).

In the following example, "key" and "value" are sent with all three `trackAppState` calls. However, "key2" and "value2" are sent only with the second `trackAppState` call:

```
ADMS_Measurement *measure = [ADMS_Measurement sharedInstance];

NSMutableDictionary *contextData = [NSMutableDictionary dictionary];
[contextData setObject:@"value" forKey:@"key"];
//add additional...

[measure setPersistentContextData:contextData];

NSMutableDictionary *contextDataState = [NSMutableDictionary dictionary];
[contextDataState setObject:@"value2" forKey:@"key2"];

[measure trackAppState:@"state name"];
[measure trackAppState:@"state name" withContextData:contextDataState];
[measure trackAppState:@"state name"];
```

Configuration Variables

All configuration variables are optional except `ReportSuiteID` and `trackingServer`.

Shared Instance

Before making measurement calls, you must retrieve the singleton instance of the library for each method you call on the measurement library:

```
ADMS_Measurement *measure = [ADMS_Measurement sharedInstance];
```



Note: For string variables, use only *NSStrings* (`@"value"`), not *C-Strings* (`"value"`).

Configuration Variables	Description
reportSuiteIDs	<p>(Required) The report suite or report suites (multi-suite tagging) that you wish to track. To track to multiple report suites, pass a comma delimited list of the names of each report suite.</p> <p>You cannot override this variable for a single call, you must change the persistent value.</p> <p>For example:</p> <pre>measure.reportSuiteIDs = @"rsid";</pre> <p>Multi-suite tagging:</p> <pre>measure.reportSuiteIDs = @"rsid1, rsid2";</pre>
trackingServer	<p>(Required) Identifies the collection server.</p> <p>This variable should be populated with your tracking server domain, without an "http://" or "https://" protocol prefix. The protocol prefix is handled automatically by the library based on the <code>ssl</code> variable.</p> <p>If <code>ssl</code> is YES, a secure connection is made to this server. If <code>ssl</code> is NO, a non-secure connection is made to this server.</p> <p>You cannot override this variable for a single call, you must change the persistent value.</p> <p>For example:</p> <pre>measure.trackingServer = @"metrics.corpl.com";</pre>
visitorID	<p>Default: CFUUID</p> <p>Unique identifier for each visitor. By default, an app-specific unique visitor id is generated (using Apple's CFUUID) on initial launch and then stored in a user defaults key. This key is used by AppMeasurement from that point forward. This key is also saved and restored during the standard application backup process.</p> <p>We recommend using the default unless you have a specific use case to set the visitorID. Setting a custom visitorID has the potential to cause duplicate visits when using lifecycle tracking.</p> <p>To avoid this, set the visitor ID before you configure app measurement.</p>
characterSet	<p>Default: UTF-8</p>

Configuration Variables	Description
	For example: <pre>measure.characterSet = @"UTF-8";</pre>
currencyCode	Default: none (value defined for report suite is used) The Currency Code used for purchases or currency events that are tracked in the iOS application. For example: <pre>measure.currencyCode = @"USD";</pre>
lifecycleSessionTimeout	Default: 5 minutes Specifies the length of time, in seconds, that must elapse between app launches before the launch is considered a new session. This timeout also applies when your application is sent to the background and reactivated. The time that your app spends in the background is not included in the session length. For example: <pre>measure.lifecycleSessionTimeout = 600; //10 minute session</pre>
ssl	Default: NO Enables (YES) or disables (NO) sending measurement data via SSL (HTTPS). You cannot override this variable for a single call, you must change the persistent value. For example: <pre>measure.ssl = YES;</pre>
debugLogging	Default: NO Enables (YES) or disables (NO) viewing debug information and the library version in the Xcode console. By default, this variable is NO. You cannot override this variable using variable overrides. For example: <pre>measure.debugLogging = YES;</pre>

Tracking Variables

Shared Instance

Before making measurement calls, you must retrieve the singleton instance of the library for each method you call on the measurement library:

```
ADMS_Measurement *measure = [ADMS_Measurement sharedInstance];
```



Note: For string variables, use only *NSStrings* (`@"value"`), not *C-Strings* (`"value"`).

Persistent Tracking Variables	Description
setEvar	<p>Sets the specified eVar to the provided value. The eVar is sent with the next tracking call.</p> <p>Syntax:</p> <pre>- (void)setEvar:(NSUInteger)evvarNum toValue:(NSString *)value;</pre> <p>Examples:</p> <pre>ADMS_Measurement *measure = [ADMS_Measurement getInstance]; [measure setEvar:1 toValue:@"value"];</pre>
setProp	<p>Sets the specified prop to the provided value. The prop is sent with next tracking call.</p> <p>Syntax:</p> <pre>- (void)setProp:(NSUInteger)propNum toValue:(NSString *)value;</pre> <p>Examples:</p> <pre>ADMS_Measurement *measure = [ADMS_Measurement getInstance]; [measure setProp:1 toValue:@"value"];</pre>
setHier	<p>Sets the specified hier variable to the provided value. The variable is sent with next tracking call.</p> <p>Syntax:</p> <pre>- (void)setHier:(NSUInteger)hierNum toValue:(NSString *)value;</pre> <p>Examples:</p> <pre>ADMS_Measurement *measure = [ADMS_Measurement getInstance]; [measure setHier:1 toValue:@"value"];</pre>
setListVar	<p>Sets the specified listvar to the provided value. The listvar is sent with next tracking call.</p> <p>Syntax:</p> <pre>- (void)setListVar:(NSUInteger)listNum toValue:(NSString *)value;</pre> <p>Examples:</p> <pre>ADMS_Measurement *measure = [ADMS_Measurement getInstance]; [measure setListVar:1 toValue:@"value"];</pre>
purchaseID	<p>The purchaseID is used to keep an order from being counted multiple times.</p>

Persistent Tracking Variables	Description
	<p>Whenever the purchase event is used on your site, you should assign this purchase a unique id using the purchaseID variable.</p> <pre>measure.purchaseID = @"unique_id";</pre>
transactionID	<p>Integration Data Sources use a transaction ID to tie offline data to an online transaction (like a lead or purchase generated online).</p> <p>Each unique transactionID sent to Adobe is recorded in preparation for a Data Sources upload of offline information about that transaction.</p> <pre>measure.transactionID = @"unique_id";</pre>
appState (page name)	<p>The value provided for the app state is stored in the page name variable.</p> <pre>measure.appState = @"state";</pre>
channel	<pre>measure.channel = @"channel";</pre>
appSection	<p>The value provided for the app section is stored in the server variable.</p> <pre>measure.appSection = @"section";</pre>
campaign	<pre>measure.campaign = @"112233";</pre>
products	<pre>measure.products = @"Running;Shoe"; //// example Products string: Category;Product[,Category;Product]</pre>
events	<pre>measure.events = @"event1, event2";</pre>
geoState	<pre>measure.geoState = @"UT";</pre>
geoZip	<pre>measure.geoZip = @"12345";</pre>
Persistent Context Data	<p>Context data is the recommended way to collect data. To send context data, create an <code>NSMutableDictionary</code> and assign key value pairs for the values you want to send. After the dictionary is populated, assign it to the measurement object's <code>persistentContextData</code>:</p> <pre>NSMutableDictionary *contextData = [NSMutableDictionary dictionary]; [contextData setObject:@"value" forKey:@"key"]; [contextData setObject:@"value2" forKey:@"key2"]; ADMS_Measurement *measure = [ADMS_Measurement getInstance]; measure.persistentContextData = contextData;</pre> <p>See Context Data.</p>
linkTrackVars	<p>A comma delimited list of variable names that restricts the current set of variables for link tracking.</p>

Persistent Tracking Variables	Description
	<p>If <code>linkTrackVars</code> is not defined, <code>AppMeasurement</code> for iOS sends all defined variables with a <code>trackLink</code> call.</p> <p>For example:</p> <pre>measure.linkTrackVars = @"eVar1, prop1";</pre>
<code>linkTrackEvents</code>	<p>A comma delimited list of events that restricts the current set of events for link tracking.</p> <p>If <code>linkTrackEvents</code> is not defined, <code>AppMeasurement</code> for iOS sends all events with a <code>trackLink</code> call.</p> <p>For example:</p> <pre>measure.linkTrackEvents = @"event1, event2";</pre>

Methods

This section contains a list of the methods provided by the iOS library.

Shared Instance

Before making measurement calls, you must retrieve the singleton instance of the library for each method you call on the measurement library:

```
ADMS_Measurement *measure = [ADMS_Measurement sharedInstance];
```

Initial Configuration

This method configures the required variables and must be called before other methods.

Configuration Methods	Description
configureMeasurementWithReportSuiteIDs	<p>This method is used to configure the two parameters required to start application measurement. You must set the <code>reportSuiteIDs</code> and <code>trackingServer</code> values on the measurement object using this method prior to sending any server calls.</p> <p>Syntax:</p> <pre>- (void)configureMeasurementWithReportSuiteIDs:(NSString *)reportSuiteIDs trackingServer:(NSString *)trackingServer;</pre> <p>Examples:</p> <pre>[measure configureMeasurementWithReportSuiteIDs:TRACKING_RSID trackingServer:TRACKING_SERVER];</pre>

Auto Tracking Configuration

Configuration Methods	Description
setAutoTrackingOptions	<p>By default, lifecycle events (launches, daily and monthly users, and so on) and navigation events (moving between different application views), are tracked automatically.</p> <pre>- (void)setAutoTrackingOptions:(ADMS_AutoTrackOptions)options;</pre> <p>Set options using the following values:</p> <pre>ADMS_AutoTrackOptionsNone = 0, ///< Disable all auto tracking ADMS_AutoTrackOptionsLifecycle = 1, ///< Track application lifecycle ADMS_AutoTrackOptionsNavigation = 2</pre> <p>To configure both lifecycle and navigation auto tracking, set the following:</p> <pre>[measure setAutoTrackingOptions:ADMS_AutoTrackOptionsNavigation ADMS_AutoTrackOptionsLifecycle];</pre> <p>To enable only automatic lifecycle event tracking and disable navigation event tracking:</p> <pre>[measure setAutoTrackingOptions:ADMS_AutoTrackOptionsLifecycle];</pre>

Configuration Methods	Description
	<p>To enable only automatic navigation event tracking and disable lifecycle event tracking:</p> <pre>[measure setAutoTrackingOptions:ADMS_AutoTrackOptionsNavigation];</pre> <p>To disable all automatic event tracking:</p> <pre>[measure setAutoTrackingOptions:ADMS_AutoTrackOptionsNone];</pre>

Event and State Tracking

These are the primary methods used to track custom events and app states.

Method	Description
trackAppState	<p>This is the recommended way to track application states (for example, pages) in your application. The value provided in <code>appState</code> appears as the page name variable of the server call. The <code>appState</code> string must be provided for each call since it isn't carried over to the next call.</p> <p>Context data sent with this call is appended to any values in <code>persistentContextData</code> and sent with the call. Only values set in <code>persistentContextData</code> remain for the next call.</p> <p>If you are using auto track, this method is called automatically each time a new view controller loads. To pass additional data with these auto track calls, you must set <code>persistentContextData</code> on the measurement object in the <code>viewDidLoad</code> method of your <code>UIViewController</code> instance.</p> <p>Syntax:</p> <pre>- (void)trackAppState:(NSString *)appState; - (void)trackAppState:(NSString *)appState withContextData:(NSDictionary *)contextData;</pre> <p>Examples:</p> <pre>[measure trackAppState:@"state name"]; NSMutableDictionary *contextData = [NSMutableDictionary dictionary]; [contextData setObject:@"value" forKey:@"custom key"]; //add additional key:value pairs to this dictionary... [measure trackAppState:@"state name" withContextData:contextData];</pre>
trackEvents	<p>This is the recommended way to track events in your application. <code>trackEvents</code> is similar to <code>trackAppState</code>, but sends events instead of page names. Events are provided as a comma delimited string. The <code>events</code> string must be provided for each call since it isn't carried over to the next call.</p> <p>This method make an underlying call to <code>trackLinkURL</code> where <code>linkURL</code> is set to <code>nil</code>, <code>linkType</code> is set to "o", and the <code>linkName</code> is populated with the application ID.</p> <p>This means that your <code>linkTrackVars</code> and <code>linkTrackEvents</code> apply for calls to <code>trackEvents</code>.</p>

Method	Description
	<p>Context data sent with this call is appended to any values in <code>persistentContextData</code> and sent with the call. Only values set in <code>persistentContextData</code> remain for the next call.</p> <p>Syntax:</p> <pre>- (void)trackEvents:(NSString *)eventNames; - (void)trackEvents:(NSString *)eventNames withContextData:(NSDictionary *)contextData;</pre> <p>Examples:</p> <pre>NSMutableDictionary *contextData = [NSMutableDictionary dictionary]; [contextData setObject:@"value" forKey:@"custom key"]; //add additional key:value pairs to this dictionary... [measure trackEvents:@"event1, event2" withContextData:contextData];</pre> <p>Import note if you are using the trackLink method</p> <p><code>trackEvents</code> makes an underlying call to <code>trackLinkURL</code>, where <code>linkURL</code> is set to <code>nil</code>, <code>linkType</code> is set to "o", and the <code>linkName</code> is populated with the application ID. This is done to prevent the page view (state view) count from being incremented each time you track events.</p> <p>If you are calling <code>trackLinkURL</code> directly and setting values for <code>linkTrackVars</code> and <code>linkTrackEvents</code>, these variable and event restrictions apply to <code>TrackEvents</code>. This means that only variables and events defined on these lists are sent with <code>TrackEvents</code>. You should set <code>linkTrackVars</code> and <code>linkTrackEvents</code> to <code>nil</code> unless you want those restrictions applied to <code>trackEvents</code>.</p>

Advanced Tracking (track and trackLink)

These methods provide additional tracking options, allowing you to populate props, eVars, and other implementation variables directly. These should be used by customers with an existing implementation or customers who are very familiar with other reporting implementations.

`track` and `trackLink` are the core measurement methods that are implemented in all versions of the Adobe Measurement Libraries across multiple platforms. In most circumstances when tracking mobile applications, it is easier to use `trackAppState`, `trackEvents` methods rather than calling `track` and `trackLink` directly.

Page view tracking (`track`) and link tracking (`trackLink`) sends all persistent variables that have values (non-nil, non-empty, non-zero). You should reset or empty all variables, as needed, before calling `track` or `trackLink`.



Note: Due to the multi-threaded nature of modern applications, setting persistent variable values to send with a future tracking call is not recommended (using `setEvar`, `setProp`, `setHier`, `setListVar`, and the `persistentContextData` variable). For example, if a variable value is set in multiple threads, the value might be in an inconsistent state when the tracking call is made. To avoid this, we recommend passing context data with each tracking call and setting the variable value in Processing Rules.

Method	Description
<code>track</code> <code>trackWithContextData</code>	Sends a standard page view, along with any additional variables that are set on the measurement object, to the collection server.

Method	Description
	<p>Each call to track sends all persistent data defined on the measurement object (these are listed in the description for <code>clearVars</code>), plus any <code>contextData</code> or variables you provide for that call only. If you send a variable that is defined, any value in the corresponding persistent variable is overwritten.</p> <p>Syntax:</p> <pre data-bbox="521 443 1338 541">- (void)track; - (void)trackWithContextData:(NSDictionary *)contextData; - (void)trackWithContextData:(NSDictionary *)contextData variables:(NSDictionary *)variables;</pre> <p>Examples:</p> <pre data-bbox="521 611 1471 1045">[measure track]; //set an eVar [measure setEvar:1 toValue:@"evar1value"]; //contextData NSMutableDictionary *contextData = [NSMutableDictionary dictionary]; [contextData setObject:@"value" forKey:@"custom key"]; [measure trackWithContextData:contextData]; //variable overrides NSMutableDictionary *variableOverrides = [NSMutableDictionary dictionary]; [variableOverrides setObject:@"evar2Value" forKey:@"eVar2"]; [measure trackWithContextData:contextData variables:variableOverrides];</pre>
trackLinkURL	<p>Sends custom, download or exit link data to Adobe data collection servers, along with any track config variables that have values.</p> <p>Use <code>trackLink</code> to track all activity that should not capture a page view.</p> <p>Syntax:</p> <pre data-bbox="521 1289 1284 1413">- (void)trackAppStattrackLinkURL:(NSString *)linkURL withLinkType:(NSString *)linkType linkName:(NSString *)linkName contextData:(NSDictionary *)contextData variables:(NSDictionary *)variables;</pre> <p>linkURL: Identifies the clicked URL. If no URL is specified, the name is used. Use this only when linking to a Web page from within your iOS application. Otherwise, pass in <code>nil</code> for this parameter.</p> <p>linkType: Identifies the link report that will display the URL or name. Supported values include:</p> <ul data-bbox="521 1650 764 1749" style="list-style-type: none"> • “o” (Custom Links) • “d” (File Downloads) • “e” (Exit Links) <p>linkName: The name that appears in the link report. If no name is specified, the report uses the URL.</p>

Method	Description
	<p>To collect data, you must specify either the <code>linkURL</code>, or <code>linkName</code> parameter. When not using one of these parameters, context data, or additional variables, pass in <code>nil</code> as the value.</p> <pre>NSMutableDictionary *contextData = [NSMutableDictionary dictionary]; [contextData setObject:@"value" forKey:@"custom key"]; NSMutableDictionary *variables = [NSMutableDictionary dictionary]; [variables setObject:@"eVar1Value" forKey:@"eVar1"]; [variables setObject:@"prop50Value" forKey:@"prop50"]; [measure trackLinkURL:@"linkURL" withLinkType:@"o" linkName:@"linkName" contextData:contextData variables:variables];</pre>
setEvar	<p>Sets the specified eVar to the provided value. The eVar is sent with the next tracking call.</p> <p>Syntax:</p> <pre>- (void)setEvar:(NSUInteger)evanum toValue:(NSString *)value;</pre> <p>Examples:</p> <pre>ADMS_Measurement *measure = [ADMS_Measurement getInstance]; [measure setEvar:1 toValue:@"value"];</pre>
setProp	<p>Sets the specified prop to the provided value. The prop is sent with next tracking call.</p> <p>Syntax:</p> <pre>- (void)setProp:(NSUInteger)propnum toValue:(NSString *)value;</pre> <p>Examples:</p> <pre>ADMS_Measurement *measure = [ADMS_Measurement getInstance]; [measure setProp:1 toValue:@"value"];</pre>
setHier	<p>Sets the specified hier variable to the provided value. The variable is sent with next tracking call.</p> <p>Syntax:</p> <pre>- (void)setHier:(NSUInteger)hiernum toValue:(NSString *)value;</pre> <p>Examples:</p> <pre>ADMS_Measurement *measure = [ADMS_Measurement getInstance]; [measure setHeir:1 toValue:@"value"];</pre>
setListVar	<p>Sets the specified listvar to the provided value. The listvar is sent with next tracking call.</p> <p>Syntax:</p> <pre>- (void)setListVar:(NSUInteger)listnum toValue:(NSString *)value;</pre> <p>Examples:</p> <pre>ADMS_Measurement *measure = [ADMS_Measurement getInstance]; [measure setListVar:1 toValue:@"value"];</pre>

Method	Description
setPersistentContextData	<p>Values placed in Persistent Context Data are sent with each server call. To send context data for a single call, send a contextData hashtable as a parameter to the tracking call (trackAppState, trackEvents, and so on).</p> <pre>Hashtable contextData = new Hashtable<String, Object>(); contextData.put("key", "value"); measure.setPersistentContextData(contextData);</pre> <p>See Context Data.</p>
clearVars	<p>Removes values from the following variables on the object:</p> <pre>evars props listvars hiers events purchaseID transactionID appState channel appSection campaign products persistentContextData geoState geoZip linkTrackVars linkTrackEvents</pre> <p>Syntax:</p> <pre>- (void)clearVars;</pre> <p>Examples:</p> <pre>[measure clearVars];</pre>

Offline Tracking Reference



Note: To enable offline AppMeasurement, your report suite must be timestamp-enabled.

The following variables let you store measurement calls when the application is offline. After you enable offline tracking, all hits must be time-stamped or they are dropped. If you are currently reporting AppMeasurement data to a report suite that also collects data from JavaScript, you might need to set up a separate report suite for Offline AppMeasurement to avoid data loss.

When enabled, Offline AppMeasurement behaves in the following way:

- The application sends a server call, but the data transmission fails.
- AppMeasurement generates a timestamp for the current hit.
- AppMeasurement buffers the hit data, and backs up buffered hit data to persistent storage to prevent data loss.

At each subsequent hit, or at the interval defined by `offlineThrottleDelay`, AppMeasurement attempts to send the buffered hit data, maintaining the original hit order. If the data transmission fails, it continues to buffer the hit data (This continues while the device is offline).

Property or Method	Description
offlineTrackingEnabled	<p>Default: NO</p> <p>Enables or disables offline tracking for the measurement library.</p> <p>Syntax:</p> <pre>BOOL offlineTrackingEnabled;</pre> <p>Examples:</p> <pre>measure.offlineTrackingEnabled = YES;</pre>
offlineHitLimit	<p>Default: 1000</p> <p>Maximum number of offline hits stored in the queue. If the hit limit is set over 5000, performance may suffer.</p> <p>Syntax:</p> <pre>NSUInteger offlineHitLimit;</pre> <p>Examples:</p> <pre>measure.offlineHitLimit = 2000;</pre>
trackingQueueSize	<p>Returns the number of stored tracking calls in the offline queue.</p> <p>Syntax:</p> <pre>-(NSUInteger)trackingQueueSize;</pre> <p>Examples:</p> <pre>[measure trackingQueueSize];</pre>
clearTrackingQueue	<p>Removes all stored tracking calls from the offline queue.</p> <p>Syntax:</p> <pre>-(void)clearTrackingQueue;</pre> <p>Examples:</p> <pre>[measure clearTrackingQueue];</pre> <p>Warning: use caution when clearing the queue manually as it cannot be reversed.</p>
setOnline setOffline	<p>Manually set the online or offline state of the measurement object. The library automatically detects when the device is offline or online, so these methods are needed only if you want to force measurement offline. <code>setOnline</code> is used only to return to the online state after manually going offline.</p> <p>When measurement is offline:</p> <ul style="list-style-type: none"> • If <code>offlineTrackingEnabled</code> is true: hits are stored until measurement is online. • If <code>offlineTrackingEnabled</code> is false: hits are discarded. <p>Syntax:</p> <pre>-(void)setOnline; (void)setOffline;</pre>

Property or Method	Description
	Examples: <pre>[measure setOffline]; [measure setOnline];</pre>

Offline Tracking

AppMeasurement lets you measure application usage even when the iOS device is offline.



Note: *To enable offline AppMeasurement, your report suite must be timestamp-enabled.*

See [Offline Tracking Reference](#).

Target

Adobe provides the ability to deliver targeted content within iOS applications.

The content returned from Target can be any text-based content such as HTML, XML or plain text. Once the content is loaded, it is then up to the iOS application developer to decide how the content is used within the application. The content can be displayed or used to change the behavior of the application. This integration guide provides a simple use example of the `Mbox` and `MboxFactory` classes as well as a detailed API for the two classes.

Get the Library

Visit https://developer.omniture.com/en_US/content_module/mobile to download the iOS library.

After unzipping the download file, you'll have the following software components:

- `ADMS_Measurement.h`: The Objective-C header file used for iOS AppMeasurement.
- `admsAppLibrary.a`: a fat binary containing the library builds for devices (armv7 and armv7x) as well as the simulator (i386/x86_64). Requires iOS 4.3 or later.
- `Examples/TrackingHelper.h`, `Examples/TrackingHelper.m`: Optional class that is designed to keep tracking code separate from your application logic.

Add the Library to your Project

1. Launch the Xcode IDE.
2. Copy the `ADMS_AppLibrary` folder from the `ADMS_AppLibrary-*.DMG` you downloaded to your project folder or another location on your drive.
3. Drag and Drop the `ADMS_Measurement` folder on your Xcode project, and confirm the following settings:
 - Select **Copy items into destination group's folder (if needed)**.
 - Select **Create groups for any added folders**.
 - Select the targets where you want to use AppMeasurement code.
4. Click **Finish**.

Code Sample

After you add the library to your project, you can use the two Test&Target classes, `MboxFactory` and `Mbox`.

This example provides a basic demo of how to load HTML content from Test&Target into a simple View-based iPhone application. This example assumes that the desired HTML offers and campaign have already been created within the Test&Target Admin Tool, and that the campaign is approved.

1. Add the following code to the `viewDidLoad` event handler of your custom `UIViewController` (that implements the `MboxContentConsumerDelegate` category) implementation file (see comments for an explanation of each line of code). To complete this step, you will need to know your client code and the name of the `mbox` used in the campaign setup in the Test&Target Admin Tool.

```
#import "ADMS_MboxFactory.h"
// Create an MboxFactory and specify the client code.
ADMS_MboxFactory *factory = [[ADMS_MboxFactory alloc] initWithClientCode: @"iphonedemo16"];

// Use the MboxFactory to create the Mbox.
ADMS_Mbox *mbox = [factory create: @"iPod_Nano" delegate: self];

// Specify the string to be used as default content.
mbox.defaultContent = [NSString stringWithString:
    @"<div style='text-align:center'><img border='0' alt='iPod Nano' "
    "src='http://testandtargeting.com/iphone/default.jpg' /></div>"];
```

```
//load the Mbox.
[mbox load];
}
```

2. Define the function (required by MboxContentConsumerDelegate) to display the HTML in a UIWebView object created in Interface Builder once its finished loading from Test&Target. This method can also be defined in a separate class file if user-defined class is used as the delegate parameter for the factory's create method.

```
-(void) consume: (NSString *) content {
    //Create a URL object.
    NSString *urlAddress = @"http://www.client.com";
    NSURL *baseURL = [NSURL URLWithString:urlAddress];
```

3. Build and run the application, and if the client code and mbox are correct and if the campaign is approved, you should see the Test&Target content displayed on the iPhone simulator.

```
// Copyright 2012 Adobe Systems Incorporated. All rights reserved.

#import "ADMS_Measurement.h"

// Implement viewDidLoad to do additional setup after loading the view, typically from a nib.
- (void)viewDidLoad {
    [super viewDidLoad];

    // Create an MboxFactory and specify the client code.
    ADMS_MboxFactory *factory = [[ADMS_MboxFactory alloc] initWithClientCode: @"iphonedemo16"];

    // Use the MboxFactory to create the Mbox.
    ADMS_Mbox *mbox = [factory create: @"iPod_Nano" delegate: self];

    // Specify the string to be used as default content.
    mbox.defaultContent = [NSString stringWithString:
        @"<div style='text-align:center'><img border='0' alt='iPod Nano' "
        "src='http://testandtargeting.com/iphone/default.jpg' /></div>"];

    //load the Mbox.
    [mbox load];
}

-(void) consume: (NSString *) content {
    //Create a URL object.
    NSString *urlAddress = @"http://www.client.com";
    NSURL *baseURL = [NSURL URLWithString:urlAddress];

    [webView loadHTMLString:content baseURL:(NSURL *)baseURL];
}
```

Lifecycle Metrics

If lifecycle metrics are enabled, lifecycle metrics are sent as parameters to each mbox load.

- [\(Recommended\) Track Lifecycle Events](#)
- [Lifecycle Metrics](#)

ADMS_MboxFactory Reference

Methods


Method	Description
create:delegate:	<pre>- (ADMS_Mbox *) create: (NSString *) mboxName delegate: (id< MboxContentConsumerDelegate >) delegate</pre> <p>Creates new Mbox.</p> <p>mboxName: MBox name to create</p> <p>delegate: callback delegate for mbox.</p>
initWithClientCode:	<pre>- (id) initWithClientCode: (NSString *) client</pre> <p>client: Client code for which to create the Factory.</p>
recordEvent:	<pre>- (void) recordEvent: (NSString *) mboxName</pre> <p>mboxName: MBox name to record impression on.</p> <p>Records impression for a given Mbox. This is useful for recording user actions such as clicks on a particular button.</p>
useFirstPartyCookieOnly:mboxPC:	<pre>- (void) useFirstPartyCookieOnly: (NSString *) mboxSession mboxPC: (NSString *) mboxPC</pre> <p>Forces the factory to use only first-party cookie information for making requests to Test&Target.</p> <p>mboxSession: The session ID to use for the visitor.</p> <p>mboxPC: The PC ID to use for the visitor.</p>

Properties

Property	Type	Description
clientCode	NSString *	T&T Client Code for this factory.
mboxServerURL	NSString *	Mbox URL.
debugLogging	BOOL	Enable/disable debug logging.
isEnabled	BOOL	Enable/disable this factory.

ADMS_Mbox Reference

Methods

Method	Description
addMboxParameter:paramValue:	<pre>- (void) addMboxParameter: (NSString *) paramName paramValue: (NSString *) paramValue</pre> <p>Adds an additional parameter value to the mbox request.</p> <p>paramName: parameter name</p> <p>paramValue: paramter value</p>
load	<pre>- (void) load</pre> <p>Loads mbox content.</p> <p> Note: Delegate methods will be called based on the success/failure of this method.</p>

Properties

Property	Type	Description
maxResponseTime	NSTimeInterval	Max number of seconds to wait for a response before returning default content. Default is 2.0.
defaultContent	NSString *	Default content to return when an offer is not available.
name	NSString *	Mbox Name.
parentFactory	ADMS_MboxFactory *	Mbox factory.
delegate	id< MboxContentConsumerDelegate >	delegate

Audience Management

Adobe provides the ability to send signals and retrieve visitor segments from audience management.

Get the Library

Visit https://developer.omniture.com/en_US/content_module/mobile to download the iOS library.

After unzipping the download file, you'll have the following software components:

- `ADMS_Measurement.h`: The Objective-C header file used for iOS AppMeasurement.
- `admsAppLibrary.a`: a fat binary containing the library builds for devices (armv7 and armv7x) as well as the simulator (i386/x86_64). Requires iOS 4.3 or later.
- `Examples/TrackingHelper.h`, `Examples/TrackingHelper.m`: Optional class that is designed to keep tracking code separate from your application logic.

Add the Library to your Project

1. Launch the Xcode IDE.
2. Copy the `ADMS_AppLibrary` folder from the `ADMS_AppLibrary-*.DMG` you downloaded to your project folder or another location on your drive.
3. Drag and Drop the `ADMS_Measurement` folder on your Xcode project, and confirm the following settings:
 - Select **Copy items into destination group's folder (if needed)**.
 - Select **Create groups for any added folders**.
 - Select the targets where you want to use AppMeasurement code.
4. Click **Finish**.

Code Sample

After you add the library to your project, you can use the `ADMS_AudienceManager` class.

1. Configure audience management:

```
[ADMS_AudienceManager configureAudienceManager:@"mycompany.demdex.net"];
```

Replace `mycompany.demdex.net` with your endpoint. Audience management can be configured at any point in your application.

2. Optionally set the `dpid` and `dpuuid`.

```
[ADMS_AudienceManager setDpid:@"284" dpuuid:@"abc123"];
```

3. Create a trait dictionary and submit the dictionary in a signal.

```
NSDictionary *traits = @{@"trait":@"b"};  
[ADMS_AudienceManager submitSignal:traits callback:^(NSDictionary *content) {  
    // do something with content  
}];
```

The visitor profile dictionary is returned in the callback as the `content` parameter.

Lifecycle Metrics

If lifecycle metrics are enabled and audience management is configured in `application:didFinishLaunchingWithOptions:`, a signal containing lifecycle metrics is sent after configuration completes.

- [\(Recommended\) Track Lifecycle Events](#)
- [Lifecycle Metrics](#)

ADMS_AudienceManager Reference**Methods**

Method	Description
configureAudienceManager:	<p>Sets the audience management endpoint.</p> <p>Syntax:</p> <pre>+ (void) configureAudienceManager:(NSString *)server;</pre>
dpid	<p>Returns the dpid.</p> <p>Syntax:</p> <pre>+ (NSString *) dpid;</pre>
dpuuid	<p>Returns the dpuuid.</p> <p>Syntax:</p> <pre>+ (NSString *) dpuuid;</pre>
isConfigured	<p>Returns YES if audience management is configured.</p> <p>Syntax:</p> <pre>+ (bool) isConfigured;</pre>
setDpid:dpuuid:	<p>If dpid and dpuuid are set, they will be sent with each signal.</p> <p>Syntax:</p> <pre>+ (void) setDpid:(NSString *)newDpid dpuuid:(NSString *)newDpuuid;</pre>
submitSignal:callback:	<p>Sends in a dictionary of traits and receives the updated visitor profile in callback.</p> <p>The uuid from the JSON response is stored internally and used with all subsequent signals. A pixel request is also sent to each URL found in the <code>dests</code> object.</p> <p>Syntax:</p> <pre>+ (void) submitSignal:(NSDictionary *)data callback:(void(^)(NSDictionary *)callback);</pre>
visitorProfile	<p>This method can be called at any time after you've submitted a signal to retrieve the most recent visitor profile.</p> <p>The visitor profile contains all of the key value pairs that were returned in the <code>stuff</code> object.</p> <p>Syntax:</p> <pre>+ (NSDictionary *) visitorProfile;</pre>

PhoneGap Plug-in

This plug-in lets you send iOS AppMeasurement calls from your PhoneGap project.

For help creating a PhoneGap project, see [PhoneGap Getting Started with iOS](#).

To download the plug-in, see [PhoneGap iOS and Android Plug-ins for Analytics](#).

Include the Plug-in

1. Copy `ADMS_Measurement_PhoneGap.m` and `ADMS_Measurement_PhoneGap.h` to the `Plugins` folder in your PhoneGap project.
2. Copy `ADMS_Helper.js` to the `www/js` folder in your PhoneGap project.
3. In `config.xml`, add `<plugin name="ADMS_Plugin" value="ADMS_Measurement_PhoneGap" />` as a child under `plugins`.

Include the AppMeasurement Library

To download the AppMeasurement library, see [Get the Library](#).

1. Launch the Xcode IDE.
2. Copy the `ADMS_AppLibrary` folder from the `ADMS_AppLibrary-* .DMG` you downloaded to your project folder or another location on your drive.
3. Drag and Drop the `ADMS_Measurement` folder on your Xcode project, and confirm the following settings:
 - Select **Copy items into destination group's folder (if needed)**.
 - Select **Create groups for any added folders**.
 - Select the targets where you want to use AppMeasurement code.
4. Click **Finish**.

Lifecycle Metrics Auto Tracking

Tracking the [Lifecycle Metrics](#) requires configuration outside of PhoneGap. To enable Lifecycle Auto Tracking, complete the [Implementation](#) steps in the Developer Quick Start.

Use the Plug-in

In `html` files where you want to use tracking, include:

```
<script type="text/javascript" src="js/ADMS_Helper.js"></script>
```

That's it, you are now ready to make measurement calls. See [PhoneGap Plug-in Methods](#).

Troubleshooting

My syntax is correct, why is the code in the plugin not getting reached?

Check your output console for an error that is similar to: `ERROR: Plugin 'ADMS_Plugin' not found, or is not a CDVPlugin`

if this error occurs, make sure you performed step 3 in Include the Plug-in.

PhoneGap Plug-in Methods

In html files where you want to use these methods, include:

```
<script type="text/javascript" src="js/ADMS_Helper.js"></script>
```

Configuration Methods

Method	Description
configureMeasurementWithReportSuiteIDsTrackingServer	<p>This method is used to configure the two parameters required to start application measurement. You must set the <code>reportSuiteIDs</code> and <code>trackingServer</code> values on the measurement object using this method prior to sending any server calls.</p> <p>Syntax:</p> <pre>void configureMeasurementWithReportSuiteIDsTrackingServer(string reportSuiteIDs, string trackingServer);</pre> <p>Example:</p> <pre>ADMS.configureMeasurementWithReportSuiteIDsTrackingServer("testRSID", "10.20.40.199:5050");</pre>
setOnline setOffline	<p>Manually set the online or offline state of the measurement object. The library automatically detects when the device is offline or online, so these methods are needed only if you want to force measurement offline. SetOnline is used only to return to the online state after manually going offline.</p> <p>When measurement is offline:</p> <ul style="list-style-type: none"> • If <code>offlineTrackingEnabled</code> is true: hits are stored until measurement is online. • If <code>offlineTrackingEnabled</code> is false: hits are discarded. <p>Syntax:</p> <pre>void setOnline(); void setOffline();</pre> <p>Example:</p> <pre>ADMS.setOnline(); ADMS.setOffline();</pre>
setDebugLogging	<p>Enables (true) or disables (false) viewing debug information. By default, this variable is false.</p> <p>You cannot override this variable using variable overrides.</p> <p>Syntax:</p> <pre>void setDebugLogging(bool debugLogging);</pre> <p>Example:</p> <pre>ADMS.setDebugLogging(true);</pre>

Event and State Tracking

Method	Description
trackAppState	<p>This is the recommended way to track application states (for example, pages) in your application. The value provided in <code>appState</code> appears as the page name variable of the server call. The <code>appState</code> string must be provided for each call since it isn't carried over to the next call.</p> <p>Syntax:</p> <pre>void trackAppState(string stateName);</pre> <p>Example:</p> <pre>ADMS.trackAppState("login page");</pre>
trackAppStateWithContextData	<p>This is the recommended way to track application states (for example, pages) in your application. The value provided in <code>appState</code> appears as the page name variable of the server call. The <code>appState</code> string must be provided for each call since it isn't carried over to the next call.</p> <p>Context data sent with this call is appended to any values in <code>persistentContextData</code> and sent with the call. Only values set in <code>persistentContextData</code> remain for the next call.</p> <p>Syntax:</p> <pre>void trackAppStateWithContextData(string stateName, JSON cData);</pre> <p><code>cData</code>: JSON object with key-value pairs to send in context data.</p> <p>Example:</p> <pre>trackAppStateWithContextData("login page", {"user": "john", "remember": "true"});</pre>
trackEvents	<p>This is the recommended way to track events in your application. <code>trackEvents</code> is similar to <code>trackAppState</code>, but sends events instead of page names. Events are provided as a comma delimited string. The <code>events</code> string must be provided for each call since it isn't carried over to the next call.</p> <p>This method make an underlying call to <code>trackLinkURL</code> where <code>linkURL</code> is set to <code>null</code>, <code>linkType</code> is set to "o", and the <code>linkName</code> is populated with the application ID.</p> <p>This means that your <code>linkTrackVars</code> and <code>linkTrackEvents</code> apply for calls to <code>trackEvents</code>.</p> <p>Syntax:</p> <pre>void trackEvents(string eventNames);</pre> <p>Example:</p> <pre>ADMS.trackEvents("login,event2");</pre> <p>Import note if you are using the trackLink method</p> <p><code>trackEvents</code> makes an underlying call to <code>trackLinkURL</code>, where <code>linkURL</code> is set to <code>null</code>, <code>linkType</code> is set to "o", and the <code>linkName</code> is populated with the application ID. This is done</p>

Method	Description
	<p>to prevent the page view (state view) count from being incremented each time you track events.</p> <p>If you are calling <code>trackLinkURL</code> directly and setting values for <code>linkTrackVars</code> and <code>linkTrackEvents</code>, these variable and event restrictions apply to <code>TrackEvents</code>. This means that only variables and events defined on these lists are sent with <code>TrackEvents</code>. You should set <code>linkTrackVars</code> and <code>linkTrackEvents</code> to null unless you want those restrictions applied to <code>trackEvents</code>.</p>
trackEventsWithContextData	<p>This is the recommended way to track events in your application. <code>trackEvents</code> is similar to <code>trackAppState</code>, but sends events instead of page names. Events are provided as a comma delimited string. The <code>events</code> string must be provided for each call since it isn't carried over to the next call.</p> <p>This method make an underlying call to <code>trackLinkURL</code> where <code>linkURL</code> is set to null, <code>linkType</code> is set to "o", and the <code>linkName</code> is populated with the application ID.</p> <p>This means that your <code>linkTrackVars</code> and <code>linkTrackEvents</code> apply for calls to <code>trackEvents</code>.</p> <p>Context data sent with this call is appended to any values in <code>persistentContextData</code> and sent with the call. Only values set in <code>persistentContextData</code> remain for the next call.</p> <p>Syntax:</p> <pre>void trackEventsWithContextData(string eventNames, JSON cData);</pre> <p><code>cData</code>: JSON object with key-value pairs to send in context data.</p> <p>Example:</p> <pre>ADMS.trackEventsWithContextData("login,event2", {"user":"john","remember":"true"});</pre> <p>Import note if you are using the trackLink method</p> <p><code>trackEvents</code> makes an underlying call to <code>trackLinkURL</code>, where <code>linkURL</code> is set to null, <code>linkType</code> is set to "o", and the <code>linkName</code> is populated with the application ID. This is done to prevent the page view (state view) count from being incremented each time you track events.</p> <p>If you are calling <code>trackLinkURL</code> directly and setting values for <code>linkTrackVars</code> and <code>linkTrackEvents</code>, these variable and event restrictions apply to <code>TrackEvents</code>. This means that only variables and events defined on these lists are sent with <code>TrackEvents</code>. You should set <code>linkTrackVars</code> and <code>linkTrackEvents</code> to null unless you want those restrictions applied to <code>trackEvents</code>.</p>

Advanced Tracking Methods (track and trackLink)

These methods provide additional tracking options, allowing you to populate props, eVars, and other Analytics variables directly. These should be used by customers with an existing implementation or customers who are very familiar with other Analytics implementations.

`track` and `trackLink` are the core measurement methods that are implemented in all versions of the Adobe Measurement Libraries across multiple platforms. In most circumstances when tracking mobile applications, it is easier to use `trackAppState`, `trackEvents` methods rather than calling `track` and `trackLink` directly.

Page view tracking (`track`) and link tracking (`trackLink`) sends all persistent variables that have values (non-null, non-empty, non-zero). You should reset or empty all variables, as needed, before calling `track` or `trackLink`.

Method	Description
<code>track</code>	<p>Sends a standard page view, along with any additional variables that are set on the measurement object, to the collection server.</p> <p>Syntax:</p> <pre>void track();</pre> <p>Example:</p> <pre>ADMS.track();</pre>
<code>trackWithContextData</code>	<p>Sends a standard page view, along with any additional variables that are set on the measurement object, to the collection server.</p> <p>Each call to <code>trackWithContextData</code> sends all persistent data defined on the measurement object (these are listed in the description for <code>clearVars</code>), plus any <code>contextData</code> you provide for that call only.</p> <p>Syntax:</p> <pre>void trackWithContextData(JSON cData);</pre> <p><code>cData</code>: JSON object with key-value pairs to send in context data.</p> <p>Example:</p> <pre>ADMS.trackWithContextData({ "key1": "value1", "key2": "value2" });</pre>
<code>trackWith ContextDataAndVariables</code>	<p>Sends a standard page view, along with any additional variables that are set on the measurement object, to the collection server.</p> <p>Each call to <code>trackWith ContextDataAndVariables</code> sends all persistent data defined on the measurement object (these are listed in the description for <code>clearVars</code>), plus any <code>contextData</code> and variables you provide for that call only. If you send a variable that is defined, any value in the corresponding persistent variable is overwritten.</p> <p>Syntax:</p> <pre>void trackWith ContextDataAndVariables(JSON cData, JSON vars);</pre> <p><code>cData</code>: JSON object with key-value pairs to send in context data.</p> <p>Example:</p> <pre>ADMS.trackWith ContextDataAndVariables({ "key1": "value1", "key2": "value2" }, { "eVar1": "newValue", "prop3": "newValue" });</pre>
<code>trackLinkURLWithLinkTypeName ContextDataVariables</code>	<p>Sends custom, download or exit link data to Adobe data collection servers, along with any track config variables that have values.</p>

Method	Description
	<p>Use <code>trackLink</code> to track all activity that should not capture a page view.</p> <p>Each call to <code>trackLinkURLWithLinkTypeNameContextDataVariables</code> sends all persistent data defined on the measurement object (these are listed in the description for <code>clearVars</code>), plus any <code>contextData</code> you provide for that call only.</p> <p>Syntax:</p> <pre>void trackLinkURLWithLinkTypeNameContextDataVariables(string linkUrl, string linkType, string linkName, JSON cData, JSON vars);</pre> <p><code>cData</code>: JSON object with key-value pairs to send in context data.</p> <p>Example:</p> <pre>ADMS.trackLinkURLWithLinkTypeNameContextDataVariables("theLink", "o", "logout", {"key1":"value1"}, {"eVar1":"newValue"});</pre>

Set and Get AppMeasurement Variables

Method	Description
<code>setEvarToValue</code>	<p>Sets the specified eVar to the provided value. The eVar is sent with the next tracking call.</p> <p>Syntax:</p> <pre>void setEvarToValue(int evar, string value);</pre> <p>Example:</p> <pre>ADMS.setEvarToValue(1, "newValue");</pre>
<code>setPropToValue</code>	<p>Sets the specified prop to the provided value. The prop is sent with next tracking call.</p> <p>Syntax:</p> <pre>void setPropToValue(int prop, string value);</pre> <p>Example:</p> <pre>ADMS.setPropToValue(1, "newValue");</pre>
<code>setHierToValue</code>	<p>Sets the specified hier variable to the provided value. The variable is sent with next tracking call.</p> <p>Syntax:</p> <pre>void setHierToValue(int hier, string value);</pre> <p>Example:</p> <pre>ADMS.setHierToValue(1, "newValue");</pre>
<code>setListVarToValue</code>	<p>Sets the specified listvar to the provided value. The listvar is sent with next tracking call.</p> <p>Syntax:</p> <pre>void setListVarToValue(int list, string value);</pre>

Method	Description
	<p>Example:</p> <pre>ADMS.setListVarToValue(1, "newValue");</pre>
getEvar	<p>Gets the specified variable.</p> <p>Syntax:</p> <pre>void getEvar(int evar, function success, function fail);</pre> <p>Example:</p> <pre>ADMS.getEvar(1, function (value) { myTempEvar1 = value; }, function () { myTempEvar1 = null; });</pre>
getProp	<p>Gets the specified variable.</p> <p>Syntax:</p> <pre>void getProp(int prop, function success, function fail);</pre> <p>Example:</p> <pre>ADMS.getProp(1, function (value) { myTempProp1 = value; }, function () { myTempProp1 = null; });</pre>
getHier	<p>Gets the specified variable.</p> <p>Syntax:</p> <pre>void getHier(int hier, function success, function fail);</pre> <p>Example:</p> <pre>ADMS.getHier(1, function (value) { myTempHier1 = value; }, function () { myTempHier1 = null; });</pre>
getListVar	<p>Gets the specified variable.</p> <p>Syntax:</p> <pre>void getListVar(int list, function success, function fail);</pre> <p>Example:</p> <pre>ADMS.getListVar(1, function (value) { myTempListVar1 = value; }, function () { myTempListVar1 = null; });</pre>
clearVars	<p>Removes values from the following variables on the object:</p> <pre>props eVars heirs listVars events PersistentContextData purchaseID transactionID appState channel appSection campaign</pre>

Method	Description
	products geoZip geoState linkTrackVars linkTrackEvents Syntax: <pre>void clearVars();</pre> Example: <pre>ADMS.clearVars();</pre>
trackingQueueSize	Gets or sets the number of stored tracking calls in the offline queue. Syntax: <pre>void trackingQueueSize(function success, function fail);</pre> Example: <pre>ADMS.trackingQueueSize(function (value) { myQueueSize = value; }, function () { myQueueSize = 0; });</pre>
clearTrackingQueue	Removes all stored tracking calls from the offline queue. Warning: use caution when clearing the queue manually as it cannot be reversed. Syntax: <pre>void clearTrackingQueue();</pre> Example: <pre>ADMS.clearTrackingQueue();</pre>

Set and Get Configuration Variables

Method	Description
getVersion	Gets the library version. Syntax: <pre>void getVersion(function success, function fail);</pre> Example: <pre>ADMS.getVersion(function (value) { versionNum = value; }, function () { versionNum = 1.0; });</pre>
setReportSuiteIDs	<p>(Required) The report suite or report suites (multi-suite tagging) that you wish to track. To track to multiple report suites, pass a comma delimited list of the names of each report suite. You cannot override this variable for a single call, you must change the persistent value.</p> Syntax: <pre>void setReportSuiteIDs(string reportSuiteIDs);</pre>

Method	Description
	<p>Example:</p> <pre>ADMS.setReportSuiteIDs("rsid1, rsid2");</pre>
setTrackingServer	<p>(Required) Identifies the collection server.</p> <p>This variable should be populated with the value generated for you in Code Manager.</p> <p>The same value is used for secure tracking if ssl is enabled.</p> <p>You cannot override this variable for a single call, you must change the persistent value.</p> <p>Syntax:</p> <pre>void setTrackingServer(string trackingServer);</pre> <p>Example:</p> <pre>ADMS.setTrackingServer("10.23.52.191:5923");</pre>
setDataCenter	<p>Syntax:</p> <pre>void setDataCenter(string dataCenter);</pre> <p>Example:</p> <pre>ADMS.setDataCenter("myDataCenter");</pre>
setVisitorID	<p>Default: CFUUID</p> <p>Unique identifier for each visitor. If you do not set a custom visitorID, a unique visitorID is generated (using Apple's CFUUID) on initial launch and then stored in a user defaults key. This key is used by AppMeasurement and PhoneGap from that point forward. This key is also saved and restored during the standard application backup process.</p> <p>Syntax:</p> <pre>void setVisitorID(string visitorId);</pre> <p>Example:</p> <pre>ADMS.setVisitorID("myVisitorId");</pre>
setCharSet	<p>Default: UTF-8</p> <p>Syntax:</p> <pre>void setCharSet(string charSet);</pre> <p>Example:</p> <pre>ADMS.setCharSet("UTF-8");</pre>
setCurrencyCode	<p>Default: none (value defined for report suite is used)</p> <p>The Currency Code used for purchases or currency events that are tracked in the iOS application.</p>

Method	Description
	<p>Syntax:</p> <pre>void setCurrencyCode(string currencyCode);</pre> <p>Example:</p> <pre>ADMS.setCurrencyCode("USD");</pre>
setSSLEnabled	<p>Default: false</p> <p>Enables (true) or disables (false) sending measurement data via SSL (HTTPS).</p> <p>You cannot override this variable for a single call, you must change the persistent value.</p> <p>Syntax:</p> <pre>void setSSLEnabled(bool sslEnabled);</pre> <p>Example:</p> <pre>ADMS.setSSLEnabled(false);</pre>

Set and Get Persistent Tracking Variables

Method	Description
setPurchaseID	<p>Syntax:</p> <pre>void setPurchaseID(string purchaseId);</pre> <p>Example:</p> <pre>ADMS.setPurchaseID("purchase1");</pre>
setTransactionID	<p>Syntax:</p> <pre>void setTransactionID(string transactionId);</pre> <p>Example:</p> <pre>ADMS.setTransactionID("transaction1");</pre>
setAppState	<p>Syntax:</p> <pre>void setAppState(string stateName);</pre> <p>Example:</p> <pre>ADMS.setAppState("myAppState");</pre>
setChannel	<p>Syntax:</p> <pre>void setChannel(string channel);</pre> <p>Example:</p> <pre>ADMS.setChannel("myChannel");</pre>

Method	Description
setAppSection	<p>Syntax:</p> <pre>void setAppSection(string appSection);</pre> <p>Example:</p> <pre>ADMS.setAppSection("myAppSection");</pre>
setCampaign	<p>Syntax:</p> <pre>void setCampaign(string campaign);</pre> <p>Example:</p> <pre>ADMS.setCampaign("myCampaign");</pre>
setProducts	<p>Syntax:</p> <pre>void setProducts(string products);</pre> <p>Example:</p> <pre>ADMS.setProducts("myProduct1,myProduct2");</pre>
setEvents	<p>Syntax:</p> <pre>void setEvents(string events);</pre> <p>Example:</p> <pre>ADMS.setEvents("event1, event2");</pre>
setGeoState	<p>Syntax:</p> <pre>void setGeoState(string state);</pre> <p>Example:</p> <pre>ADMS.setGeoState("FL");</pre>
setGeoZip	<p>Syntax:</p> <pre>void setGeoZip(string zip);</pre> <p>Example:</p> <pre>ADMS.setGeoZip("39984");</pre>
setPersistentContextData	<p>Context data is the recommended way to collect data. To send context data, create a JSON object and assign key value pairs for the values you want to send.</p> <p>Syntax:</p> <pre>void setPersistentContextData(JSON cData);</pre> <p>Example:</p> <pre>ADMS.setPersistentContextData({"key1": "value1"});</pre>

Method	Description
persistentContextData	<p>Syntax:</p> <pre>void persistentContextData(function success, function fail);</pre> <p>Example:</p> <pre>ADMS.persistentContextData(function(value) { alert(value); }, function(error) { alert(error); });</pre>
setLinkTrackVars	<p>A comma delimited list of variable names that restricts the current set of variables for link tracking.</p> <p>If <code>linkTrackVars</code> is not defined, the PhoneGap plug-in sends all defined variables with a <code>trackLink</code> call.</p> <p>Syntax:</p> <pre>void setLinkTrackVars(string linkTrackVars);</pre> <p>Example:</p> <pre>ADMS.setLinkTrackVars("eVar1,eVar2");</pre>
setLinkTrackEvents	<p>A comma delimited list of events that restricts the current set of events for link tracking.</p> <p>If <code>linkTrackEvents</code> is not defined, the PhoneGap plug-in sends all events with a <code>trackLink</code> call.</p> <p>Syntax:</p> <pre>void setLinkTrackEvents(string linkTrackEvents);</pre> <p>Example:</p> <pre>ADMS.setLinkTrackEvents("event1,loginEvent");</pre>
setLightTrackVars	<p>A comma delimited list of variables that restricts the current set of variables for light tracking calls. This variables you send with a light server call are configured by Customer Care.</p> <p>Syntax:</p> <pre>void setLightTrackVars(string lightTrackVars);</pre> <p>Example:</p> <pre>ADMS.setLightTrackVars("prop1,hier3");</pre>

Offline Tracking

Method	Description
setOfflineTrackingEnabled	<p>Syntax:</p> <pre>void setOfflineTrackingEnabled(bool offlineTrackingEnabled);</pre> <p>Example:</p> <pre>ADMS.setOfflineTrackingEnabled(true);</pre>

Method	Description
setOfflineHitLimit	<p>Syntax:</p> <pre data-bbox="480 289 1469 321">void setOfflineHitLimit(int offlineHitLimit);</pre> <p>Example:</p> <pre data-bbox="480 384 1469 415">ADMS.setOfflineHitLimit(3000);</pre>

iOS Version 2.x to 3.x Migration Guide

- AppMeasurement is now ADMS_Measurement. Find locations where you reference this class and update the name to ADMS_Measurement.
- getInstance is now sharedInstance. Find locations where you call getInstance and replace it with sharedInstance.
- Remove all calls to churn measurement (getChurnInstance). These calls are handled by auto tracking and the variables are now inserted using context data.
- Timestamp is removed. The library handles timestamps automatically.

The syntax for the following methods have changed. Updated examples for all properties and methods are available in [Configuration Variables](#) and [Methods](#).

Previous Version (2.1.x)	New Version (3.x)
Report Suite: account	Report Suite: reportSuiteIDs
Track: <pre>(void)track:(NSDictionary *)variableOverrides;</pre>	Track: Pass nil for unused values. <pre>(void)trackWithContextData:(NSDictionary *)contextData variables:(NSDictionary *)variables;</pre>
Track Link: <pre>(void)trackLink:(NSString *)linkURL linkType:(NSString *)linkType linkName:(NSString *)linkName; (void)trackLink:(NSString *)linkURL linkType:(NSString *)linkType linkName:(NSString *)linkName variableOverrides:(NSDictionary *)variableOverrides;</pre>	Track Link: These two calls have been replaced with a single call. Pass nil for unused values. <pre>(void)trackLinkURL:(NSString *)linkURL withLinkType:(NSString *)linkType linkName:(NSString *)linkName contextData:(NSDictionary *)contextData variables:(NSDictionary *)variables;</pre>
Offline Tracking Methods: <pre>(void)forceOffline; (void)forceOnline;</pre>	Offline Tracking Methods: <pre>(void)setOnline; (void)setOffline;</pre>

Renamed Properties

The following header snippet shows version 2.x properties with their corresponding values in version 3.x. Several properties were removed from version 3.x to make the library more mobile focused and to simplify implementation.

```
timestamp;           //Removed
trackOffline;       //offlineTrackingEnabled
offlineLimit;       //offlineHitLimit
account;            //reportSuiteIDs
linkURL;            //Removed (sent with linkTrackURL)
linkName;           //Removed (sent with linkTrackURL)
linkType;           //Removed (sent with linkTrackURL)
linkTrackVars;     /**SAME**
linkTrackEvents;   /**SAME**
dc;                 //Removed
```

```
trackingServer;           /**SAME**
trackingServerSecure;    //Removed, trackingServer value is used for secure server if ssl=YES
userAgent;               //Removed
dynamicVariablePrefix;  //Removed
visitorID;               /**SAME**
charSet;                 /**SAME**
visitorNamespace;       //Removed
pageName;                //AppState
pageURL;                 //Removed
referrer;                //Removed
currencyCode;           /**SAME**
purchaseID;              /**SAME**
channel;                 /**SAME**
server;                  //appSection
pageType;                //Removed
transactionID;          /**SAME**
campaign;                /**SAME**
state;                   //geoState
zip;                     //geoZip
events;                  /**SAME**
products;                /**SAME**
list1-list3;            //Handled with [measurement setListVar:# toValue:@"value"];
hier1-heir5;            //Handled with [measurement setHier:# toValue:@"value"];
prop1-prop75;           //Handled with [measurement setProp:# toValue:@"value"];
eVar1-eVar75;           //Handled with [measurement setEvar:# toValue:@"value"];
ssl;                     /**SAME**
linkLeaveQueryString;    //Removed
debugTracking;          //debugLogging
usePlugins;              //Removed
useBestPractices;        //Removed - handled by Lifecycle AutoTracking
contextData;            //persistentContextData
```

iOS Device Versions

The following table contains the iOS version string sent by many iOS devices.

This list is based on internal tests and information found online, and might contain inaccuracies or incomplete information. Additional details on iOS devices are available online at sites such as [The iPhone Wiki](#).

iPhone	Version
Original	iPhone1,1
3G	iPhone1,2
3GS	iPhone2,1
4 (GSM) (shipped with firmware 4.0)	iPhone3,1
4 (GSM) (shipped with firmware 6.0)	iPhone3,2
4 (CDMA)	iPhone3,3
4S	iPhone4,1
5 (GSM)	iPhone5,1
5 (CDMA)	iPhone5,2
5s (GSM)	iPhone6,1
5s (Global)	iPhone6,2
5c (GSM)	iPhone5,3
5c (Global)	iPhone5,4
6	iPhone7,2
6 Plus	iPhone7,1

iPod Touch	Version
1st generation	iPod1,1
2nd generation	iPod2,1
3rd generation	iPod3,1
4th generation	iPod4,1
5th generation	iPod5,1

iPad	Version
Original	iPad1,1
iPad 2	iPad2,1
iPad 2 GSM	iPad2,2
iPad 2 CDMA	iPad2,3
iPad 2 (New 16gb)	iPad2,4
iPad mini, WiFi	iPad2,5

iPad mini, GSM	iPad2,6
iPad mini, Global version (the same as the GSM iPad mini, but includes an additional cellular radio: CDMA EV-DO Rev. A and Rev. B (3.5G))	iPad2,7
iPad 3 WiFi	iPad3,1
iPad 3 CDMA	iPad3,2
iPad 3 GSM	iPad3,3
iPad 4 WiFi	iPad3,4
iPad 4GSM	iPad3,5
iPad 4, Global version (the same as the GSM 4th gen iPad, but includes an additional cellular radio: CDMA EV-DO Rev. A and Rev. B (3.5G))	iPad3,6
iPad Air, WiFi	iPad4,1
iPad Air, Cellular	iPad4,2
iPad Air CDMA	iPad4,3
iPad mini retina (2nd generation), WiFi	iPad4,4
iPad mini retina (2nd generation), Cellular	iPad4,5
iPad Mini 2 (China)	iPad4,6
iPad Mini 3 (Wi-Fi)	iPad4,7
iPad Mini 3 (CDMA)	iPad4,8
iPad Mini 3 (GSM)	iPad4,9
iPad Air 2 (Wi-Fi)	iPad5,3
iPad Air 2 (LTE)	iPad5,4

Using Bloodhound to Test Mobile Applications

The Bloodhound Tool lets you send server calls to a local computer to test mobile applications.



Important: *As of April 30, 2017, Adobe Bloodhound has been sunset. Starting on May 1, 2017, no additional enhancements and no additional Engineering or Adobe Expert Care support will be provided.*

During application development, Bloodhound lets you view server calls locally, and optionally forward the data to Adobe collection servers.

Bloodhound is available for Mac and Windows.

Requirements

- An Intel-based Mac computer running Snow Leopard (10.6) or later, or a Windows PC.
- Network connectivity to your mobile devices or simulators.

Download

The Adobe Bloodhound download is available from the releases page of the [Adobe Experience Cloud/Mobile Services GitHub repository](#).

Installation

- **Mac:** Open the dmg you downloaded and drag Bloodhound to the Applications folder.
- **Windows:** Run the installation file you downloaded.

Using Bloodhound

After you start the tool, the server is disabled until you click the **Start** button. Click the **Start** button when you are ready to capture server calls from your application.

Optionally, you can specify a custom port number (must be above 1024) before you start the server. If you do not provide a port number, the server automatically selects an open port.

After the server is running, you need to configure your applications or devices to send data to the tool, as discussed in the next section.

Configure Devices to Send Hits to Bloodhound

You can change proxy settings on the device to send http requests to the tool. Requests that are sent to the tool can optionally be forwarded to Adobe Data Collection servers.

If your device does not support a proxy, you can send the hits directly to Bloodhound for testing.



Important: *Bloodhound does not support SSL tracking. You must disable SSL in the AppMeasurement library when testing using Bloodhound.*

iOS Devices

The iOS Device must be on the same network as the computer hosting Bloodhound.

1. Navigate to **Settings > Wi-Fi**.
2. Click the blue arrow to the right of your current Wi-Fi network.
3. Scroll to the **HTTP Proxy settings**.

4. Select **Auto**.
5. Enter the Proxy URL and port (from the Bloodhound UI) into the **URL** field.

Other Devices

If the device supports proxy auto config, simply use the Proxy URL (From the Bloodhound UI) as the Proxy Auto Config (PAC) URL. Proxy support varies across Android versions, and there are some proxy configuration tools available for Android to simplify configuration.

Send Hits Directly

For devices that do not support proxy (iOS Simulator, older Android versions, etc) it is possible to specify Bloodhound as your tracking server in order to capture the hits it generates. To do this set your tracking server to "<Bloodhound IP>:<Bloodhound Port>".

For example:

```
//iOS
[measure configureMeasurementWithReportSuiteIDs:@"my_rsid" trackingServer:@"10.10.2.2:5000"];
measure.ssl = NO;
```

```
//Android
measure.configureMeasurement("my_rsid", "10.10.2.2:5000");
measure.ssl = false;
```

```
//WinRT for Windows 8, Windows Phone 8
measure.ConfigureMeasurement("my_rsid", "10.10.2.2:5000");
measure.ssl = false;
```

Troubleshooting/Common Issues

- Bloodhound only functions with non-ssl tracking. Any tracking hits sent via SSL are not captured, regardless of the method used above.

Sample Code

```
+ (void)trackWithVariableOverrides {
    [TrackingHelper configureAppMeasurement];

    NSDictionary *overridesDictionary = [NSDictionary dictionaryWithObjectsAndKeys:
        @"event22,event23", @"events",
        @"propvalue", @"prop22",
        @"blah", @"pagename", nil];

    NSDictionary *contextData = nil;    // You can use contextData, if desired

    [[ADMS_Measurement sharedInstance] trackWithContextData: contextData variables:
overridesDictionary];

    // To send as a trackLink, use below instead of line above
    //[[ADMS_Measurement sharedInstance] trackLinkURL: @"URL" withLinkType: @"o" linkName:@"name"
contextData: nil variables:overridesDictionary];
}
```

Contact and Legal Information

Information to help you contact Adobe and to understand the legal issues concerning your use of this product and documentation.

Help & Technical Support

The Adobe Experience Cloud Customer Care team is here to assist you and provides a number of mechanisms by which they can be engaged:

- [Check the Experience Cloud help pages for advice, tips, and FAQs](#)
- [Ask us a quick question on Twitter @AdobeExpCare](#)
- [Log an incident in our customer portal](#)
- [Contact the Customer Care team directly](#)
- [Check availability and status of Experience Cloud Solutions](#)

Service, Capability & Billing

Dependent on your solution configuration, some options described in this documentation might not be available to you. As each account is unique, please refer to your contract for pricing, due dates, terms, and conditions. If you would like to add to or otherwise change your service level, or if you have questions regarding your current service, please contact your Account Manager.

Feedback

We welcome any suggestions or feedback regarding this solution. Enhancement ideas and suggestions [can be added to our Customer Idea Exchange](#).

Legal

© 2017 Adobe Systems Incorporated. All Rights Reserved.
Published by Adobe Systems Incorporated.

[Terms of Use](#) | [Privacy Center](#)

Adobe and the Adobe logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries. A trademark symbol (®, ™, etc.) denotes an Adobe trademark.

All third-party trademarks are the property of their respective owners. Updated Information/Additional Third Party Code Information available at <http://www.adobe.com/go/thirdparty>.