



Adobe® Experience Cloud
AppMeasurement 3.x for Android

Contents

AppMeasurement 3.x for Android.....	4
Download the Library.....	5
Analyst Quick Start.....	6
Developer Quick Start.....	7
Video Measurement Quick Start.....	10
Lifecycle Metrics.....	13
Context Data.....	17
Configuration Variables.....	18
Tracking Variables.....	21
Methods.....	24
Offline Tracking.....	30
Target.....	31
Audience Management.....	34
PhoneGap Plug-in.....	37
PhoneGap Plug-in Methods.....	37
Android Version 2.x to 3.x Migration Guide.....	50

Using Bloodhound to Test Mobile Applications.....52

Android Widgets.....54

AppMeasurement 3.x for Android

Adobe® AppMeasurement for Android lets you measure native Android applications in the Adobe Experience Cloud.

New! 4.x SDKs for Experience Cloud Solutions are now available. See [Measuring and Optimizing Mobile Applications](#).

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

Developer Quick Start	This section walks you through selecting and configuring the events, eVars, and props that you'll use to collect Android data. Steps are also included to create Processing Rules to copy the context data sent by the Android libraries to these variables.
Analyst Quick Start	This section walks you through implementing the Android 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

Android 2.0 or later.

Release History

[Android](#)

Documentation Revision History

Revision Date	Description
August 06, 2013	Added support for Audience Management .
November 08, 2012	Added the <code>lifecycleSessionTimeout</code> variable that is new in version 3.1.1.
October 19, 2012	Removed the requirement for <code>READ_PHONE_STATE</code> app permission from Developer Quick Start .

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 SiteCatalyst 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 SiteCatalyst

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

1. Open **Admin Console > Report Suites > Edit Settings > Mobile Management > Mobile Application Reporting**.
2. Click **Enable Mobile App Livecycle Tracking**.

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

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 ClientCare.

Developer Quick Start

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

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

Get the Library

Visit [Measuring and Optimizing Mobile Applications](#) on Developer Connection to download the Android library.

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

- `admsAppLibrary.jar`: Library designed for use with Android devices and simulators. Requires Android 2.0 or later.
- `Examples\TrackingHelper.java`: Optional class that is designed to keep tracking code separate from your application logic.

Add the Library to your Project

1. In the Eclipse IDE, right-click on the project name.
2. Select **Build Path > Add External Archives**.
3. Select `admsAppLibrary.jar`.
4. Click **Open**.
5. Right-click the project again, then select **Build Path > Configure Build Path**.
6. Click the **Order and Export** tab.
7. Ensure that `admsAppLibrary.jar` is selected.

Your application can import the classes/interfaces from the `admsAppLibrary.jar` library by using `import com.adobe.ADMS.*;`

Add App Permissions

The AppMeasurement Library requires the following permissions to send data and record offline tracking calls:

- `INTERNET`
- `ACCESS_NETWORK_STATE`

To add these permissions, add the following lines to your `AndroidManifest.xml` file (in the application project directory):

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

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):

```
Hashtable<String, Object> contextData = new Hashtable<String, Object>();
contextData.put("username", "username_value");
measure.trackEvents("event1", 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`:

```
public static void trackLogonEvent (String username_value) {
    Hashtable<String, Object> contextData = new Hashtable<String, Object>();
    contextData.put("username", username_value);
    measure.trackEvents("event1", contextData);
}
```

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

```
TrackingHelper.trackLogonEvent("username");
```

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, even though setup requires an extra minute or two. The remaining steps in this guide walk you through the steps to set up `TrackingHelper` and start sending measurement data.

Be sure to copy `TrackingHelper.java` to a directory that contains class files for your app, and replace the package name at the top of this file to match your package structure (`com.company.application`). If you'd rather implement without `TrackingHelper`, you can find several samples inside of `TrackingHelper` that you can copy to your application.

Implementation

1. Open `TrackingHelper.java` and update `TRACKING_RSID` and `TRACKING_SERVER` with your report suite ID and tracking server. For example:

```
private static final String TRACKING_RSID = "rsid";
private static final String TRACKING_SERVER = "server";
```

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

2. Find the `configureAppMeasurement` method. This is where you enable SSL, enable offline tracking, and make other global changes to your configuration. For now, uncomment the line to enable `debugLogging` until things are working the way you'd expect.
3. Add a call to `configureAppMeasurement` from the root activity in your application.

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    TrackingHelper.configureAppMeasurement(this);
}
```

That is all the code you need to start tracking custom metrics. However, with a few additional lines of code, you can enable lifecycle tracking to automatically send lifecycle metrics, including launches, upgrades, crashes, and daily and monthly users. The `AppMeasurement` library does all of the heavy lifting, all you need to do is add two method calls to each of your application's Activity classes.

(Recommended) Track Lifecycle Events

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

To start tracking lifecycle events, in your app, add calls to the `onResume()` and `onPause()` methods in each Activity inside of your application, as shown in the following example. We also recommend passing the Activity or Service as the context object instead of the global Application context.

```
@Override
protected void onResume() {
    super.onResume();
    TrackingHelper.startActivity(this);
}
@Override
protected void onPause() {
    super.onPause();
    TrackingHelper.stopActivity();
}
```

That's it! You've now implemented basic lifecycle tracking in your Android app. After your Web Analyst configures SiteCatalyst to process the AppMeasurement metrics you are reporting, your SiteCatalyst report suite will contain the default lifecycle metrics.

Where to go from here:

- [\(Optional\) 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.
- [\(Optional\) Track App 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.

(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.java`, define a custom event track method:

```
public static void trackCustomEvents () {
    Hashtable<String, Object> contextData = new Hashtable<String, Object>();
    contextData.put("contextKey", "value");
    measure.trackEvents("event1, event2", contextData);
}
```

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) Track App 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", 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 SiteCatalyst](#) 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 same library, `admsAppLibrary.jar`, 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 Android, complete the following tasks:

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

Map Player Events to SiteCatalyst Variables

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

Your Web Analyst should provide you with a [Video Implementation Worksheet](#) that contains a list of the SiteCatalyst 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

1. Add the following code after the code you added in [Implementation](#), replacing the SiteCatalyst variable you selected with the example in the code:

```
ADMS_MediaMeasurement mediaMeasure = ADMS_MediaMeasurement.sharedInstance();

Hashtable<String, Object> contextDataMapping = new Hashtable<String, Object>();
contextDataMapping.put("a.media.name", "eVar2,prop2");
contextDataMapping.put("a.media.segment", "eVar3");
contextDataMapping.put("a.contentType", "eVar1"); //note that this is not in the .media
namespace
contextDataMapping.put("a.media.timePlayed", "event3");
contextDataMapping.put("a.media.view", "event1");
contextDataMapping.put("a.media.segmentView", "event2");
contextDataMapping.put("a.media.complete", "event7");

mediaMeasure.ContextDataMapping = contextDataMapping;
```

2. [Configure Milestones, Segments, and Call Frequency](#).
3. [Track Player Events](#).

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.

```
Hashtable<String, Object> milestoneMapping = new Hashtable<String, Object>();
    milestoneMapping.put("25", "event4");
    milestoneMapping.put("50", "event5");
    milestoneMapping.put("75", "event6");

contextDataMapping.put("a.media.milestones", milestoneMapping);
```

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.

```
Hashtable<String, Object> offsetMilestoneMapping = new Hashtable<String, Object>();
    offsetMilestoneMapping.put("20", "event8");
    offsetMilestoneMapping.put("40", "event9");
    offsetMilestoneMapping.put("60", "event10");

contextDataMapping.put("a.media.offsetMilestones", offsetMilestoneMapping);
```

Examples

```
//get sharedInstance
ADMS_MediaMeasurement mediaMeasure = ADMS_MediaMeasurement.sharedInstance();

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

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

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

// track Milestones & segmentByMilestones & seconds:
mediaMeasure.trackMilestones = "25,50,75";
mediaMeasure.segmentByMilestones = true;
mediaMeasure.trackSeconds = 15;

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

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

Track Player Events

To measure video, you need to add code that tells the media module when events occur in your player (using the `open`, `play`, `stop`, and `close` methods). When a user starts playing a video, you need to call the `play` method so the media module starts counting seconds viewed. If the user pauses the video, you need to call `stop` so the count is paused. This is typically performed using event handlers that are exposed by your player.

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:

```
function startMovie(){
  mediaMeasure.open(mediaName,mediaLength,mediaPlayerName);
  playMovie();
}

/*Call on video resume from pause and slider release*/
function playMovie(){
  mediaMeasure.play(mediaName,mediaOffset);
}

/*Call on video pause and slider grab*/
function stopMovie(){
  mediaMeasure.stop(mediaName,mediaOffset);
}

/*Call on video end*/
function endMovie(){
  stopMovie();
  mediaMeasure.close(mediaName);
}
```



Lifecycle Metrics

This worksheet lists the metrics and dimensions that are measured when automatic lifecycle tracking is enabled.

Lifecycle Metrics and Dimensions

When configured, lifecycle metrics are sent in context data parameters to Analytics, parameters to Target with each mbox call, and as a signal to audience management. Analytics and Target use the same format, while audience management uses a different prefix for each metric.



For Analytics, the context data that is sent with each lifecycle tracking call is automatically captured in and reported using the metric or dimension listed in the first column, with the exceptions noted in the description column.


Metric	Analytics Context Data/Target Parameter	Audience Manager Signal	Description
First Launces	a.InstallEvent	c_a_InstallEvent	Triggered on first run after installation (or re-installation).
Upgrades	a.UpgradeEvent	c_a_UpgradeEvent	Triggered on first run after upgrade (anytime the version number changes).
Daily Engaged Users	a.DailyEngUserEvent	c_a_DailyEngUserEvent	Triggered when the application is used on a particular day.  Note: <i>Daily Engaged Users is not automatically stored in an Analytics metric. You must create a processing rule that sets a custom event to capture this metric.</i>
Monthly Engaged Users	Monthly Engaged Users	a.MonthlyEngUserEvent	Triggered when the application is used during a particular month.  Note: <i>Monthly Engaged Users is not automatically stored in an Analytics metric. You must create a processing rule that sets a custom event to capture this metric.</i>

Metric	Analytics Context Data/Target Parameter	Audience Manager Signal	Description
			Note:
Launches	a.LaunchEvent	c_a_LaunchEvent	Triggered on every run, including crashes and installs. Also triggered on a resume from background when the lifecycle session timeout has been exceeded.
Crashes	a.CrashEvent	c_a_CrashEvent	Triggered when the application does not exit gracefully. Event is sent on application start after crash (the application is considered to crash if quit is not called).
Previous Session Length	a.PreviousSessionLength	Cc_a_PreviousSessionLength	Aggregated total Previous Session Length in seconds.

Dimensions

Metric	Analytics Context Data/Target Parameter	Audience Manager Signal	Description
Install Date	a.InstallDate	c_a_InstallDate	Date of first launch after installation. MM/DD/YYYY
App ID	a.AppID	c_a_AppID	Stores the Application name and version in the following format: [AppName] [BundleVersion] For example, myapp 1.1
Days since first use	a.DaysSinceFirstUse	c_a_DaysSinceFirstUse	Number of days since first run.
Days since last use	a.DaysSinceLastUse	c_a_DaysSinceLastUse	Number of days since last use.
Day of Week	a.DayOfWeek	c_a_DayOfWeek	Number of the week day the app was launched.
Hour of Day	a.HourOfDay	c_a_HourOfDay	Measures the hour the app was launched. 24 hour numerical

Metric	Analytics Context Data/Target Parameter	Audience Manager Signal	Description
			format. Used for time parting to determine peak usage times.
Day of Week	a.DayOfWeek	c_a_DayOfWeek	Number of the week day the app was launched.
Operating System Version	a.OSVersion	c_a_OSVersion	OS version.
Days since last upgrade	a.DaysSinceLastUpgrade	c_a_DaysSinceLastUpgrade	<p>Number of days since the application version number has changed.</p> <p> Note: Days since last upgrade is not automatically stored in an Analytics variable. You must create a processing rule to copy this value to an Analytics variable for reporting.</p>
Launches since last upgrade	a.LaunchesSinceUpgrade	c_a_LaunchesSinceUpgrade	<p>Number of launches since the application version number has changed.</p> <p> Note: Launches since last upgrade is not automatically stored in an Analytics variable. You must create a processing rule to copy this value to an Analytics variable for reporting.</p>
Device Name	a.DeviceName	c_a_DeviceName	<p>Stores the device name.</p> <p>iOS: Comma-separated 2 digit string that Identifies the iOS device. The first number typically represents the device generation, and the second</p>

Metric	Analytics Context Data/Target Parameter	Audience Manager Signal	Description
			number typically versions different members of the device family. See iOS Device Versions for a list of common device names.
Carrier Name	a.CarrierName	c_a_CarrierName	<p>Stores the name of the mobile service provider as provided by the device.</p> <p> Note: Carrier name is not automatically stored in an Analytics variable. You must create a processing rule to copy this value to an Analytics variable for reporting.</p>
Resolution	a.Resolution	c_a_Resolution	Width x Height in actual pixels

Context Data

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

Instead of explicitly assigning values to props and eVars, you can use context data variables to send name value pairs that are mapped in the SiteCatalyst. Based on these key value pairs, you can set events, copy values to eVars and props, and execute additional conditional statements. This 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. Any context data set directly on the `PersistentContextData` object is sent in with each call. You can also pass context data as a parameter to a single tracking call that is 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:

```
Hashtable<String, Object> contextData = new Hashtable<String, Object>();
contextData.put("key", "value");
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:

```
Hashtable<String, Object> contextData = new Hashtable<String, Object>();
contextData.put("key", "value");
measure.setPersistentContextData = contextData;

Hashtable<String, Object> contextDataState = new Hashtable<String, Object>();
contextDataState.put("key2", "value2");

measure.trackAppState("state1");
measure.trackAppState("state2", contextDataState);
measure.trackAppState("state3");
```

Configuration Variables

The following variables are set when the application launches

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

Shared Instance

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

```
ADMS_Measurement measure = ADMS_Measurement.sharedInstance(((Activity)
context).getApplication());
```



Note: Configuration variables are private. Use the get and set methods to change these values.

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>Syntax:</p> <pre>String getReportSuiteIDs() void setReportSuiteIDs(String reportSuiteIDs)</pre> <p>Examples:</p> <pre>measure.setReportSuiteIDs("rsid");</pre> <p>Multi-suite tagging:</p> <pre>measure.setReportSuiteIDs("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 <code>true</code>, a secure connection is made to this server. If <code>ssl</code> is <code>false</code>, 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>Syntax:</p> <pre>String getTrackingServer() void setTrackingServer(String trackingServer)</pre> <p>Examples:</p> <pre>measure.setTrackingServer("metrics.corpl.com");</pre>
visitorID	<p>Default: uuid</p> <p>Unique identifier for each visitor. If you do not set a custom visitorID, a unique visitorID is generated.</p>

Configuration Variables	Description
	<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. To avoid this, set the visitor ID before you configure app measurement.</p>
characterSet	<p>Default: UTF-8</p> <p>Syntax:</p> <pre>String getCharSet() void setCharSet(String charSet)</pre> <p>Examples:</p> <pre>[measure.setCharacterSet("UTF-8");</pre>
currencyCode	<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 Android application.</p> <p>Syntax:</p> <pre>String getCurrencyCode() void setCurrencyCode(String currencyCode)</pre> <p>Examples:</p> <pre>measure.setCurrencyCode("USD");</pre>
lifecycleSessionTimeout	<p>Default: 5 minutes</p> <p>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.</p> <p>Syntax:</p> <pre>int getLifecycleSessionTimeout() void setLifecycleSessionTimeout(int sessionLength)</pre> <p>Examples:</p> <pre>measure.setLifecycleSessionTimeout(600); //10 minute session</pre>
ssl	<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 setSSL(boolean ssl)</pre> <p>Examples:</p> <pre>measure.setSSL(true);</pre>

Configuration Variables	Description
debugLogging	<p data-bbox="475 247 618 279">Default: false</p> <p data-bbox="475 302 1474 369">Enables (true) or disables (false) viewing debug information and the library version in the output console. By default, this variable is false.</p> <p data-bbox="475 392 1406 424">You cannot override this variable for a single call, you must change the persistent value.</p> <p data-bbox="475 447 558 478">Syntax:</p> <pre data-bbox="475 489 1073 520">void setDebugLogging(boolean debugLogging)</pre> <p data-bbox="475 537 589 569">Examples:</p> <pre data-bbox="475 579 902 611">measure.setDebugLogging(true);</pre>

Tracking Variables

Shared Instance

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

```
ADMS_Measurement measure = ADMS_Measurement.sharedInstance(((Activity)
context).getApplication());
```



Note: Configuration variables are private. Use the get and set methods to change these values.

Persistent Tracking Variables	Description
Evar	<p>Sets the specified eVar to the provided value. The eVar is sent with all tracking calls until it is cleared by setting it to an empty string, or by calling <code>clearVars</code>.</p> <p>Syntax:</p> <pre>void setEvar(int evarNum, String evarValue)</pre> <p>Example:</p> <pre>measure.setEvar(1, "value");</pre>
Prop	<p>Sets the specified prop to the provided value. The prop is sent with all tracking calls until it is cleared by setting it to an empty string, or by calling <code>clearVars</code>.</p> <p>Syntax:</p> <pre>void setProp(int propNum, String propValue)</pre> <p>Example:</p> <pre>measure.setProp(1, "value");</pre>
Hier	<p>Sets the specified heir variable to the provided value. The heir variable is sent with all tracking calls until it is cleared by setting it to an empty string, or by calling <code>clearVars</code>.</p> <p>Syntax:</p> <pre>void setHier(int hierNum, String hierValue)</pre> <p>Example:</p> <pre>measure.setHier(1, "value");</pre>
ListVar	<p>Sets the specified listVar to the provided value. The listVar is sent with all tracking calls until it is cleared by setting it to an empty string, or by calling <code>clearVars</code>.</p> <p>Syntax:</p> <pre>void setListVar(int listNum, String listValue)</pre> <p>Example:</p> <pre>measure.setListVar(1, "value");</pre>

Persistent Tracking Variables	Description
purchaseID	<p>The purchaseID is used to keep an order from being counted multiple times by SiteCatalyst.</p> <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.setPurchaseID("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.setTransactionID("unique_id");</pre>
appState (page name)	<p>The value provided for the app state is stored in the page name variable.</p> <pre>measure.setAppState("state");</pre>
channel	<pre>measure.setChannel("mobile");</pre>
appSection	<p>The value provided for the app section is stored in the server variable.</p> <p>Syntax:</p> <pre>void setAppSection(String Section)</pre> <p>Example:</p> <pre>measure.setAppSection("news");</pre>
campaign	<p>Syntax:</p> <pre>void setCampaign(String Campaign)</pre> <p>Example:</p> <pre>measure.setCampaign("112233");</pre>
products	<p>Syntax:</p> <pre>void setProducts(String Products) // example Products string: Category;Product[,Category;Product]</pre> <p>Example:</p> <pre>measure.setProducts("Running;Shoe");</pre>
events	<p>Syntax:</p> <pre>void setEvents(String Event) //comma-delimited list</pre> <p>Example:</p> <pre>measure.setEvents("event1, event2");</pre>

Persistent Tracking Variables	Description
geoState	<p>Syntax:</p> <pre>void setGeoState(String GeoState)</pre> <p>Example:</p> <pre>measure.setGeoState("UT");</pre>
geoZip	<p>Syntax:</p> <pre>void setGeoZip(String GeoZip)</pre> <p>Example:</p> <pre>measure.setGeoZip("12345");</pre>
Persistent Context Data	<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 (<code>trackAppState</code>, <code>trackEvents</code>, and so on).</p> <p>Example:</p> <pre>Hashtable contextData = new Hashtable<String, Object>(); contextData.put("key", "value"); measure.setPersistentContextData(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. If <code>linkTrackVars</code> is not defined, AppMeasurement for Android sends all defined variables with a <code>trackLink</code> call.</p> <p>Syntax:</p> <pre>void setLinkTrackVars(String Vars) //comma-delimited list</pre> <p>Example:</p> <pre>measure.setLinkTrackVars("eVar1, prop1");</pre>
linkTrackEvents	<p>A comma delimited list of events that restricts the current set of events for link tracking. If <code>linkTrackEvents</code> is not defined, AppMeasurement for Android sends all events with a <code>trackLink</code> call.</p> <p>Syntax:</p> <pre>void setLinkTrackEvents(String Events) //comma-delimited list</pre> <p>Example:</p> <pre>measure.setLinkTrackEvents("event1, event2");</pre>

Methods

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

Shared Instance

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

```
ADMS_Measurement measure = ADMS_Measurement.sharedInstance(((Activity)
context).getApplication());
```

Initial Configuration

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

Configuration Methods	Description
configureMeasurement	<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 configureMeasurement(String reportSuiteIDs, String trackingServer)</pre> <p>Examples:</p> <pre>measure.configureMeasurement("my_rsid", "my_tracking_server");</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 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 trackAppState(string AppStateName)</pre> <p>Examples:</p> <pre>measure.trackAppState("state"); Hashtable contextData = new Hashtable<String, Object>(); contextData.put("key", "value"); measure.trackAppState("state", contextData);</pre>

Method	Description
trackEvents	<p>This is the recommended way to track events in your application. trackEvents is similar to trackAppState, but sends events instead of page names. Events are provided as a comma delimited string. The events 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 trackLinkURL where linkURL is set to nil, linkType is set to "o", and the linkName is populated with the application ID.</p> <p>This means that your linkTrackVars and linkTrackEvents apply for calls to trackEvents.</p> <p>Context data sent with this call is appended to any values in persistentContextData and sent with the call. Only values set in persistentContextData remain for the next call.</p> <p>Syntax:</p> <pre>void trackEvents(string Events)</pre> <p>Examples:</p> <pre>measure.trackEvents("event1");</pre> <pre>Hashtable<String, Object> contextData = new Hashtable<String, Object>(); contextData.put("key", "value"); measure.trackEvents("event1", contextData);</pre> <p>Import note if you are using the trackLink method</p> <p>trackEvents makes an underlying call to trackLink, where linkURL is set to null, linkType is set to "o", and the linkName is populated with the application ID. This is done to prevent the page view (state view) count from being incremented each time you send an event.</p> <p>If you are calling trackLink directly and setting values for linkTrackVars and linkTrackEvents, these variable and event restrictions apply to TrackEvents. This means that only variables and events defined on these lists are sent with TrackEvents. You should set linkTrackVars and linkTrackEvents to null unless you want those restrictions applied to trackEvents.</p>

Advanced Tracking (track and trackLink)

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



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 setPersistentContextData). 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
track	<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 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>void track() void track(Hashtable<String, Object> contextData) void track(Hashtable<String, Object> contextData, Hashtable<String, Object> variables)</pre> <p>Examples:</p> <pre>measure.track(); measure.setEvar(1, "evar1value"); Hashtable contextData = new Hashtable<String, Object>(); contextData.put("key", "value"); measure.track(contextData); //set an eVar measure.setEvar(1, "evar1value"); //contextData Hashtable contextData = new Hashtable<String, Object>(); contextData.put("key", "value"); //variable overrides Hashtable variableOverrides = new Hashtable<String, Object>(); variableOverrides.put("evar2", "evar2value"); //sends eVar1, eVar2 (set in overrides), and context data measure.track(contextData, variableOverrides);</pre>
trackLink	<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>void trackLink(String linkURL, String linkType, String linkName, Hashtable<String, Object> contextData, Hashtable<String, Object> 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 Android application. Otherwise, pass in null for this parameter.</p> <p>linkType: Identifies the link report that will display the URL or name. Supported values include:</p> <ul style="list-style-type: none"> • “o” (Custom Links) • “d” (File Downloads) • “e” (Exit Links)

Method	Description
	<p>linkName: The name that appears in the link report. If no name is specified, the report uses the URL.</p> <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>null</code> as the value.</p> <p>Examples:</p> <pre>measure.trackLink("url", "o", "name", contextData, null);</pre>
<code>setEvar</code>	<p>Sets the specified eVar to the provided value. The eVar is sent with all tracking calls until it is cleared by setting it to an empty string, or by calling <code>clear vars</code>.</p> <p>Syntax:</p> <pre>void setEvar(int evarNum, String evarValue)</pre>
<code>setProp</code>	<p>Sets the specified prop to the provided value. The prop is sent with all tracking calls until it is cleared by setting it to an empty string, or by calling <code>clear vars</code>.</p> <p>Syntax:</p> <pre>void setProp(int propNum, String propValue)</pre>
<code>setHier</code>	<p>Sets the specified heir variable to the provided value. The heir variable is sent with all tracking calls until it is cleared by setting it to an empty string, or by calling <code>clear vars</code>.</p> <p>Syntax:</p> <pre>void setHier(int hierNum, String hierValue)</pre>
<code>setListVar</code>	<p>Sets the specified listVar to the provided value. The listVar is sent with all tracking calls until it is cleared by setting it to an empty string, or by calling <code>clear vars</code>.</p> <p>Syntax:</p> <pre>void setListVar(int listNum, String listValue)</pre>
<code>setPersistentContextData</code>	<p>Values placed in Persistent Context Data are sent with each server call. To send context data for a single call, send a <code>contextData</code> hashtable as a parameter to the tracking call (<code>trackAppState</code>, <code>trackEvents</code>, and so on).</p> <pre>Hashtable<String, Object> contextData = new Hashtable<String, Object>(); contextData.put("key", "value"); measure.setPersistentContextData(contextData);</pre> <p>See Context Data.</p>
<code>clearVars</code>	<p>Removes values from the following variables on the object:</p> <pre>props eVars heirs listVars events PersistentContextData</pre>

Method	Description
	<p>purchaseID transactionID appState channel appSection campaign products geoZip geoState linkTrackVars linkTrackEvents</p> <p>Syntax:</p> <pre>void clearVars()</pre> <p>Examples:</p> <pre>measure.clearVars();</pre>

Offline Tracking



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. To enable offline AppMeasurement, your report suite must be timestamp-enabled. After you make this change, 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).

Configuration Method	Description
<code>setOfflineTrackingEnabled</code>	<p>Default: false</p> <p>Enables or disables offline tracking for the measurement library.</p> <p>Syntax:</p> <pre>void setOfflineTrackingEnabled(boolean Enabled);</pre> <p>Examples:</p> <pre>measure.setOfflineTrackingEnabled(true);</pre>
<code>setOfflineHitLimit</code>	<p>Default: 1000</p>

Configuration Method	Description
	<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>void setOfflineHitLimit(int offlineHitLimit)</pre> <p>Examples:</p> <pre>measure.setOfflineHitLimit(2000);</pre>
getTrackingQueueSize	<p>Returns the number of stored tracking calls in the offline queue.</p> <p>Syntax:</p> <pre>int getTrackingQueueSize()</pre> <p>Examples:</p> <pre>int size = measure.getTrackingQueueSize();</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. 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 offlineTrackingEnabled is true: hits are stored until measurement is online. • If offlineTrackingEnabled is false: hits are discarded. <p>Syntax:</p> <pre>void setOnline() void setOffline()</pre> <p>Examples:</p> <pre>measure.setOffline(); measure.setOnline();</pre>

Offline Tracking

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



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

See [Offline Tracking](#).

Target

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

The content returned from Test&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 Android application developer to decide how the content is consumed within the application. The content can be displayed as HTML, or used to change the behavior of the application. This guide provides a simple use example of the Test&Target classes.

Requirements

- JDK 5/6/7
- Android SDK for Platform 1.5 or later.

The example in this section is based on Eclipse.

Get the Library

Visit [Measuring and Optimizing Mobile Applications](#) on Developer Connection to download the Android library.

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

- `admsAppLibrary.jar`: Library designed for use with Android devices and simulators. Requires Android 2.0 or later.
- `Examples\TrackingHelper.java`: Optional class that is designed to keep tracking code separate from your application logic.

Add the Library to your Project

1. In the Eclipse IDE, right-click on the project name.
2. Select **Build Path > Add External Archives**.
3. Select `admsAppLibrary.jar`.
4. Click **Open**.
5. Right-click the project again, then select **Build Path > Configure Build Path**.
6. Click the **Order and Export** tab.
7. Ensure that `admsAppLibrary.jar` is selected.

Your application can import the classes/interfaces from the `admsAppLibrary.jar` library by using `import com.adobe.ADMS.*;`

Add App Permissions

The AppMeasurement Library requires the following permissions to send data and record offline tracking calls:

- `INTERNET`
- `ACCESS_NETWORK_STATE`

To add these permissions, add the following lines to your `AndroidManifest.xml` file (in the application project directory):

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Example

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

The following example shows how to load HTML content from Test&Target into a `WebView` within a simple Android application. This example assumes that the associated HTML Offers and campaign have already been created in the Test&Target Admin Tool, and that the campaign has been approved.

1. Complete the steps in [Implementation](#), then import the `testandtarget` package at the top of your Application or Activity subclass:

```
com.adobe.adms.testandtarget.*;
```

2. Follow the numbered code below for creating an mbox (see comments for 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.

```
public class AndroidDemoApplication extends Activity {
    private WebView _webView;

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        _webView = new WebView(this);
        setContentView(_webView);

        // 1. Create an instance of the MboxFactory class with your client code.
        MboxFactory factory = new MboxFactory("androiddemo16");

        // 2. Use the MboxFactory instance to create an Mbox.
        Mbox mbox = factory.create("DemoApp");

        // 3. Set the default content for the Mbox.
        mbox.setDefaultContent("<h1>DEFAULT CONTENT</h1>");

        /**
         * 4. Create a custom MboxContentConsumer to consume the content returned. The
         *    CustomMboxContentConsumer class defined below simply displays the content
         *    returned in a BrowserField as HTML.
         */
        CustomConsumer consumer = new CustomConsumer(_webView);
        mbox.addOnLoadConsumer(consumer);

        // 5. Load the Mbox.
        mbox.load();
        ...
    }
}
```

3. Define the custom class that implements the `MboxContentConsumer` interface. This abstract interface only requires that you define a method named “consume” to pass content into. The `CustomConsumer` class defined below simply displays the content in a `WebView`.

```
class CustomConsumer implements MboxContentConsumer {
    private WebView _view;
    public CustomConsumer(WebView webview) {
        _view = webview;
    }
    public void consume(String content) {
        _view.loadData(content, "text/html", "utf-8");
    }
}
```

4. Right-click your project, and select **Run As > Android Application** to build and test your application. If you have correctly setup and approved your Test&Target campaign, you should see your offer displayed in the Android device emulator.

Lifecycle Metrics

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

- *(Recommended) Track Lifecycle Events*
- *Lifecycle Metrics*

Audience Management

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

Requirements

- JDK 5/6/7
- Android SDK for Platform 1.5 or later.

The example in this section is based on Eclipse.

Get the Library

Visit [Measuring and Optimizing Mobile Applications](#) on Developer Connection to download the Android library.

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

- `admsAppLibrary.jar`: Library designed for use with Android devices and simulators. Requires Android 2.0 or later.
- `Examples\TrackingHelper.java`: Optional class that is designed to keep tracking code separate from your application logic.

Add the Library to your Project

1. In the Eclipse IDE, right-click on the project name.
2. Select **Build Path > Add External Archives**.
3. Select `admsAppLibrary.jar`.
4. Click **Open**.
5. Right-click the project again, then select **Build Path > Configure Build Path**.
6. Click the **Order and Export** tab.
7. Ensure that `admsAppLibrary.jar` is selected.

Your application can import the classes/interfaces from the `admsAppLibrary.jar` library by using `import com.adobe.ADMS.*;`

Add App Permissions

The AppMeasurement Library requires the following permissions to send data and record offline tracking calls:

- `INTERNET`
- `ACCESS_NETWORK_STATE`

To add these permissions, add the following lines to your `AndroidManifest.xml` file (in the application project directory):

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Code Sample

After you add the library to your project, you can use the `ADMS_AudienceManager` class. To import the audience manager package add: `import com.adobe.adms.audiencemanager;`

1. Configure audience management:

```
ADMS_AudienceManager.ConfigureAudienceManager("mycompany.demdex.net", this);
```

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.SetDpidAndDpuuid("284", "abc123");
```

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

```
HashMap<String, Object> data = new HashMap<String, Object>();
data.put("trait", "b");
ADMS_AudienceManager.SubmitSignal(data, new
ADMS_AudienceManager.AudienceManagerCallback<HashMap>() {
    @Override
    public void call(HashMap visitorProfile) {
        // do things with visitorProfile
    }
});
```

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>public static void ConfigureAudienceManager(String server, Context context);</pre> <p>Example:</p> <pre>ADMS_AudienceManager.ConfigureAudienceManager("mycompany.demdex.net", this);</pre>
GetDebugLogging	<p>Returns boolean indicating whether or not Audience Manager methods will provide debug logging in your console.</p> <p>Syntax:</p> <pre>public static boolean GetDebugLogging();</pre>
GetDpid	<p>Returns the <code>dpid</code>.</p> <p>Syntax:</p> <pre>public static String GetDpid();</pre>
GetDpuuid	<p>Returns the <code>dpuuid</code>.</p>

Method	Description
	<p>Syntax:</p> <pre>public static String GetDpuuid();</pre>
GetVisitorProfile	<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>public static HashMap GetVisitorProfile();</pre>
SetDebugLogging	<p>Enables or disables debug logging in your console.</p> <p>Syntax:</p> <pre>public static void SetDebugLogging(boolean logging);</pre>
SetDpidAndDpuuid	<p>If <code>newDpid</code> and <code>newDpuuid</code> are set, they will be sent with each signal.</p> <p>Syntax:</p> <pre>public static void SetDpidAndDpuuid(String newDpid, String newDpuuid);</pre>
SubmitSignal	<p>Sends in a dictionary of traits and receives the updated visitor profile in <code>callback</code>.</p> <p>The <code>uuid</code> 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>public static void SubmitSignal(HashMap<String, Object> data, AudienceManagerCallback<HashMap> callback);</pre>

Interfaces

Method	Description
AudienceManagerCallback	<p>Syntax:</p> <pre>AudienceManagerCallback public interface AudienceManagerCallback<T> { public void call(T item); }</pre> <p>Interface for object used in callback from <code>SubmitSignal</code> call.</p>

PhoneGap Plug-in

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

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

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

Include the Plug-in

1. Copy `ADMS_plugin.java` to the package in your `src` folder.
2. copy `ADMS_Helper.js` to the `assets/www/js` folder in your PhoneGap project.
3. In the `res/xml` folder, open `config.xml` file and register a new plugin by creating a new child node under `plugins`:

```
<plugin name="ADMS_Plugin" value="[YOUR_PACKAGE_NAME].ADMS_plugin" />
```

For example, if you package is named `com.example.phonegaptest`, then your plugin node would be the following:

```
<plugin name="ADMS_Plugin" value="com.example.phonegaptest.ADMS_plugin" />
```

Include the AppMeasurement Library

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

1. Copy `admsAppLibrary.jar` to the `libs` folder in your PhoneGap project.
2. Verify `admsAppLibrary.jar` is in your build path by right-clicking on **libs** > **Build Path** > **Configure Build Path**.
3. in the **Libraries** tab, if it's not already in your list, click **Add JARs** and select `admsAppLibrary.jar` from your `libs` folder

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 the following error: `java.lang.ClassNotFoundException: ADMS_plugin`

- if this error occurs, make sure you did step 3 in the [Include the Plug-in](#) section

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. <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> <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</p>

Method	Description
	<p>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 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>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 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>

Method	Description
	<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>
<code>trackEventsWithContextData</code>	<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 SiteCatalyst variables directly. These should be used by customers with an existing implementation or customers who are very familiar with other SiteCatalyst 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 ClientCare.</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>

Android 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: <code>account</code>	Report Suite: <code>reportSuiteIDs</code>
Track: <code>String track()</code> <code>String track(Hashtable variableOverrides)</code>	Track: Pass null for unused values. <code>void track()</code> <code>void track(Hashtable<String, Object> contextData)</code> <code>void track(Hashtable<String, Object> contextData, Hashtable<String, Object> variables)</code>
Track Link: <code>String trackLink(String linkURL, String linkType, String linkName)</code> <code>String trackLink(String linkURL, String linkType, String linkName, Hashtable variableOverrides)</code>	Track Link: These two calls have been replaced with a single call. Pass null for unused values. <code>void trackLink(String linkURL, String linkType, String linkName, Hashtable<String, Object> contextData, Hashtable<String, Object> variables)</code>
: <code>void forceOffline();</code> <code>void forceOnline();</code>	Offline Tracking Methods: <code>void setOnline();</code> <code>void setOffline();</code>

Renamed Variables

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



Note: The 3.x version uses getters and setters instead of public variables. You'll need to update locations in your code where you set variables directly to use the get and set methods.

```

timestamp;           //Removed
trackOffline;       //void setOfflineTrackingEnabled(boolean Enabled)
offlineLimit;       //void setOfflineHitLimit(int offlineHitLimit)
account;            //reportSuiteIDs
linkURL;            //Removed (sent with linkTrackURL)
linkName;           //Removed (sent with linkTrackURL)
linkType;           //Removed (sent with linkTrackURL)
linkTrackVars;      //void setLinkTrackVars(String Vars) //comma-delimited list
linkTrackEvents;    //void setLinkTrackEvents(String Events) //comma-delimited list
dc;                 //Removed
trackingServer;     //void setTrackingServer(String trackingServer)
trackingServerSecure; //Removed, trackingServer value is used for secure server if ssl=YES
userAgent;          //Removed
dynamicVariablePrefix; //Removed
visitorID;          //void setVisitorID(String ID)
charSet;            //void setCharSet(String charSet)
visitorNamespace;   //Removed
pageName;           //void setAppState(String State)
pageURL;            //Removed
referrer;           //Removed
currencyCode;       //void setCurrencyCode(String currencyCode)
purchaseID;         //void setPurchaseID(String ID)
channel;            //void setChannel(String Channel)
server;             //void setAppSection(String Section)
pageType;           //Removed
transactionID;      //void setTransactionID(String ID)
campaign;           //void setCampaign(String Campaign)
state;              //void setGeoState(String GeoState)
zip;                //void setGeoZip(String GeoZip)
events;             //void setEvents(String Event) //comma-delimited list
products;           //void setProducts(String Products)
list1-list3;        //setListVar(int listNum, String listValue);
hier1-hier5;        //setHier(int hierNum, String hierValue);
prop1-prop75;       //setProp(int propNum, String propValue);
eVar1-eVar75;       //setEvar(int evarNum, String evarValue);
ssl;                //void setSSL(boolean ssl)
linkLeaveQueryString; //Removed
debugTracking;      //debugLogging
usePlugins;         //Removed
useBestPractices;   //Removed - handled by Lifecycle AutoTracking
contextData;        //persistentContextData

```

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

See [Bloodhound - App Measurement QA Tool](#) on Developer Connection.

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.



Note: *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.

Android Widgets

Android widgets can be tracked using the same methods as your app. Widgets share the application context with your app, so hit order and visitor identification is preserved.

The following guidelines will help you track Android widgets:

- Do not implement lifecycle metrics (`startActivity/stopActivity`) calls in the widget itself.
- To track when a widget is added to the home screen, add a `trackState` or `trackEvent` call to the `onEnabled` method of the widget.
- To track when the app is launched from a widget, add a `trackState` or `trackEvent` call before you create the intent to launch your application.
- If you are interested in tracking the context of an action, you can define a `ContextData` variable that provides the option of segmenting each out separately (for example, `AppExperienceType="widget"` vs. `"app"`).