ADOBE® DIGITAL MARKETING SUITE

AppMeasurement for iOS

IMPLEMENTATION GUIDE

Table of Contents

Preface

The Adobe® AppMeasurement for iOS Integration Guide describes using the AppMeasurement interface to measure the usage of your iPhone and iPad applications. It lets you capture certain types of activity in the application, and forward that data to Adobe data collection servers where it is available for use in SiteCatalyst® reports.

This guide is intended for application developers, and assumes that you are familiar with both implementing SiteCatalyst data collection code, and iOS application development.

Terms and Conditions of Use

This document and the related software described in this document is proprietary to Adobe Systems, Inc. and is supplied only under license or nondisclosure agreement. The document and software can be used or copied only in accordance with the terms of the agreement (Enterprise Terms of Use - https://sc.omniture.com/p/l10n/1.0/terms.html).

The information in this document is subject to change without notice and does not represent a commitment on the part of Adobe.

Account Support

ClientCare is available to:

- Answer specific product questions.
- Help you utilize SiteCatalyst reports to the greatest effect.
- Resolve any technical difficulties you might have.
- Help you configure SiteCatalyst variables.

Service and Billing Information

Depending on the service level you have purchased, some of the options described in this guide might not be available to you. Additionally, each account has unique billing needs. 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.

Adobe welcomes any suggestions or feedback you might have regarding AppMeasurement for iOS or the contents of this guide. Send comments to your Account Manager.

Contact Information

[CORPORATE ADDRESS]	Adobe Systems, Inc. 550 East Timpanogos Circle Orem, UT 84097
[PHONE]	1.801.722.7000
[FAX]	1.801.722.7001
[TOLL FREE]	1.877.722.7088 (support, billing and sales)
[SUPPORT E-MAIL]	clientcare@omniture.com
[SALES E-MAIL]	sales@omniture.com
[INFORMATION E-MAIL]	info@omniture.com
[CORPORATE URL]	http://www.omniture.com
[LOG-IN URL]	http://my.omniture.com

AppMeasurement for iOS

Adobe® developed AppMeasurement for iOS to provide a mechanism to capture usage of native Apple* iPhone* and iPad* applications in the Adobe Digital Marketing Suite. AppMeasurement for iOS lets you capture application events so you can monitor and evaluate the use of your iOS applications.

AppMeasurement for iOS leverages the standard iOS development tools to let you integrate iOS application measurement into both new and existing iOS applications.

This section includes the following topics:

- Requirements
- Install AppMeasurement for iOS Libraries
- Implement AppMeasurement for iOS
- Supported SiteCatalyst Reports

1.1 Requirements

- AppMeasurement for iOS consists of the following software components. You can download the AppMeasurement software from SiteCatalyst's Code Manager. For information about using Code Manager, see "Code Manager" in the *Admin Console User Guide*, which is available in the SiteCatalyst Help System.

  AppMeasurement.h: The Objective-C header file.

  libAppMeasurement.a: a fat binary containing the library builds for device (armv6 and armv7) as well as the simulator(i386).

  libAppMeasurementNoThumb.a: a fat binary built with THUMB instruction set disabled for compatibility with some third-party development frameworks.

- Adobe recommends the following for developing iOS applications. Consult the Apple Development Center for the latest iOS development information.
  - Intel-based Mac for application development
  - Mac OS X 10.6.0 or later
  - Xcode 3.0 or later
  - SDK for iOS 2.0 or later

1.2 Install AppMeasurement for iOS Libraries

Once you have AppMeasurement for iOS, complete the following steps to install the AppMeasurement for iOS libraries into an iOS application project.

1. Launch the Xcode IDE.
2. In the Groups & Files panel, right-click on the project where you want to add the AppMeasurement for iOS libraries, then click **Add** > **Existing Files**.
3. Browse to `AppMeasurement.h`, then click **Add**.
4. Select **Copy items into destination group's folder (if needed)**.
5. Set **Reference Type** to **Default**.
6. Set **Text Encoding** to **Unicode (UTF-8)**.
7. Select **Recursively create groups for any added folders**.
8. In the Add To Targets section, make sure your project is selected.
9. Click **Add**.
10. Complete the steps for Xcode 4.x or Xcode 3.x.

Xcode 4.x

11. Navigate to the Build Phases tab of your desired target.
12. Expand the Link Binary with Libraries item.
13. Click the + button.
14. Select the Add Other button.
15. Browse to the libAppMeasurement.a file and click Open.

Xcode 3.x

11. Open the Targets group (in the Groups & Files panel), right-click the project icon, then select **Add** > **Existing Frameworks**.
12. In the Linked Libraries section, click the Add Libraries icon (+) icon, then click **Add Other**.
13. Select `libAppMeasurement.a`, then click **Add**.
14. Select **Copy items into destination group's folder (if needed)**.
15. Set **Reference Type** to **Default**.
16. Set **Text Encoding** to **Unicode (UTF-8)**.
17. Select **Recursively create groups for any added folders**.
18. In the Add To Targets section, make sure your project is selected.
19. Click **Add**.

1.3 Implement AppMeasurement for iOS

Once installed, AppMeasurement for iOS provides the following variables and methods for configuring event tracking on your iOS application.

- Application Config Variables
- Track Config Variables
- Methods
- Offline AppMeasurement
- Plug-In Architecture (Optional)
- Implementation Examples

NOTE: When you set a variable directly on the object, the value acts as the default value until you change it on the object. However, you can use the variable overrides parameter to set a one-time-use value that persists only for the current track call, then resets to the default value. You cannot unset variables with variable overrides. For information about using variable overrides, see "Implementation Examples" on page 7.

Application Config Variables

The following variables are set when the application launches, and typically do not need to change. However, you can change them at any time, if necessary.

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

VARIABLE	REQUIRED	DESCRIPTION
`account`	Yes	The report suite or report suites (multi-suite tagging) that you wish to track. Separate multiple report suites with commas. You cannot override this variable using variable overrides. For example: `s.account = @"myrsid";`
`dc`	No	The Adobe data collection servers that receive the event tracking data. Consult your Account Manager to find out how you should set this value. 112 specifies the San Jose, CA collection servers, while 122 specifies the Dallas, TX collection servers. By default, this variable is 112 (San Jose). For example: `s.dc = @"112";`
`linkTrackEvents`	No	A comma-separated list of events that restricts the current set of events for link tracking. If `s.linkTrackEvents` is not defined, AppMeasurement for iOS sends all events. For example: `s.linkTrackEvents = @"purchase,event1";`

VARIABLE	REQUIRED	DESCRIPTION
`linkTrackVars`	No	A comma-separated list of variable names that restricts the current set of variables for link tracking. If `s.linkTrackVars` is not defined, AppMeasurement for iOS sends all variables with values. Even though `pageName` is sent, the Page View count does not increment. For example: `s.linkTrackVars = @"events,prop1,eVar49";`

`trackingServer`
 	
`trackingServerSecure`
	No	Identifies the collection domain.
		
		If you are using a first-party cookie in other SiteCatalyst implementations, set these variables to the same value as your other implementations.
		
		If you want to use these variables, you must set them both. You cannot set only one tracking server variable.
		
		For example:
		`s.trackingServer = @"metrics.corp1.com";`
		`s.trackingServerSecure = @"smetrics.corp1.com";`

`userAgent`	No	The HTTP User-Agent field that is sent with each track call to Adobe data collection servers. By default, AppMeasurement for iOS sets a default value for this variable that includes iOS device information and application name and version. If necessary, you can append additional information to this default value. For example: `s.userAgent = [NSString stringWithFormat:@"%@ more data", s.userAgent];`
`visitorNamespace`	No	The domain used to set cookies. For more information about namespace, see the **SiteCatalyst Implementation Guide**. For example: `s.visitorNamespace = @"mycompany";`
`currencyCode`	No	The Currency Code used for purchases or currency events that are tracked in the iOS application. For example: `s.currencyCode = @"USD";`
`visitorID`	No	Unique identifier for each visitor. Defaults to the iOS device ID. For example: s.visitorID = @`"12345";`
`linkLeaveQueryString`	No	Enables (`YES`) or disables (`NO`) preserving the query-string on the URL for links to Web pages from within the iOS application. By default, this variable is `NO`. You cannot override this variable using variable overrides. For example: `s.linkLeaveQueryString = YES;`
`ssl`	No	Enables (`YES`) or disables (`NO`) sending measurement data via SSL (HTTPS). By default, this variable is `NO`. You cannot override this variable using variable overrides. For example: `s.ssl = YES;`

VARIABLE	REQUIRED	DESCRIPTION
`debugTracking`	No	Enables (`YES`) or disables (`NO`) viewing debug information in the Xcode console. By default, this variable is `NO`. You cannot override this variable using variable overrides. For example: `s.debugTracking = YES;`
`usePlugins`	No	Enables (`YES`) or disables (`NO`) using plug-ins as part of a delegate implementation. By default, this variable is `NO`. You cannot override this variable using variable overrides. For example: `s.usePlugins = YES;`

Track Config Variables

The following variables are typically set before calling one of the track methods. For more information about these tracking variables, see the *SiteCatalyst Implementation Guide*.

`pageName`	`server`	`zip`
`pageURL`	`pageType`	`events`
`referrer`	`dynamicVariablePrefix`	`products`
`purchaseID`	`variableProvider`	`prop (1-75)`
`transactionID`	`campaign`	`eVar (1-75)`
`channel`	`state`	`hier (1-5)`

Methods

The following tracking methods send data to Adobe data collection servers.

NOTE: When using variable overrides, use the variable name as the key in a key-value pair. Also, use only strings as values in the key-value pairs.

METHOD	DESCRIPTION
`track`	Sends a standard page view to Adobe data collection servers, along with any Track Config Variables that have values. **NOTE:** AppMeasurement automatically creates a thread to execute `track`. Do not create a separate thread manually.
`track:`	Same as track, except you can pass in a list of key-value pairs that indicates temporary variable overrides for that track call.

`trackLink:linkType:linkName:`	Sends custom, download or exit link data to Adobe data collection servers, along with any track config variables that have values. Use `trackLink` to track all activity that should not capture a page view. `trackLink` has the following parameters: **url**: 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 nil for this parameter. **type**: Identifies the link report that will display the URL or name. Supported values include: • "o" (Custom Links) • "d" (File Downloads) • "e" (Exit Links) **name**: The name that appears in the link report. If no name is specified, the report uses the URL. **NOTE:** To collect data, you must specify either the `name` *or* `url` parameter. When not using one of these parameters, pass in `nil` as the value.

`trackLink:linkType:linkName:`
`variableOverrides:`
	Same as `trackLink`, except you can pass in a list of key-value pairs that indicates temporary variable overrides for that trackLink call.

`clearVars`	Clears the following track config variables on the object:
	
	`channel`
	`events`
	`purchaseID`
	`transactionID`
	`products`
	`state`
	`zip`
	`campaign`
	`props (1-75)`
	`eVars (1-75)`
	`hiers (1-5)`

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

Offline AppMeasurement

AppMeasurement lets you measure application usage even when the iOS device is offline. When enabled, Offline AppMeasurement behaves in the following way:

- The application generates a "hit" (a `track()` or `tracklink()` method 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 500-millisecond intervals (whichever is shorter), 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).

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.

Offline measurement uses the following methods and variables:

Table 1.1: Offline Measurement Variables

NAME	TYPE / DEFAULT VALUE	DESCRIPTION
`trackOffline`	Boolean / false	Enables/Disables offline AppMeasurement.
`offlineLimit`	Integer / 1000	Sets the maximum number of hits to buffer. When the number of buffered hits reaches `offlineLimit`, AppMeasurement drops the oldest buffered hit, then buffers the newest hit. Tune this value based on the amount of data (the number of variables) sent with each hit.
`offlineThrottleDelay`	Integer / 0	Specifies a cadence (or delay), in milliseconds, for sending buffered hit data when AppMeasurement detects an active network connection. Doing so mitigates the performance impact of sending multiple hits on the application. For example, if offlineThrottleDelay=1000, and it takes 300ms to send the hit data, AppMeasurement waits 700ms before sending the next buffered hit.
`timestamp`	Integer / 0	Overrides the default timestamp mechanism in AppMeasurement so you can integrate the application's offline measurement with an existing queue system. Adobe recommends letting AppMeasurement manage timestamps, if possible.

Table 1.2: Offline Measurement Methods

NAME	DESCRIPTION
`forceOffline()`	Forces AppMeasurement to behave as if it is offline. This lets you delay sending hit data during a hardware-intensive operation, such as video playback. This lets you collect the data offline, and then send it once the operation completes.
`forceOnline()`	Returns AppMeasurement to normal operating mode after using the `forceOffline()` method.

1.4 Implementation Examples

The following examples demonstrate some common use cases for AppMeasurement for iOS:

(Instance instantiation) Declare instance

Declare instance as either a class ivar (in class header file), a global/static variable, or as a singleton.

```
AppMeasurement * s;
```

(Instance instantiation) Instantiate instance and setup account configuration variables

The most likely place to do this is in the `applicationDidFinishLaunching` method of your AppDelegate class, but can be done elsewhere.

```
- (void)applicationDidFinishLaunching {
      //Instantiate instance
      s = [[AppMeasurement alloc] init];
      //Setup application config variables
      s.account = @"myaccount";
      s.ssl = YES;
}
```

The following instantiation example uses the singleton interface. The singleton interface is useful to prevent multiple instances, but it should not be used if you are sending data to multiple companies as multiple instances are required.

```
- (void)applicationDidFinishLaunching {
      //Instantiate instance
      AppMeasurement * s = [AppMeasurement getInstance];
      //Setup application config variables
      s.account = @"myaccount";
      s.ssl = YES;
}
```

(track) Send a Page View Only

The most likely place to do this is in the `viewDidLoad` method of your ViewController class, but can be done anywhere.

```
- (void)viewDidLoad {
      //Set Variables
      s.pageName = @"Some Page Name";
      s.channel = nil;//clears any previously set value for channel
      s.prop1 = nil;//clears any previously set value for prop1
      [s track];
}
```

(track) Send a Page View with Channel Information

The most likely place to do this is in the `viewDidLoad` method of your ViewController class, but can be done anywhere.

```
- (void)viewDidLoad {
      //Set Variables
      s.pageName = @"Some Page Name";
      s.channel = @"Some Site Section";
      [s track];
}
```

(track) Send a Page View with Channel and Custom Insight (prop) Information

The most likely place to do this is in the `viewDidLoad` method of your ViewController class, but can be done anywhere.

```
- (void)viewDidLoad {
      //Set Variables
      s.pageName = @"Some Page Name";
      s.channel = @"Some Site Section";
      s.prop1 = @"Some Value";
      [s track];
}
```

(track) Send pageView data using variable overrides

Specifies a temporary variable value used only for track call.

```
- (void)viewDidLoad {
      //Create an NSMutableDictionary, set values on it corresponding to tracking
variable keys, pass it into track call
      NSMutableDictionary * trackData = [[NSMutableDictionary alloc] init];
      [trackData setObject:@"Some page name" forKey:@"pageName"];
         //temporary override
      [trackData setObject:@"Some Prop Val" forKey:@"prop1"];
         //temporary override
      [s track:trackData];
      [trackData release];
}

- (void)viewDidLoad {
      //Create NSDictionary inline with keys/values, pass it directly into track call
(only temporary override)
      [s track:[NSDictionary dictionaryWithObjectsAndKeys:@"Some page name",
@"pageName", @"Some Prop Val", @"prop1", nil]];
      //must 'nil-terminate' dictionary when creating it inline
}
```

(trackLink) Send a Custom Link

```
- (void)someNonPageViewAction {
      [s trackLink:nil linkType:@"o" linkName:@"Some Action Name"];
      //may pass in 'nil' for linkURL if it doesn't apply
}
```

(trackLink) Send a Custom Link with Channel Information

```
- (void)someNonPageViewAction {
      //Set Variables
      s.channel = @"Some Application Section";
         [s trackLink:nil linkType:@"o" linkName:@"Some Action Name"];
         //may pass in 'nil' for linkURL if it doesn't apply
}
```

(trackLink) Send a Download Link with Channel and Custom Insight (prop) Information

```
- (void)someNonPageViewAction {
      //Set Variables
      s.channel = @"Some Application Section";
      s.prop1 = @"Snoop";
      [s trackLink:@"http://www.somedownloadURL.com" linkType:@"d" linkName:@"Some
Action Name"];
}
```

(trackLink) Send an Exit Link with no variables

```
- (void)someNonPageViewAction {
      //Set Variables
      s.channel = nil;
         //clears any previously set value for channel
      s.prop1 = nil;
         //clears any previously set value for prop1
      [s trackLink:@"http://www.someexitdomain.com" linkType:@"e" linkName:@"Some
Action Name"];
}
```

1.5 Plug-In Architecture (Optional)

AppMeasurement for iOS supports a plug-in architecture that lets you build custom plug-ins that extend AppMeasurement for iOS functionality. It implements the Objective-C delegate design pattern. Use this functionality to modify the AppMeasurement for iOS object to manipulate data before sending it to Adobe data collection servers.

The following example demonstrates how to link the delegate object to the AppMeasurement for iOS object:

```
      AppMeasurement * s = [[AppMeasurement alloc] init];
         s.delegate = self;
         s.ssl = YES;
         s.usePlugins = YES;
         s.account = @"myreportsuiteid";
         s.pageName = @"Main View";
         [s track];

// After instantiating the object, have your delegate object implement the
AppMeasurementDelegate protocol

@interface MyClass : NSObject<AppMeasurementDelegate>

...

- (void)appMeasurementDoPlugins:(AppMeasurement *)s
{
      // custom code
      s.prop4 = @"plugin";
}
```

1.6 iOS Best Practices Plug-In

Adobe Best Practices iOS App Measurement plug-in provides a simple way to measure many common metrics, including many churn metrics, for your iOS app with only a few lines of code.

Code Samples

The code required to use the iOS Best Practices Plug-in is included in the AppMeasurement iOS libraries you downloaded from Code Manager.

In your AppDelegate implementation file, you need to add the code listed in this section to your project. After you add the code, review the customization instructions to select which variables, events, and props are used to store the data sent by the plug-in. See "Customization Example" on page 11.

```
#import "AppMeasurement.h"
...
- (BOOL)application:(UIApplication
*)applicationdidFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
// init app measurement
AppMeasurement* s = [AppMeasurement getInstance];
s.account = "account";
s.useBestPractices = YES;
...
}
```

Customization Example

To customize which variables, events, and props are populated by the plug-in, you need to assign the selected name to the appropriate variable on the ChurnMeasurement object. The following example changes the specified variables, events, and props, but leaves the rest of the defaults:

```
AppMeasurement *s = [AppMeasurement getInstance];
ChurnMeasurement * c = [getChurnInstance];
c.appInstallDateEvar = @"eVar5";
c.daysSinceLastUpgradeEvar = @"eVar7";
c.appIdEvar = nil;  // omits appIdEvar data completely from the request
c.appLaunchEvent = @"event4";
c.appLaunchNumberProp = @"prop2";
s.useBestPractices = YES;
```

if you would like only a few variables, events, or props populated, you can use the populateDefaults flag to turn off defaults being populated (the default behavior is to populate all defaults). After turning this flag off, simply set the variables that you want to use. In the following example, only eVar1, event1, and event2 (and pageName) are sent on the plug-in request.

```
AppMeasurement * s = [AppMeasurement getInstance];
ChurnMeasurement * c = [s getChurnInstancePopulateDefaults:NO];
c.appInstallEvent = @"event1";
c.appInstallDateEvar = @"eVar1";
c.appCrashEvent = @"event2";
s.useBestPractices = YES;
```

The following list contains general notes regarding ChurnMeasurement variables:

- Variables may be turned off if desired by setting the variable to nil (some variables being turned off will cause other variables to be turned off automatically, because their values are used by the other).
- Even if all variables are turned off, pageName will still be set on the plug-in request.
- "Days Since" metrics are determined by 24-hour incremental periods from the first use (install). "Days Since" metrics are not sent on subsequent launches of the event in question (install, upgrade, etc). "Days Since" metrics are relative to the first use after an event (install, upgrade, etc), not the actual install/upgrade of the app on the device.

The following table lists the variables, events, and props (including the default value) that can be set on the ChurnMeasurement object:

NAME	DEFAULT VALUE	DESCRIPTION
appInstallEvent	event1	Triggered on first run after installation (or re-installation).
appUpgradeEvent	event2	Triggered on first run after upgrade (anytime the version number changes).
dailyEngagedUserEvent	event3	Triggers an event when the application is used on a particular day.
monthlyEngagedUserEvent	event4	Triggers an event when the application is used during a particular month.
appLaunchEvent	event5	Triggered on any run that is not an install or an upgrade. This also triggers when the application is brought out of the background.
appScreenViewEvent	event6	Provides a replacement for page view.
appCrashEvent	event7	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).
appInstallDateEvar	eVar1	Date of first launch after installation. MM/DD/YYYY
appIdEvar	eVar2	Stores the Application name and version in the following format: [AppName] [BundleVersion] For example, `myapp 1.1`
engagedDaysLifetimeEvar	eVar3	Stores the number of days the application is used.
daysSinceFirstUseEvar	eVar4	Number of days since first run.
daysSinceLastUseEvar	eVar5	Number of days since last use.
appLaunchNumberEvar	eVar6	Number of times the application was launched or brought out of the background.
hourOfDayEvar	eVar7	Measures the hour the app was launched. 24 hour numerical format. Used for time parting to determine peak usage times.
dayOfWeekEvar	eVar8	3 character day when the application was used.
appEnvironmentEvar	eVar9	Contains the current iOS version.
daysSinceLastUpgradeEvar	eVar10	Number of days since the application version number has changed.

NAME	DEFAULT VALUE	DESCRIPTION
appLaunchNumberSinceLastUpgradeEvar	eVar11	Number of launches since the application version number has changed.
engagedDaysMonthEvar	eVar12	Total count of days the application was used in the last month.
engagedDaysLastUpgradeEvar	eVar13	Total count of days the application was used since last upgrade.
appIdProp	prop1	Stores the Application name and version in the following format: [AppName] [BundleVersion] For example, `myapp 1.1`
appLaunchNumberProp	prop2	Number of times the application was launched or brought out of the background.
appLaunchNumberSinceLastUpgradeProp	prop3	Number of launches since the application version number has changed.
PageName	n/a	Contains "appName (appVersion) useType" where useType is either Install, Upgrade, or Launch.

1.7 Supported SiteCatalyst Reports

The following SiteCatalyst reports include data submitted by AppMeasurement for iOS. For more information about these reports, see the *SiteCatalyst User Guide*.

Site Metrics Report Support

- Page Views
- Hourly Unique Visitors
- Daily Unique Visitors
- Monthly Unique Visitors
- Yearly Unique Visitors
- Visits
- File Downloads

Finding Methods Report Support

NOTE: These reports are available only if the developer has access to the referrer information when developing the application, which might not be feasible in some circumstances.

- Referring Domains
- Referrers
- Return Frequency
- Daily Return Visits
- Return Visits
- Visit Number

Visitor Profile Report Support

- Domains
- Full Domains
- Top Level Domain
- Languages
- Time Zones
- Visitor Detail
- Last 100 Visitors

GeoSegmentation Report Support

- Country
- State
- City

Mobile Technology Report Support

- Devices

- Manufacturer
- Screen Size
- Screen Height
- Screen Width
- Cookie Support
- Image Support
- Color Depth
- Audio Support
- Video Support

Segmentation Report Support

- Most Popular Pages
- Most Popular Channels
- Most Popular Servers
- Key Visitors
- Pages Viewed by Key Visitors
- Custom Insight
- Custom Links
- Custom Insight Properties 1-50

Pages Report Support

- Page Summary
- Most Popular Pages
- Clicks to Page
- Time Spent on Page
- Pages Not Found

Entries & Exits

- Entry Pages
- Exit Pages
- Exit Links

Complete Paths Report Support

- Full Paths
- Longest Paths
- Path Length
- Time Spent per Visit
- Single-page Visits

Advanced Analysis Report Support

- Previous Page
- Next Page
- Previous Page Flow
- Next Page Flow
- PathFinder
- Fall-out

Purchases

- Conversions & Averages
- Revenue
- Orders
- Units

Shopping Cart

- Conversions & Averages
- Carts
- Cart Views
- Cart Additions
- Cart Removals
- Checkouts

Products Report Support

- Conversions & Averages
- Products
- Cross-Sell
- Categories

Campaigns

- Conversions & Averages
- Campaigns

Customer Loyalty

- Customer Loyalty

Sales Cycle

- Days Before First Purchase
- Days Since Last Purchase
- Visit Number
- Daily Unique Customers
- Monthly Unique Customers

- Yearly Unique Customers

Finding Methods

- Referring Domains
- Original Referring Domains

Visitor Profile

- Countries
- Languages
- Time Zones
- States
- ZIP/Postal Codes
- Domains

Site Path

- Entry Pages
- Original Entry Pages
- Pages per Visit
- Time Spent on Site
- Content Groups

Custom Variables

- Custom E-Commerce Variables 1-50

Video Tracking in iOS Apps

AppMeasurement for iOS lets you capture data about video usage in your iOS application. To do this, AppMeasurement for iOS gathers basic information from the media player, then builds a session of events and sends it to Adobe data collection servers for processing. After the collection servers process the video data, it is accessible through a set of SiteCatalyst video reports.

This section contains the following topics:

- iOS Video Tracking Variables
- iOS Video Tracking Methods
- Implementation Examples

2.1 iOS Video Tracking Variables

AppMeasurement for iOS includes the Media Module, which provides several video tracking variables that help you configure the level of video tracking you need. Consider the following when working with video tracking variables in iOS apps:

- Video Tracking for iOS Apps does not currently support video auto-tracking due to restrictions in the default iOS video player (`MPMoviePlayerController`).
- The `Media.trackWhilePlaying` variable is always true for video tracking in iOS apps. Because it cannot be set by the developer, this variable is excluded from Table 2.1.

Table 2.1: iOS Video Tracking Variables

VARIABLE	DESCRIPTION
`Media.playerName`	**Syntax:** `s.Media.playerName = @"Custom Player Name";` Specifies a custom video player name.
`Media.trackVars`	**Syntax:** `s.Media.trackVars = @"events,prop1,prop5";` Specifies a comma-separated list of variables to send with the video tracking data. **NOTE:** To use `Media.trackEvents`, you must include `events` as one of the variables in `Media.trackVars`. For more information, see the *SiteCatalyst Implementation Guide*, available in the SiteCatalyst Help System.
`Media.trackEvents`	**Syntax:** `s.Media.trackEvents = @"event2,event3";` Use when `Media.trackVars` includes the `events` variable. Specifies a comma separated list of events to send with the video tracking data. For more information, see the *SiteCatalyst Implementation Guide*, available in the SiteCatalyst Help System.

VARIABLE	DESCRIPTION
`Media.trackSeconds*`	**Syntax:** s.Media.trackSeconds = 15; Defines the interval, in seconds, for sending video tracking data to Adobe data collection servers while the video is playing. The value must be set in increments of 5 seconds. The variable sends video tracking data at the start (when `Media.open` is called) and end (when `Media.close` is called) of the video, and on multiples of the value specified for `Media.trackSeconds`. For example, every 15 seconds that the video is playing. `Media.trackSeconds` tracks time spent viewing a video. It does not track how much of the video a visitor views. It does not distinguish between viewing the file from beginning to end, and replaying a portion of the video multiple times. **NOTE:** Do not use this variable simultaneously with `Media.trackMilestones`.
`Media.trackMilestones*`	**Syntax:** `s.Media.trackMilestones = @"10,50,90";` Defines the interval, as a percentage of the video watched, for sending video tracking data to Adobe data collection servers. Specify the milestones as a comma-separated list of whole numbers. For example: 10 = 10%, 23 = 23%). The variable sends video tracking data at the start (when `Media.open` is called) and end (when `Media.close` is called) of the video, and on video percentages specified in `Media.trackMilestones`, based on the length of the video. Because these milestones are fixed points in the video, if a visitor views past the 10% milestone, then rewinds and passes the 10% milestone again, Adobe sends the tracking data multiple times. Similarly, if a visitor fast forwards past a milestone, Adobe does not send the tracking data for that milestone. **NOTE:** Do not use this variable simultaneously with `Media.trackSeconds`.
`Media.cuePoints`	**Syntax:** `s.Media.cuePoints = @"5:first,17:second";` Defines cue points in the video by specifying a second offset and, optionally, a cue point name. For example, `5:first` defines a cue point 5 seconds into the video with a name of "first".
`Media.trackAtCuePoints*`	**Syntax:** `s.Media.trackAtCuePoints = YES;` `Media.trackAtCuePoints` defaults to `NO`. Set this variable to `YES` to automatically send data to Adobe data collection servers at each of the defined cue points.

* Enabling these variables does not automatically send custom variables (Props, eVars, Events) to Adobe data collection severs. To do this, you must use the Media Monitor (see "`appMeasurementModuleMediaMonitor`" on page 21).

2.2 iOS Video Tracking Methods

AppMeasurement for iOS includes the Media Module, which provides several video tracking methods that help you configure the level of video tracking you need.

Table 2.2: iOS Video Tracking Methods

METHOD	DESCRIPTION
`Media.open`	**Syntax:** `[s.Media open:mediaName length:mediaLength playerName:mediaPlayerName cuePoints:cuePoints playerID:playerID];` Prepares the media module to collect the following video tracking data. This method takes the following parameters: **mediaName**: The name of the video as you want it to appear in SiteCatalyst video reports. **mediaLength**: The length of the video in seconds. **mediaPlayerName**: The name of the media player used to view the video, as you want it to appear in SiteCatalyst video reports. **cuePoints:** (Optional) Defines cue points in the video by specifying a second offset and, optionally, a cue point name. This is identical to setting the `Media.cuePoints` variable. **playerID:** (Optional) A unique identifier for the video player. If you have multiple video players in your application, use this parameter so Adobe can distinguish between them.
`Media.close`	**Syntax:** `[s.Media close:mediaName];` Ends video data collection and sends information to Adobe data collection servers. Call this method at the end of the video. This method takes the following parameters: **mediaName**: The name of the video as you want it to appear in SiteCatalyst video reports.
`Media.play`	**Syntax:** `[s.Media play:mediaName offset:mediaOffset];` Call this method anytime a video starts playing. This method takes the following parameters: **mediaName**: This must match the name used in Media.open. **mediaOffset**: The number of seconds into the video that play begins. Specify the offset based on the video starting at second zero.
`Media.stop`	**Syntax:** `[s.Media stop:mediaName offset:mediaOffset];` Tracks a stop event (stop, pause, etc.) for the specified video. This method takes the following parameters: **mediaName**: The name of the video as you want it to appear in SiteCatalyst video reports. **mediaOffset**: The number of seconds into the video that the stop or pause event occurs. Specify the offset based on the video starting at second zero.

METHOD	DESCRIPTION
`appMeasurementModuleMediaMonitor`	**Syntax:** -(void)appMeasurementModuleMediaMonitor:(AppMeasurement *)s media:(AppMeasurementModuleMediaState *)media The iOS app media monitor implements the Objective-C delegate design pattern. Use this functionality to monitor the status of each video that is currently playing. With it, you can setup other variables (Props, eVars, Events) and call `Media.track` based on the current state of the video as it is playing. This is useful if you have code that tracks events, such as resizing and volume adjustments, that you want to send when the video finishes playing. This method takes the following parameters: **s:** The `AppMeasurement` instance. **media:** An `AppMeasurementModuleMediaState` object with members providing the state of the video. These members include: • **media.name**: The name of the video given in the call to `Media.open`. • **media.length**: The length of the video in seconds given in the call to `Media.open`. • **media.playerName**: The name of the media player given in the call to `Media.open`. • **media.mediaEvent**: A string containing the event name that caused the monitor call. These events are: ◦ OPEN: The first call to `Media.open`. ◦ CLOSE: When `Media.close` is called. ◦ PLAY: When `Media.play` is called. ◦ STOP: When `Media.stop` is called. ◦ MONITOR: When our automatic monitoring checks the state of the video while it's playing (every second). ◦ SECONDS: At the second interval defined by the `Media.trackSeconds` variable. ◦ MILESTONE: At the milestones defined by the `Media.trackMilestones` variable. ◦ CUEPOINT: At the cue points defined by the `Media.cuePoints` variable (or the `Media.open` method). • **media.openTime**: An `NSDate` object containing data about when `Media.open` was called. • **media.offset**: The current offset, in seconds, (actual point in the video) into the video. The offset starts at zero (the first second of the video is second 0). • **media.percent**: The current percentage of the video that has played, based on the video length and the current offset. • **media.timePlayed**: The total number of seconds played so far. • **media.eventFirstTime**: Indicates if this was the first time this media event was called for this video. • **media.offsetName**: The cue point name. This is set only when the media event is a cue point.

METHOD	DESCRIPTION
`Media.track`	**Syntax:** `[s.Media track:mediaName];` Immediately sends the current video state, along with any `Media.trackVars` and `Media.trackEvents`. Call `Media.open` and `Media.play` on the video before calling this method. This method takes the following parameter: **mediaName**: The name of the video as you want it to appear in SiteCatalyst video reports. Make sure you pass the same `mediaName` used to call `Media.open`. This method is the only way to send in other variables while the video is playing. This method resets the seconds interval and percent milestone counters to zero to prevent multiple tracking hits.
`Media.cuePoint`	**Syntax:** `[s.Media cuePoint:mediaName offset:cuePointOffset cuePointName:cuePointName];` Manually fires a cue point. This method takes the following parameter: **mediaName**: The name of the video as you want it to appear in SiteCatalyst video reports. Make sure you pass the same `mediaName` used to call `Media.open`. **cuePointOffset:** An offset into the video, in seconds. **cuePointName:** (Optional) A name for the cue point.

2.3 Implementation Examples

The following code samples demonstrate the use of video tracking in an iOS application:

- Configuring Video Tracking
- Using Media Monitor for Advanced Measurement

Configuring Video Tracking

```
#import "AppMeasurement.h"

AppMeasurement * s = [[AppMeasurement alloc] init];
s.account = @"myreportsuiteID";

// ...

s.Media.playerName = @"My Media Player";// custom media player name
s.Media.trackMilestones = @"25,50,75";// could instead use s.Media.trackSeconds if
desired

// cue points may also be set if desired
s.Media.cuePoints = @"5:first,15:second,28:third";
s.Media.trackAtCuePoints = YES;

// ...

// mediaName, mediaLength, mediaPlayerName, and mediaOffset are defined elsewhere.
// call on video load
- (void)startMovie
{
    [s.Media open:mediaName length:mediaLength playerName:mediaPlayerName];
    [self playMovie];
}

// call on video resume from pause and slider release
- (void)playMovie
{
    [s.Media play:mediaName offset:mediaOffset];
}

// call on video pause and slider grab
- (void)stopMovie
{
    [s.Media stop:mediaName offset:mediaOffset];
}

// call on video end
- (void)endMovie
{
    [self stopMovie];
    [s.Media close:mediaName];
}
```

Using Media Monitor for Advanced Measurement

```
s.Media.delegate = self;// should be assigned to the object where the media monitor
method is defined

// ...

// custom variables (Props, eVars, Events) are sent automatically on an OPEN event, and
when manually tracked here (in media monitor below)
- (void)appMeasurementModuleMediaMonitor:(AppMeasurement *)s
media:(AppMeasurementModuleMediaState *)media
{
    if ([media.mediaEvent isEqualToString:@"OPEN"]) {// executes when the video opens
        s.Media.trackVars = @"eVar1,events";
        s.Media.trackEvents = @"event1";
        s.events = @"event1";
        s.eVar1 = media.name;
        [s.Media track:media.name];
    }

    if ([media.mediaEvent isEqualToString:@"CLOSE"]) {// executes when the video
completes
        s.Media.trackVars = @"eVar1,events";
        s.Media.trackEvents = @"event4";
        s.events = @"event4";
        s.eVar1 = media.name;
        [s.Media track:media.name];
    }
}
```