Adobe® Marketing Cloud

Measuring Video in Adobe Analytics

Contents

Measuring Video in Adobe Analytics

New! Adobe has released a new way to measure video. See *Heartbeat Video*.

Adobe Analytics provides native support for measuring the most popular video formats on the Web. Almost any other player and video format can be measured using JavaScript. You can start measuring video using your existing video player and content.

The first section of this guide walks you through the analytics decisions you need to make to measure video, and then finishes with an implementation worksheet to deliver to the video developer. The second section provides the in-depth details for the video developer who implements the measurement code.

Video measurement tips, tricks and best practices on the Digital Marketing Blog:

- *Why Video Measurement Matters*
- *Creating The Perfect Plan for Video Measurement*
- *Understanding Video Measurement Implementation*
- *Initializing Video Measurement Success*
- *Navigating Video Measurement Analysis*

Recent updates to this guide:

Date	Update
02/21/2013	Added `Media.openAd`, `Media.click`, and several `media.ad` contextData variables to support a video ad tracking. See *VAST Video Ad Tracking*.
9/13/2012	Added a note that in order for JavaScript AutoTrack to work, you must have the `classid` attribute set on the object you want to track. The `classid` is required to expose the event handlers used by the Media Module to automatically track the video. Added a note that autoTrack for Windows Media Player works only with Internet Explorer. Manual tracking for Windows Media Player is required to support other browsers. Added details on the OSMF XML `autoBind` attribute that lets you start and end string literals using curly braces. See *Using OSMF Metadata to Override a Video Name*.

8/6/2012	We now recommend setting the `trackVars` and `trackEvents` variables for all implementations, even if `Media.monitor` is not being used. Populate `trackVars` with a list of each prop and eVar used in your implementation, along with the string "events":
	`s.Media.trackVars="events,prop2,eVar1,eVar2,eVar3";`
	. Populate `trackEvents` with a list of all events used in your implementation:
	`s.Media.trackEvents="event1,event2,event3,event4,event5,event6,event7"`

7/19/2012	Added links to the iOS and Android 3.x video documentation.

January 2012: New Process to Track Video Completes

You must make a small modification to your tracking code to enable the new functionality.

Why is this change needed?

Previously the 100% milestone was used to indicate a complete view. However, due to the granular nature of time tracking in video, some players never reported an offset that equaled the total length of the video. This prevented the 100% milestone from being reached even when the complete video was viewed.

To avoid this, completes are now tracked using an offset from the end of the video. This change should result in more accurate tracking of video completes.

What do I need to change?

After you update to the new version of the libraries, the new method of tracking completes is enabled with an offset equal to 1 second. In you code, you need to change your contextDataMapping to define the event that is used to track completes using the new `a.media.complete` variable.

To make this change in your code, find the `Media.contextDataMapping` section:

```
s.Media.contextDataMapping = {
```

Remove the 100% milestone (or whatever percentage you defined as complete) from `a.media.milestones`. Save the event value (event7 in this example) as it is used in the next step:

```
 "a.media.milestones":{
     25:"event4",
     50:"event5",
     75:"event6",
     100:"event7" (remove this line)
 };
```

Add `a.media.complete` and map the event value previously defined for the complete milestone. The `Media.contextDataMapping` section should appear similar to the following:

```
s.Media.contextDataMapping = {
  "a.media.name":"eVar2,prop2",
  "a.media.segment":"eVar3",
  "a.contentType":"eVar1",
  "a.media.timePlayed":"event3",
  "a.media.view":"event1",
  "a.media.segmentView":"event2",
  "a.media.complete":"event7",
  "a.media.milestones":{
     25:"event4",
     50:"event5",
     75:"event6"
 };
```

If you aren't sure which event is used to track completes, you can check your SiteCatalyst video configuration in the Admin Console.

After you make this change the complete event is sent 1 second before the end of a video.

How Video Measurement Works

This topic provides a brief overview of video measurement.

On the Web, JavaScript is added to pages on your site to enable measurement. Data is sent when a page is visited, or when a specific action occurs (for example, something is added to the shopping cart). The data sent by this code is analyzed by Analytics to determine the order pages were viewed, and how long viewers were on a particular page.

Similarly, code is added to your video player to enable video measurement. For videos, data is sent when a video is started, closed, and at specific intervals or percentage-based milestones during video playback. To measure video, you add code (called the media module) to your video player. The media module is available in multiple formats to support a variety of video players. For example, the media module is available in ActionScript for Flash, as a plug-in for OSMF, in .NET for Silverlight, and in JavaScript for other Web players (Windows Media Player, Quicktime, and others).

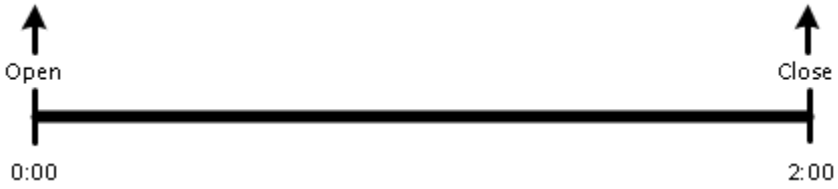
If you have the source code for the player (for example, a custom OSMF player or a direct Flash NetStream implementation), you can compile the media module directly into the player. For other players you can integrate using a plug-in interface or by using event handlers that are exposed by the player.

Video Metrics

This section describes the metrics available to measure video.

Measuring Video Views and Time Played

A basic implementation tracks video views and time played by sending a server call when a video is opened and closed. For a complete view, this results in a call when the video is opened and when it is closed.



This provides the data to track total views (a view event is sent on video open) and time viewed (the total time viewed is sent on close).

When data is sent, the time viewed tracks total 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.

This works for shorter video clips and when you are mostly interested in total views. To gain addition insights, you can divide a video into segments and track key milestones (such as complete views).

Measuring Video Segments

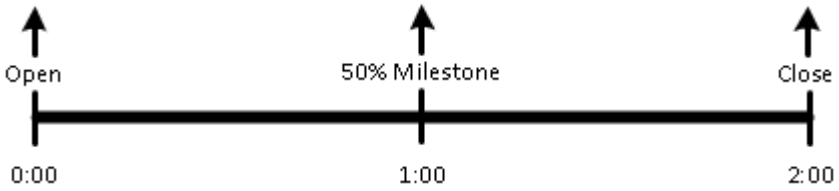
Segments let you divide an individual video into multiple parts for measurement purposes. This can provide a more granular view of how a particular video is being viewed and help you track video fall out. If you have mostly 30 or 60-second clips, you might not need segments. However, if you are measuring a sporting event, you might be very interested in comparing video data in the first quarter to video data in the fourth quarter.

See the *Video Detail Report* to see how segment data is used to provide video insights.

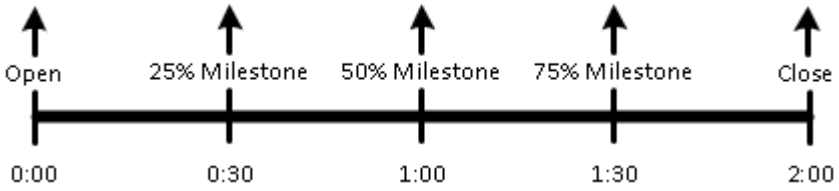
Measuring Key Milestones

Milestones let you measure when a specific location in a video is viewed. When a milestone is viewed a server call is sent containing the milestone event defined for that milestone. Milestones are defined as a percentage of total video length. Each milestones is tracked using a custom event. You need to select a custom event for each milestone you want to track.

If you define a 50% milestone for a 2 minute video, calls are made at the following points:



The 50% milestone event and time viewed is reported at 1 minute. The remaining time viewed is sent at the video end. If you define milestones at 25%, 50%, 75%, calls are made at the following points:



In this example two additional calls are made containing the 25% and 75% milestone events and the time viewed.

An additional benefit to tracking milestones is that time viewed is sent incrementally. When you are tracking only the video open and close, the time viewed is not reported until the video is closed (when the user opens a new video or the video ends). If the player is closed unexpectedly (for example, the browser window is closed), no time viewed is reported.

In the milestone example, if the player closed unexpectedly at 1:10, 1 minute of time viewed would be measured. The 10 seconds that occurred after the 50% milestone would not be measured.

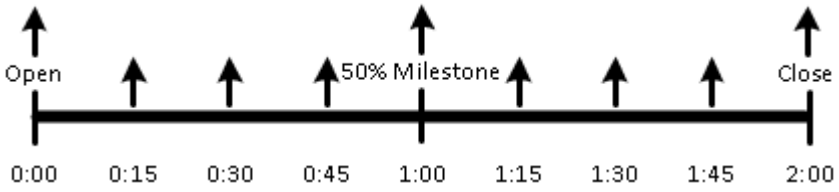
Because these milestones are fixed points in the video, if a visitor views past the 50% milestone, then rewinds and passes the 50% milestone again, the milestone event is sent multiple times. Similarly, if a visitor skips past a milestone, an event is not sent for that milestone.

Measuring Video Completes

Defining a complete event lets you track the number of viewers who view the end of a video. By default, if you define a video complete event it is sent 1 second before the end of the video.

Track Seconds

Tracking seconds lets you send video data at second-based intervals throughout your video. Tracking seconds can be used with or without video milestones. For example, if you track a 50% milestone, and then specify a track seconds interval of 15, calls would be made at the following points:



Measuring Video for Web Analysts

This topic provides an overview of the video measurement implementation process for a web analyst.

Measuring video on your Web site involves two functional teams. First, the web analytics team reviews the video reports, determines how often video data should be sent to Adobe collection servers, and selects which commerce variables and custom events should be dedicated to video measurement. Next, the video development team uses the media module to send data when videos are viewed on your web site.

The following table lists the process to measure video for the web analytics team:

Step	Task	Details
Step 1	Review the video reports to understand the video metrics you can measure.	*Video Reports* on page 9
Step 2	Define the video segments and milestones you want to measure.	*Video Metrics* on page 7
Step 3	Configure the video measurement reports.	To measure video, you need to dedicate: • 3 commerce variables (eVar) • 4 custom events • 1 custom insight (s.Prop) *Video Configuration*
Step 4	Complete the implementation worksheet for the video development team.	*Video Implementation Worksheet* on page 18

Video Reports

Analytics provides several reports and metrics to track video performance on your Web site.

Video reports are listed in the **Reports** > **Video** section.

Why are the Video Engagement Reports marked as "Beta"?

We are working to improve the functionality and layout of these reports, so expect to see some minor changes to these reports in future releases. The data that appears on these reports is accurate and is not impacted by the beta status.

Video Engagement Reports (Beta)

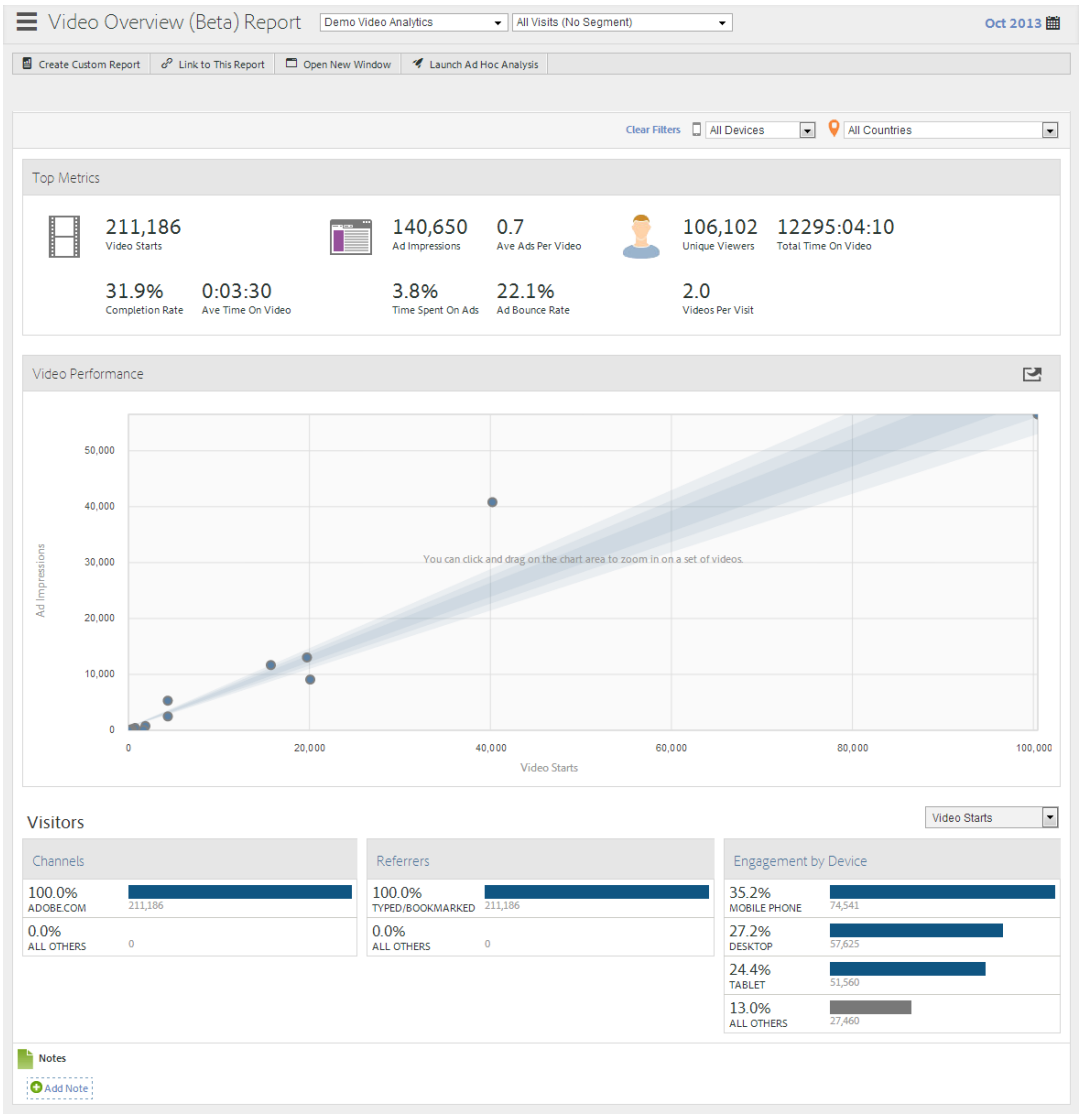
Video Report	Description	Common Business Insights
Video Overview Report	The Video Overview Report displays several aggregate measurements to quickly monitor that video is performing as expected. A graph displays video starts next to ad impressions to let you quickly view these metrics for each video.	• Totals for top video metrics including unique viewers, completion rate, average video metrics, and average videos per visit. • Total video and ad views for specific videos filtered by device type or country.

Video Report	Description	Common Business Insights
Video Detail Report	Displays detailed metrics for all videos including starts, completion rate, play percentage, and ad impressions.	• Totals for top video metrics including video starts, ad impressions, average ads per video. • Top videos by multiple metrics
Video Daypart	Displays unique visitors and video views by time of day to let you quickly view when your audience is engaged.	• Audience engagement by time of day. • Audience engagement compared to previous date ranges.
Video Metrics and Video Dimensions	Video metrics and dimensions are standard Analytics variables that can be reported directly and added to other Analytics reports.	• Video Conversion (Events that occur after video is viewed) by generating a report with visits that include a content type of video. • Next/previous video flow using the video name prop.

Video Overview Report

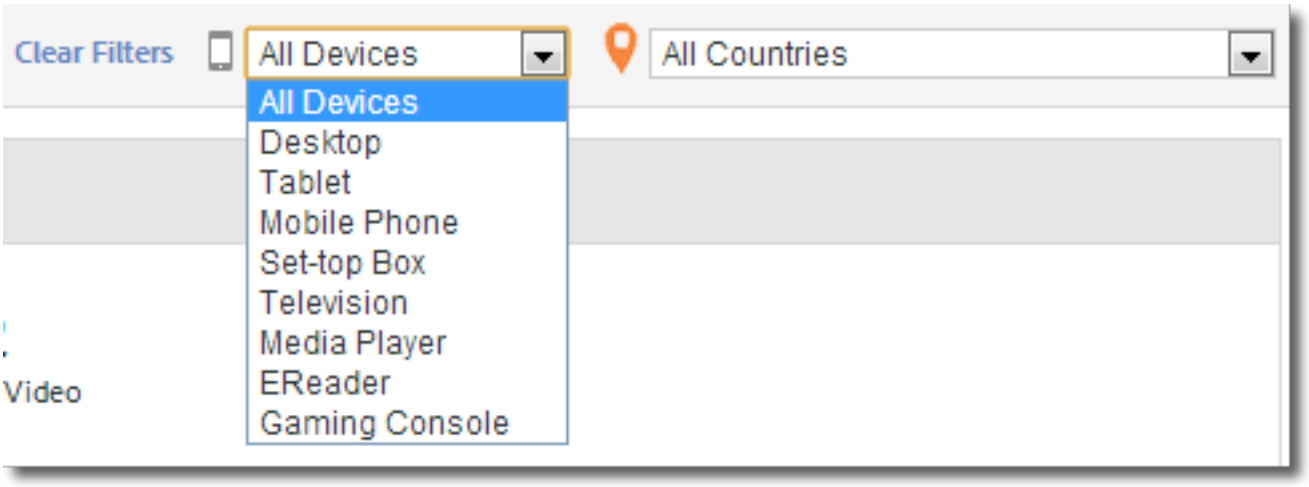
The video overview report is designed to let you monitor video across your site.

The Video Overview Report displays several aggregate measurements to quickly monitor that video is performing as expected. A graph displays video starts next to ad impressions to let you quickly view these metrics for each video.



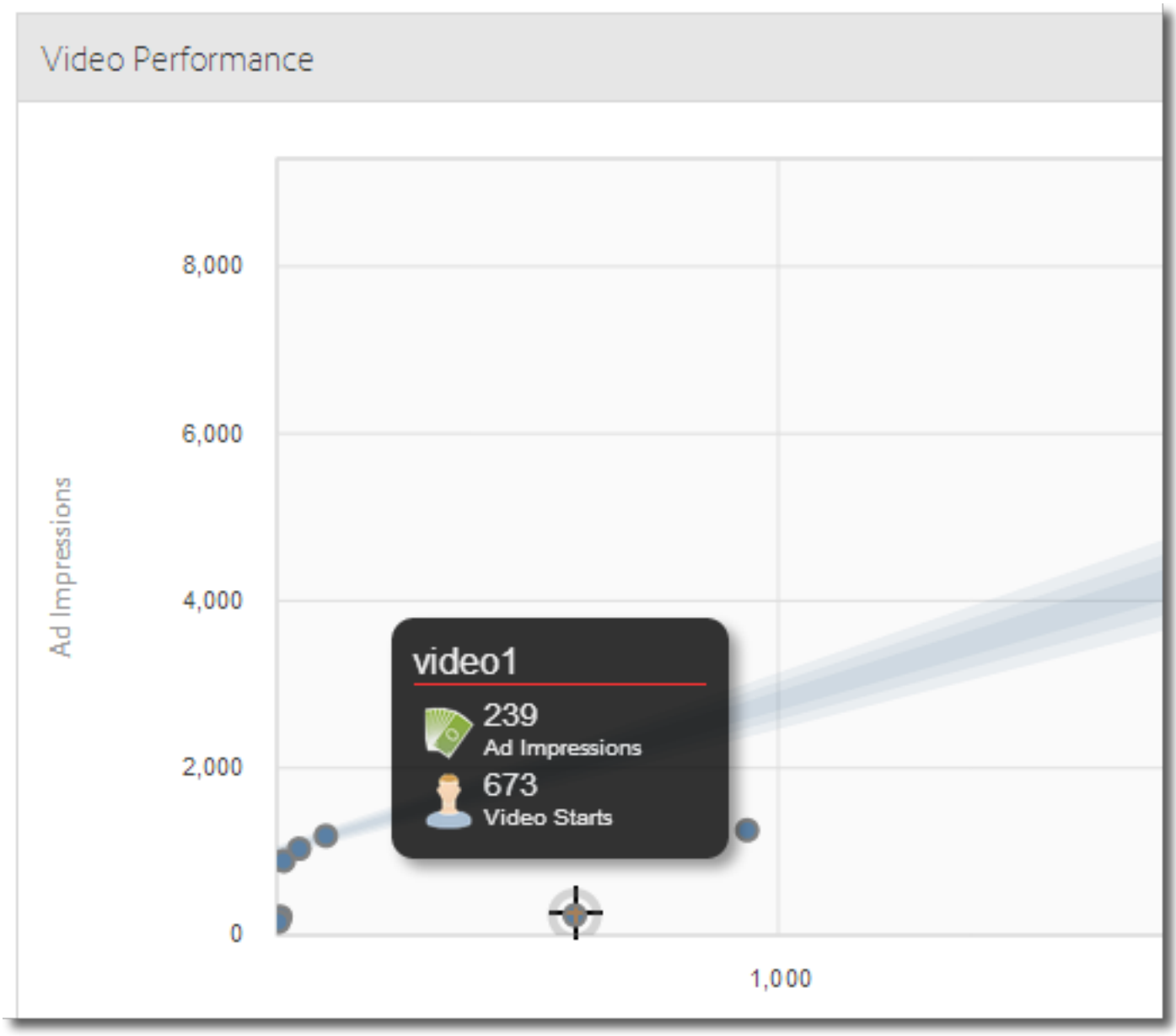
Quick Filters

Quickly display video metrics by device or geo country:



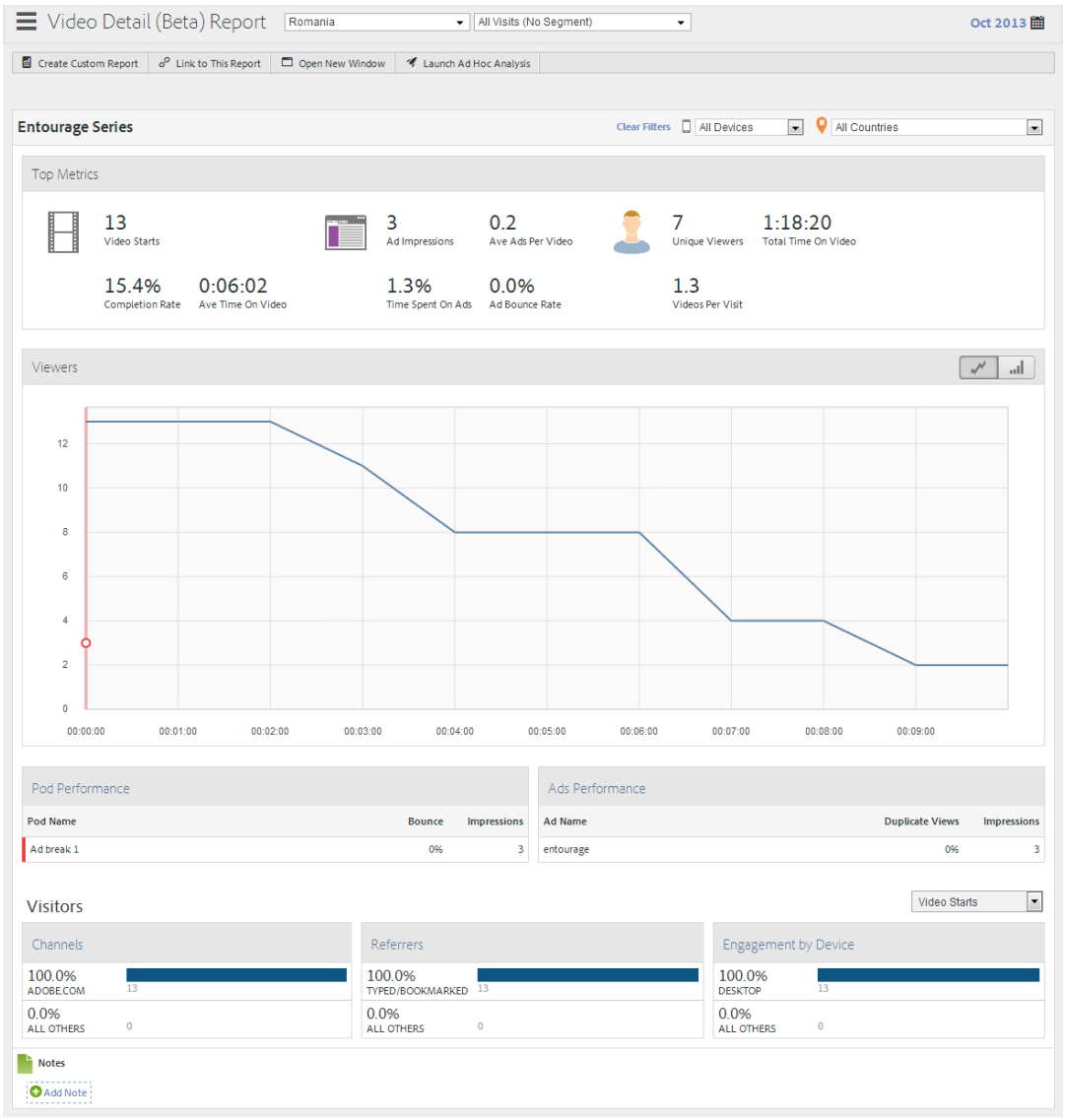
Video Performance

Click-and-drag to zoom in, then hover to view granular metrics for specific videos. Click  to reset the view after you zoom.



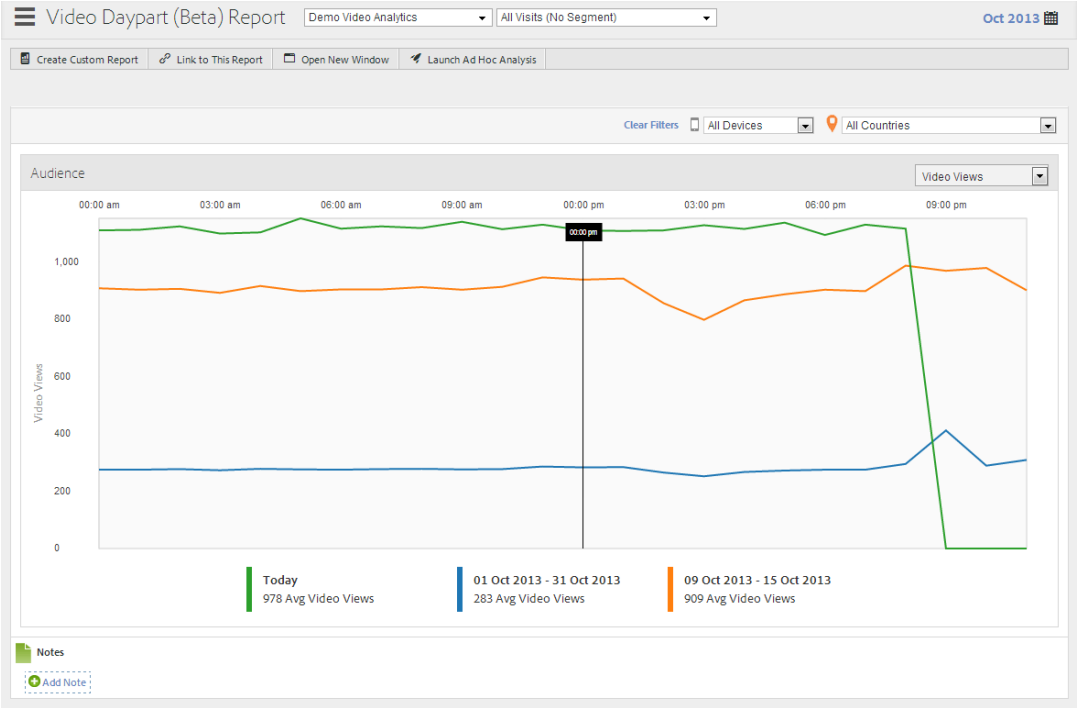
Video Detail Report

The Video Detail Report displays detailed metrics for all videos including starts, completion rate, play percentage, and ad impressions.



Video Daypart

Displays unique visitors and video views by time of day to let you quickly view when your audience is engaged.



Video Configuration

To track video, you designate a set of Custom Conversion Variables (eVars) and Custom Events for use in tracking and reporting on video.

One Custom Insight variable (s.prop) is also used for pathing.

The variables you select for each metric are added to the video configuration page. This lets the system automatically generate and format the standard video reports. The video name evar and the video views counter are both required. Other variables are optional but recommended for complete measurement. After video tracking is enabled, you can view reports generated from video data you have reported using video tracking.

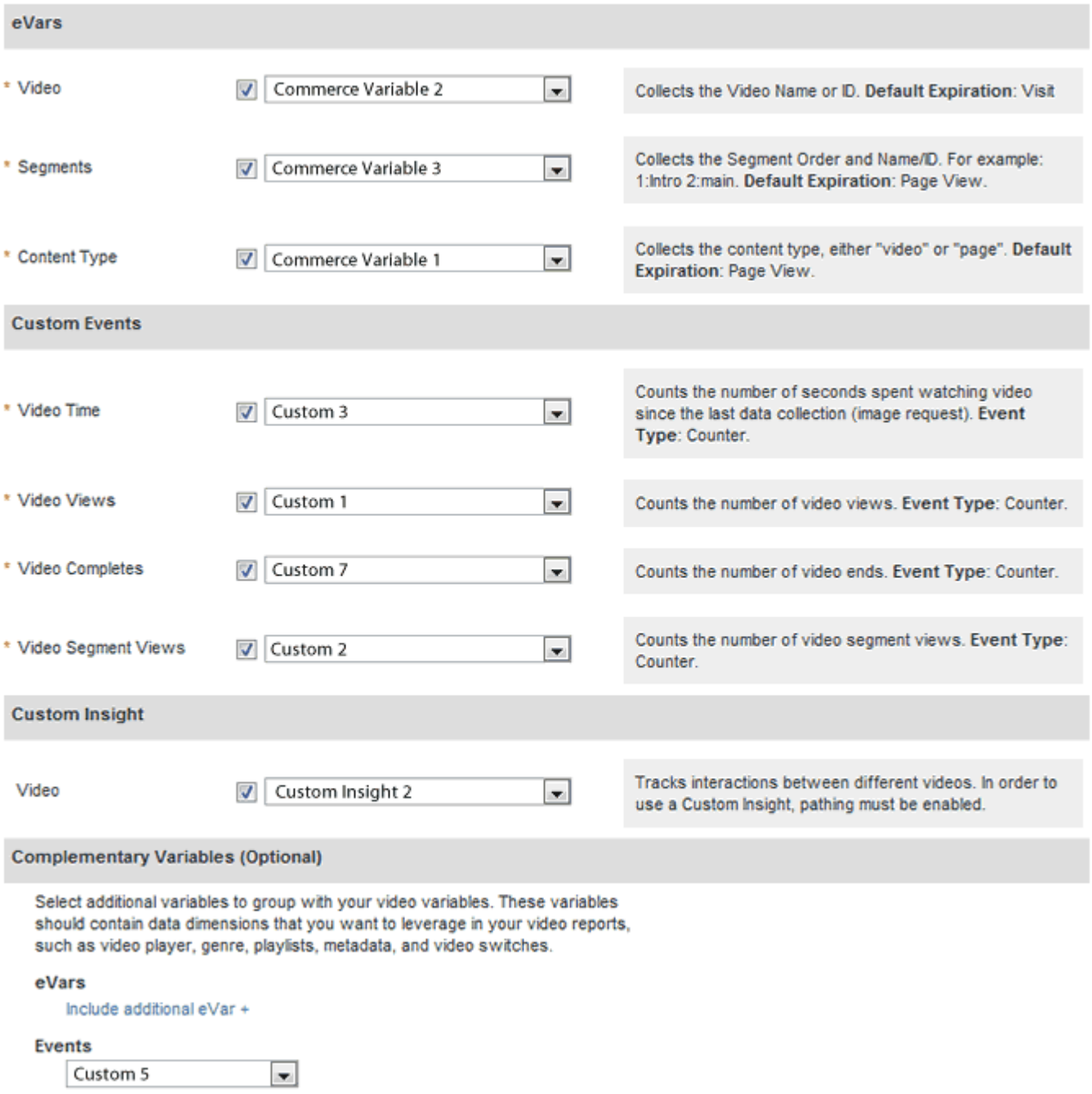
The required variables are described in detail in *Video Variable Reference*.

You can also track any number of additional metrics for video. For example, if you use multiple video players on your site, you might populate an evar with the player name. Some of the variables you select might also be used in other areas of your site. For example, if used across your site, the content type variable can let you measure what percentage of your page views are coming from video and let you relate conversion events to video.

Important: *You must log on to Version 14 of Reports & Analytics to configure these settings.*

1. Visit *Reports & Analytics*, select **Version 14** from the drop-down, and log on.
2. Click **Admin** > **Admin Tools** > **Report Suites**.
3. Select a report suite.
4. Click the **Edit Settings** drop-down list, click **Video Management** > then click **Video Reporting**.
5. A page displays that contains a training video and information to help you configure video tracking. Click **Continue**.

6. Provide the variables and events you selected to track video. Additional milestone events can be added to the **Complementary Variables** section. Any variables and events added to this section are categorized as video variables and are displayed under the Video reports menu.



Video Variable Reference

The following table contains additional details on the commerce variables and custom events for video.

Video Metric	Variable Type	Variable Type
Video Name	eVar Default expiration: Visit	(Required) Collects the name of the video, as specified in the implementation, when a visitor views the video in some way. Marketing reports use the Video eVar to generate the data displayed in video detail reports. The Video variable must be a fully sub-related eVar. If you do not have a fully-subrelated eVar to use for the Video variable, contact Customer Care to have one configured. In version 15 all eVars are fully sub-related by default. Marketing reports allow you to classify on this variable.
Video Name (s.prop for video pathing)	Custom Insight (s.prop)	(Optional) Provides video pathing information. Pathing must be enabled for this variable by Customer Care. Event type: Custom Insight (s.prop)
Segments	eVar Default expiration: Page view	(Required) Collects video segment data, including the segment name and the order in which the segment occurs in the video. This variable is populated by enabling the `segmentByMilestones` variable when tracking player events automatically, or by setting a custom segment name when tracking player events manually. For example, when a visitor views the first segment in a video, Analytics might collect the following in the Segments eVar: `1:M:0-25` The default video data collection method collects data at the following points: video start (play), segment begin, and video end (stop). The system counts the first segment view at the start of the segment, when the visitor starts watching. Subsequent segment views as the segment begins.
Content Type	eVar Default expiration: Page view	Collects data about the type of content viewed by a visitor. Hits sent by video measurement are assigned a content type of "video". This variable does not need to be reserved exclusively for video tracking. Having other content report content type using this same variable lets you analyze the distribution of visitors across the different types of content. For example, you could tag other content types using values such as “article” or “product page” using this variable.

Video Metric	Variable Type	Variable Type
		From a video measurement perspective, Content Type lets you identify video visitors and thereby calculate video conversion rates.
Video Time Played	Event Type: Counter	Counts the time, in seconds, spent watching a video since the last data collection process (image request).
Video Views	Event Type: Counter	Indicates that a visitor has viewed some portion of a video. However, it does not provide any information about how much, or what part, of a video the visitor viewed.
Video Completes	Event Type: Counter	Indicates that a user has viewed a complete video. By default, the complete event is measured 1 second before the end of the video. During implementation, you can specify how many seconds from the end of the video you would like to consider a view complete. For live video and other streams that don't have a defined end, you can specify a custom point to measure completes. For example, after a specific time viewed.
Video Segment Views	Event Type: Counter	Indicates that a visitor has viewed some portion of a video segment. However, it does not provide any information about how much, or what part, of a video segment the visitor viewed.

Video Implementation Worksheet

This worksheet lists the information you need to provide to your video developer to measure video.

Information Required	Description
Seconds and Milestones	Key Milestones to track (as a % of video length): _____ Number of seconds between measurement calls (increments of 5): _____
Video Segments	Divide each video into segments based on: _____ Examples: Quarters or halves, pre-roll, main video, post-roll, advertisement breaks.
Video Completes	Count a view as complete _____ seconds from the end of the video (default is 1 second). For live events and other streams, count a view after _____ seconds watched, or by using the following custom calculation:

Information Required	Description
Video measurement conversion variables and events	• Video Name (eVar): _____ • Video Name (Prop): _____ • Segments (eVar): _____ • Content Type (eVar): _____ • Video Time (Event): _____ • Video Views (Event): _____ • Video Completes (Event): _____ • Video Segment Views (Event): _____
AppMeasurement libraries	If your developer does not have access to Code Manager, he or she can review the *Measuring Video for Developers* section and let you know which libraries to download.

Measuring Video for Developers

This topic provides an overview of the video measurement implementation process for a developer.

The process you follow to measure video is based on the video player you are using. Before you begin development, review *How Video Measurement Works*.

Step	Task	Details
Step 1	Receive the Video Implementation Worksheet from the web analytics team.	*Video Implementation Worksheet* on page 18
Step 2	Complete the tasks outlined in the section for your player. The general process to measure video is as follows: • Download and link the AppMeasurement library that contains the media module. • Map the variables selected by your web analytics team. • Configure milestones. • Track the events that occur in your player. Follow the link in the Details column for specific instructions for your player.	• *Flash Video Playback* on page 20 • *Open Source Media Framework (OSMF)* on page 27 • *Silverlight* on page 37 • *Other Video Players* on page 45
Step 3	Measure additional metrics and review the variable and method reference.	• *Measuring Additional Metrics using Media.monitor* on page 48 • *Media Module Variables* on page 53 • *Media Module Methods* on page 60

Flash Video Playback

Flash Video (FLV) playback is a simple way to display videos on your web site.

To display Flash video, FLV files are added to a Flash Professional project that can be viewed using the Flash Player.

This section contains instructions to measure video that are displayed in the Flash Player.

Download AppMeasurement for Flash

The media module is part of the standard AppMeasurement Libraries for Flash.

The AppMeasurement libraries for Flash are available in Code Manager.

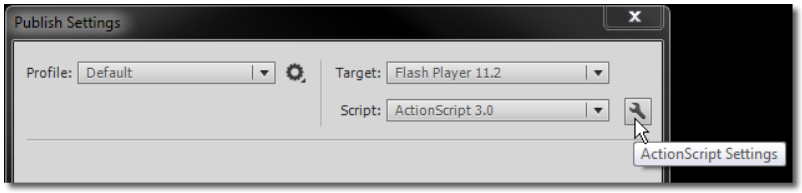
1. In *Marketing Reports & Analytics*, click **Admin Tools** > **Code Manager**.
2. Click **Flash, Flex, & AIR** to download the Flash SDK.
3. Extract the zip you downloaded and copy `AppMeasurement.swc` to a location that is accessible to your Flash project.

Next step: *Add the AppMeasurement for Flash Library to a Project*.

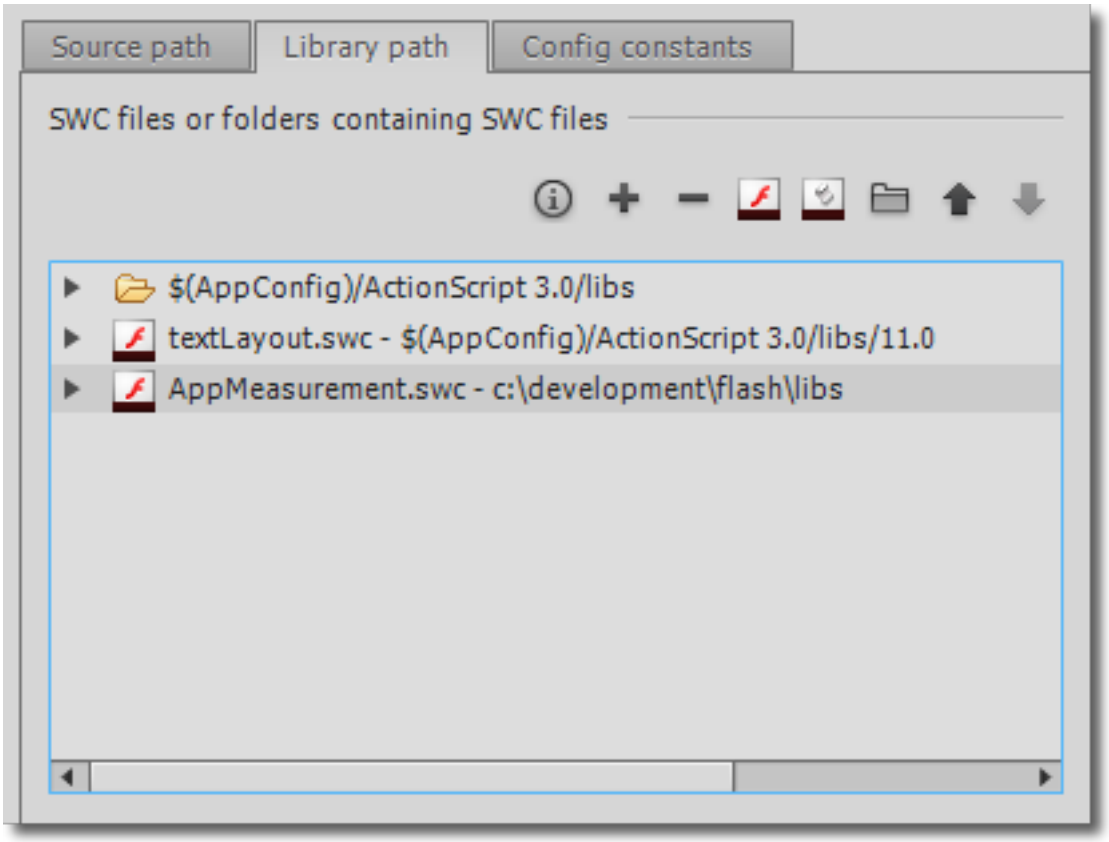
Add the AppMeasurement for Flash Library to a Project

The Flash media module provides the interface to track video.

1. Launch Flash Professional and open the Flash project where you want to include a Flash video.
2. Click **File** > **Publish Settings**, and then open **ActionScript Settings**.



3. Add the `AppMeasurement.swc` library to your project.

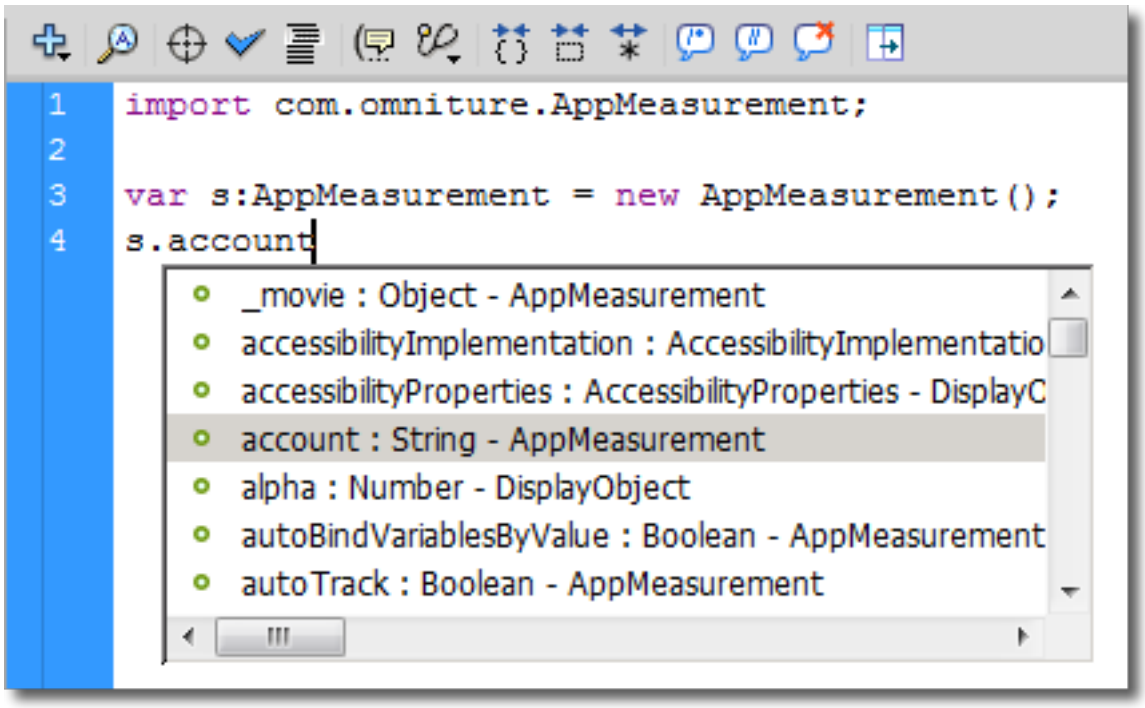


4. In the **Timeline** pane, select a frame that is available to the entire Flash application and open **Actions**.
5. To test that the library was added successfully, add the following ActionScript code to the Actions:

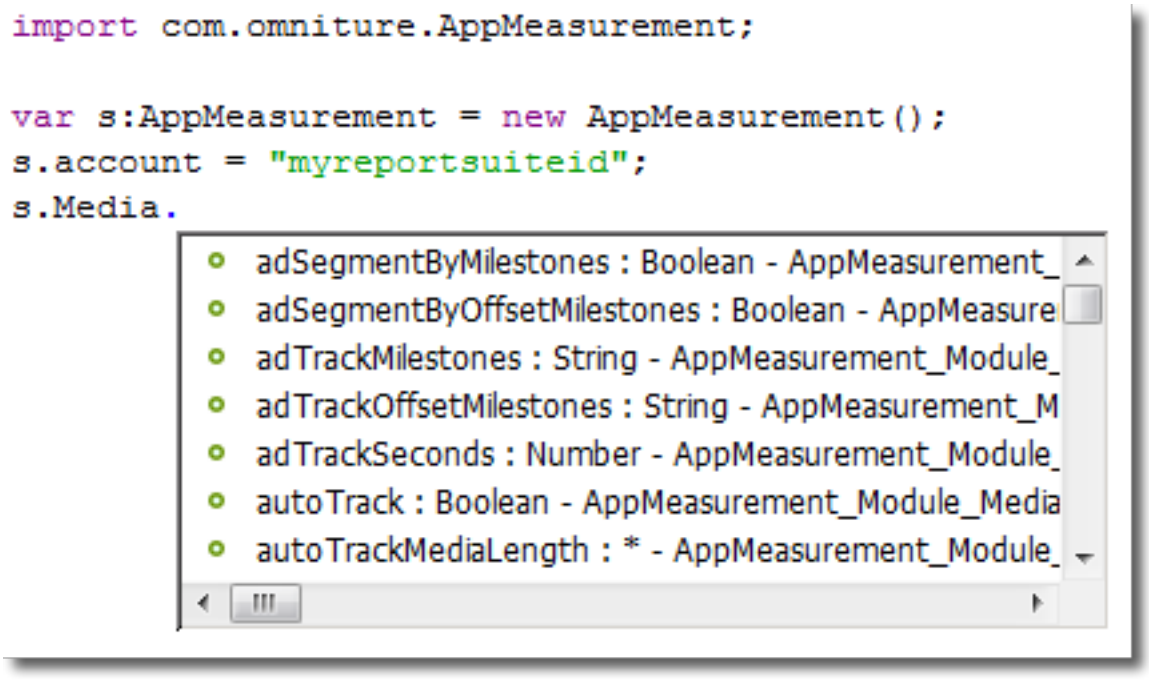
```
import com.omniture.AppMeasurement;

var s:AppMeasurement = new AppMeasurement();
s.account
```

As you type "`s.account`", auto complete should appear. This indicates that the library is being found successfully.



Take a quick look at the methods and variables in the `Media` namespace, since that is where you'll spend most of your time as you configure video:



Next step: *Configure AppMeasurement*.

Configure AppMeasurement

Flash video measurement uses ActionScript, and is configured identically to the JavaScript implementation on your website. The standard *Analytics Variables* are all available in Flash.

1. Define an AppMeasurement object and add it as a child:

```
import com.omniture.AppMeasurement;
var s:AppMeasurement = new AppMeasurement();
addChild(s);
```

2. Populate the following required configuration variables:

   - s.account
   - s.trackingServer

   These values can be copied directly from your s_code.js file.

Next step: *Map Analytics Variables and Events*.

Map Analytics Variables and Events

After you insert the code in your project, you need to map the conversion variables and events you are using to track video.

If the Web Analytics team filled out the *Video Implementation Worksheet*, you might have a list similar to the following:

- 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`

Complete the following to map conversion variables and events:

1. Add the `s.Media.trackUsingContextData` variable and set it to true:

```
s.Media.trackUsingContextData = true;
```

2. Add the `s.Media.contextDataMapping` variable and map the parameters it contains with the SiteCatalyst variables and events you selected:

```
s.Media.trackUsingContextData = true
s.Media.contextDataMapping = {
  "a.media.name":"eVar2,prop2",
  "a.media.segment":"eVar3",
  "a.contentType":"eVar1",
  "a.media.timePlayed":"event3",
  "a.media.view":"event1",
  "a.media.segmentView":"event2",
  "a.media.complete":"event7"
};
```

Next step: *Configure Milestones*

Configure Milestones

Video milestones determine specific points in the video that you want to track.

If the web analytics team filled out the *Video Implementation Worksheet*, you might have a list similar to the following:

Number of seconds between measurement calls (increments of 5): `30`

Milestones to track (as a % of video length): `25%, 50%, 75%`

Divide each video into segments based on: _____

Review *Video Metrics* then complete the following to define the collection interval:

1. (Optional) To track using seconds, add the following:
```
s.Media.trackSeconds = 30;
```
2. (Optional) To track using milestones, add additional milestone events to the milestones parameter of your contextDataMapping variable:
```
s.Media.trackUsingContextData = true
s.Media.contextDataMapping = {
  "a.media.name":"eVar2,prop2",
  "a.media.segment":"eVar3",
  "a.contentType":"eVar1",
  "a.media.timePlayed":"event3",
  "a.media.view":"event1",
  "a.media.segmentView":"event2",
  "a.media.complete":"event7",
  "a.media.milestones":{
     25: "event4",
     50: "event5",
     75:"event6"
     }
};
s.Media.trackMilestones = "25,50,75";
```
   This example measures an event for the 25%, 50%, and 75% milestones. Each milestone you track must also be specified in `trackMilestones`.

   You can also track using offset milestones instead:
```
"a.media.milestones":{
     30: "event4", // 30 seconds from start of video
     60: "event5", // 60 seconds from start of video
     120:"event6"  // 120 seconds from start of video
     }
};
s.Media.trackOffsetMilestones = "30,60,120";
```
   This example measures an event 30, 60, and 120 seconds from the start of the video. Each offset milestone you track must also be specified in `trackOffsetMilestones`.
3. You can use `segmentByMilestones` to have the media module create segments automatically based on your milestones.
```
SegmentByMilestones = true;
```
   If you do not enable `segmentByMilestones` to define segments, you must use a manual implementation (not `autoTrack`) and send in the segment details with `Media.play`. If the web analytics team has defined segments to track, you can define milestones that correspond to each segment then enable `segmentByMilestones`.

Next step: *Track Player Events Using AutoTrack*.

Track Player Events Using AutoTrack

AutoTrack automatically tracks video player events such as start, stop, and pause.

This prevents you from needing to manually track these events and call the open, play, stop, and close methods directly.

See *What is Autotrack?*.

Add the `s.Media.autoTrack` variable and set it to `true`.

```
s.Media.autoTrack = true;
```

Next step: *Test Your Video Measurement Code*.

Test Your Video Measurement Code

A simple way to test your Flash video player is to enable debugging and then view the debug output in Flash Professional.

1. Verify that AppMeasurement `debugTracking` is enabled in your ActionScript:

   ```
   /* Turn on and configure debugging here */
   s.debugTracking = true;
   ```

2. Press **F2** to display the **Output** pane.
3. Click **File** > **Publish Preview** > **Flash.**
4. Each server call appears in the **AppMeasurement Debug** output similar to the following:

   ```
   AppMeasurement Debug: http://democorp.112.2o7.net/b/ss/democorpscdoc/
   0/FAS-3.4.3-AS3/s03319123252294?AQB=1&ndh=1&t=28/6/2011%2013%3A33%3A2%204%20360
   &ce=UTF-8&pageName=video%20test&g=file%3A///F%7C/sandbox/Video/video%2520test.swf
   &cc=USD&events=event1%2C%2C&v1=video&c2=Clip14-tahoe.f4v&v2=Clip14-tahoe.f4v
   &pe=m_s&pev3=video&s=1680x1050&AQE=1
    http://democorp.112.2o7.net/b/ss/democorpscdoc/0/FAS-3.4.3-AS3/s03319123252294?AQB=1
    ndh=1
    t=28/6/2011 13:33:2 4 360
    ce=UTF-8
    pageName=video test
    g=file:///F|/sandbox/Video/video%20test.swf
    cc=USD
    events=event1,,
    v1=video
    c2=Clip14.f4v
    v2=Clip14.f4v
    pe=m_s
    pev3=video
    s=1680x1050
    AQE=1
   ```

   This example server call is from a video start. In the previous example, event1 is selected to track video views. If you are tracking milestones, play through your video and watch for a hit that contains the event you selected for that milestone.

   ```
   events=event3=8,event6
   ```

   In the previous example, `event3` is selected to track time viewed, and `event6` is selected for the milestone.

If you can't get your project to work, you can create a simple video test project:

1. Create a new **ActionScript 3.0** project and then *Add the AppMeasurement for Flash Library*.
2. Drag and drop a local video file on your project. Click **Next**, **Next**, then **Finish** to add the video using the default options.
3. Press F9 to open **Actions**, and then paste the *ActionScript Sample Code* into the actions window.
4. Click **Debug** > **Debug Movie** > **Debug** to run the project. AppMeasurement debug call appear in the **Output** window (press **F2** if you can't see it). Note that this data does not go to any report suite.

ActionScript Sample Code

This section contains a sample implementation in ActionScript.

Important: *Change* `s.account`, `s.trackingServer`, *and the variables in* `s.Media.contextDataMapping` *before you use this code.*

```
/* Import line for ActionScript 3 */
import com.omniture.AppMeasurement;

var s:AppMeasurement = new AppMeasurement();
addChild(s);

/* Specify the Report Suite ID(s) to track here */
s.account = "myrsid"; // CHANGE THIS!

/* Turn on and configure debugging here - turn this off for production deployment */
s.debugTracking = true;
s.trackLocal = true;

/* You may add or alter any code config here */
s.pageName = "";
s.pageURL = "";
s.charSet = "UTF-8";
s.currencyCode = "USD";

/* Turn on and configure ClickMap tracking here */
s.trackClickMap = false;

/* WARNING: Changing any of the below variables will cause drastic changes
to how your visitor data is collected.  Changes should only be made
when instructed to do so by your account manager.*/

/* These values can be copied from your s_code.js file. Your trackign server varies based on
1st or 3rd party cookies, and if you are using SSL. If using first party cookies,
trackingServer will be on your domain, for example metrics.mysite.com */

s.visitorNamespace = "yourNamespace";
s.trackingServer="mycompany.112.2o7.net"; // CHANGE THIS!
s.trackingServerSecure=""; //might not be needed

/* Configure Media Module Functions */

// events, and every variable you track in video, including Media.monitor, should be in this
list.
s.Media.trackVars="events,eVar45,eVar46,eVar47"; // CHANGE THESE!

// every event you track in video, including Media.monitor, should be in this list
s.Media.trackEvents="event45,event46,event47,event48"; // CHANGE THESE!
s.Media.autoTrack=true;
s.Media.trackMilestones="25,50,75";
s.Media.playerName="Video Test 1";
s.Media.trackUsingContextData=true;
s.Media.segmentByMilestones=true;
s.Media.contextDataMapping= { // CHANGE THESE!
  "a.media.name":"eVar45",
  "a.media.segment":"eVar46",
  "a.contentType":"eVar47",
  "a.media.timePlayed":"event45",
  "a.media.view":"event46",
  "a.media.segmentView":"event48",
  "a.media.complete":"event49",
  "a.media.milestones":{
    25:"event50",
    50:"event51",
    75:"event52"
```

```
}
};

s.Media.monitor = function (s,media){

  if(media.event=="OPEN") {
    trace("Media Open: " + media.name);

  /* the following code is an example of tracking an eVar on media open
  you can do something similar for any Media.monitor event
  don't forget to add any eVars and events used in Media.monitor to
  Media.trackVars and Media.trackEvents */

  // s.eVar48=s.Media.playerName;
  // s.Media.track(media.name);
  }
  if (media.event == "MILESTONE") {
    trace("Media Milestone Reached: " + media.milestone );
  }
  if(media.event=="CLOSE") {
    trace("Media Close: " + media.name);
  }
}
```

Open Source Media Framework (OSMF)

For most OSMF players we recommend using the AppMeasurement plug-in for OSMF.

This plug-in provides an instance of AppMeasurement auto-track support for OSMF players, and adds a reference plug-in to OSMF players that captures media data and events.

First, *Download the Media Module for OSMF*. You can add video measurement for OSMF using one of the following methods:

Implementation Method	Description	Guide
Dynamic Plug-in	The dynamic implementation links the plug-in into your Flash project at run-time. A dynamic implementation lets you manage the OSMF player separately from AppMeasurement tracking code using an XML configuration file.	*Dynamic Implementation*
Custom Dynamic Plug-in	This lets you extend the dynamic plug-in if you need to send additional metrics using Media.Monitor.	*Custom Dynamic Implementation*
Static	The static implementation links AppMeasurement_Media_AutoTrackOSMF.swc into your Flash project. You can then configure video measurement using the library directly.	*Static Implementation*

Download the Media Module for OSMF

You can access the AppMeasurement plug-in for OSMF from the SiteCatalyst Admin Console.

1. In the Adobe Marketing Cloud, click **Admin** > **Admin Console** > **Code Manager**.
2. In the Select the type of code to generate field, select **ActionScript (Flash/Flex)**, provide the requested information, and then click **Generate Code**.
3. Select the **Component Files** Tab, then save the necessary AppMeasurement component files to your local system:

Option	Description
Dynamic Implementation	• AppMeasurementExtensionOSMF.swf
Custom Dynamic Implementation	• AppMeasurementExtensionOSMF.swc
Static Implementation	• AppMeasurement_Media_AutoTrackOSMF.swc • AppMeasurement.swc

Dynamic Implementation

A dynamic implementation links the plug-in into your Flash project at run-time.

A dynamic implementation lets you manage the OSMF player separately from AppMeasurement tracking code, including the AppMeasurement plug-in for OSMF.

To do this, load `AppMeasurementExtensionOSMF.swf` and specify a configuration URL that points to an XML config file in the URL query-string. The config file contains all the media tracking settings to be used by your OSMF player.

For example, given an OSMF player with a `MediaFactory`, the dynamic implementation code might look like the following:

```
mediaFactory.loadPlugin(new
URLResource("http://www.corp1.com/AppMeasurementExtensionOSMF.swf?s.configURL=http%3A%2F%2Fwww.corp1.com%2Fasconfig.xml"));
```

This code tells MediaFactory to load the SWF. The SWF then loads the config file, which populates the AppMeasurement instance inside the SWF. This effectively performs the same integration process described in the static implementation (adding the AppMeasurement OSMF bridge and creating the reference plug-in). The `s.configURL` parameter name in the query string is case-sensitive.

See the following section for details on the XML configuration file.

XML Configuration File

When using a dynamic OSMF implementation, you can use an XML config file to bind variables to OSMF metadata. AppMeasurement uses the following variable binding syntax:

```
<variable>{media.player.metadata(namespace,key)}</variable>
```

variable: The name of the variable you wish to set (for example, eVar6).

namespace: (Optional) The OSMF metadata namespace you want to use. If you do not specify a namespace, the AppMeasurement OSMF plug-in uses the first matching key it locates in any namespace. When looking for keys, the plug-in looks first at MediaElement metadata, then at MediaResource metadata.

key: The specific metadata value you want to use.

The following section contains a sample XML configuration file.

• The `trackSeconds` and `milestone` sections are optional. See *Video Metrics*.

If the Web Analytics team filled out the *Video Implementation Worksheet*, you might have a list similar to the following:

• 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`

Map these variables to the appropriate `a.media` variable in the `contextDataMapping` section.

```
<config>
<account>myrsid</account>
<debugTracking>true</debugTracking>
<visitorNamespace>corp1</visitorNamespace>
<trackingServer>corp1.d1.sc.omtrdc.net</trackingServer>
<Media>
    <autoTrack>true</autoTrack>
    <trackMilestones>25,50,75</trackMilestones>
    <trackVars>events,eVar1,eVar2,eVar3,prop2</trackVars>
    <trackEvents>event1,event2,event3,event4,event5,event6,event7</trackEvents>
    <segmentByMilestones>true</segmentByMilestones>
    <trackUsingContextData>true</trackUsingContextData>
    <contextDataMapping>
            <a.media.name>eVar2,prop2</a.media.name>
            <a.media.segment>eVar3</a.media.segment>
            <a.contentType>eVar1</a.contentType>
            <a.media.timePlayed>event3</a.media.timePlayed>
            <a.media.view>event1</a.media.view>
            <a.media.segmentView>event2</a.media.segmentView>
            <a.media.complete>event7</a.media.complete>
            <a.media.milestones>
                <item name="25">event4</item>
                <item name="50">event5</item>
                <item name="75">event6</item>
            </a.media.milestones>
    </contextDataMapping>
</Media>
</config>
```

Troubleshooting a Dynamic OSMF Implementation

If you are unable to capture data with your dynamic implementation, consider the following troubleshooting tips:

- When using a dynamic implementation, you cannot load the plug-in from a local drive. You must load it from a Web server to avoid Flash security errors.
- When testing the dynamic implementation, listen for the pluginLoadError event in MediaFactory. This event captures any load errors related to the plug-in.

Be aware of the following issues when using the configURL parameter:

- XML tags are case sensitive.
- Do not enclose strings in quotes.
- Booleans can be set to true or false.
- Use entities for special characters. For example: < for "<", > for ">", and & for "&".
- Before using configURL, you might need to configure Flash security to suport using a configuration file from an external domain. For more information about Flash security, search for crossdomain at *http://www.adobe.com/support/flashplayer/*.

Custom Dynamic Implementation

If you use a dynamic plug-in implementation to track an OSMF player, and also need to use a custom `Media.monitor` method, you can build your own custom dynamic plug-in by subclassing the prebuilt dynamic plug-in.

Along with the prebuilt `AppMeasurementExtensionOSMF.swf`, you can also download `AppMeasurementExtensionOSMF.swc`, which is a Flash library that provides an AppMeasurementExtension class for subclassing to build a custom dynamic plug-in.

The custom dynamic plug-in should do the following. Once compiled, you can use it like the pre-built dynamic plug-in.

1. Import com.omniture.AppMeasurementExtension.
2. Define a class that will be the SWF that extends AppMeasurementExtension.
3. Override the public function customizeExtension():void method.
4. Inside of your customizeExtension method, update the “s” member as needed. The “s” member is the instance of AppMeasurement inside of the AppMeasurementExtension.
5. Compile your custom SWF linking against AppMeasurementExtensionOSMF.swc.

For example, the following ActionScript code for a custom dynamic plug-in does the following:

1. Overrides the video name tracked with the episodeID from the OSMF metadata.
2. In the Media.monitor method, sets eVar1 to the “series” key, eVar2 to the “season” key, and eVar3 to the “episode” key in the http://www.corp1.com/ namespace.
3. In the Media.monitor method, sends in custom events “event1” as video starts and “event2” as video stops.

```
package {
    import com.omniture.AppMeasurementExtension;
    public class CustomPlugin extends AppMeasurementExtension {
        public function customizeExtension():void {
            super();
            s.Media.autoTrackMediaName = "{media.player.metadata(http://www.corp1.com/
                ,episodeID)"
            s.Media.monitor = function (s:Object,media:Object) {
                s.trackVars = "events,eVar1,eVar2,eVar3";
                s.trackEvents = "event1,event2";
                s.eVar1 = media.player.metadata("http://www.corp1.com/","series");
                s.eVar2 = media.player.metadata("http://www.corp1.com/","season");
                s.eVar3 = media.player.metadata("http://www.corp1.com/","episode");
                if (media.mediaEvent == "OPEN") {
                    s.events = "event1";
                    s.Media.track(media.name);
                } else if (media.mediaEvent == "CLOSE") {
                    s.events = "event2";
                    s.Media.track(media.name);
                }
            }
        }
    }
}
```

OSMF Metadata

You can access OSMF metadata within your video player.

To use OSMF metadata inside of your Media.monitor method, call media.player.metadata(namespace,key), where:

namespace: (Optional) The OSMF metadata namespace you want to use. If you do not specify a namespace, the AppMeasurement OSMF plug-in uses the first matching key it locates in any namespace. When looking for keys, the plug-in looks first at MediaElement metadata, then at MediaResource metadata.

key: The specific metadata value you want to use.

For example, the following custom Media.monitor method sets eVar1 to the “series” key, eVar2 to the “season” key, and eVar3 to the “episode” key in the http://www.corp1.com/ namespace:

```
s.Media.monitor = function (s:Object,media:Object) {
    s.trackVars = "events,eVar1,eVar2,eVar3";
    s.trackEvents = "event1,event2";
    s.eVar1 = media.player.metadata("http://www.corp1.com/","series");
    s.eVar2 = media.player.metadata("http://www.corp1.com/","season");
    s.eVar3 = media.player.metadata("http://www.corp1.com/","episode");
    if (media.mediaEvent == "OPEN") {
        s.events = "event1";
```

```
        s.Media.track(media.name);
    } else if (media.mediaEvent == "CLOSE") {
        s.events = "event2";
        s.Media.track(media.name);
    }
}
```

Static Implementation

The static implementation links AppMeasurement_Media_AutoTrackOSMF.swc into your Flash project before compiling it for use.

AppMeasurement_Media_AutoTrackOSMF.swc includes the bridge class AutoTrackOSMF.

For example, once you have linked both AppMeasurement and the AppMeasurement plug-in for OSMF into the Flash project that includes an OSMF player with a MediaFactory, the tracking code implementation might look like the following:

```
import com.omniture.AppMeasurement; //1
import com.omniture.media.AutoTrackOSMF; //2

var s:AppMeasurement = new AppMeasurement();
s.debugTracking = true;
s.trackLocal = true;
s.account = "jdoe";
s.visitorNamespace = “corp1”;
s.trackingServer = "corp1.d1.sc.omtrdc.net";
s.pageName = "OSMF Player";

s.Media.trackSeconds = 15;
s.Media.addAutoTrackHandler(new AutoTrackOSMF()); //3
s.Media.autoTrack = true;

mediaFactory.loadPlugin(s.Media.AutoTrackOSMF.pluginResource); //4
```

The highlighted code performs the linking of AppMeasurement to the OSMF player, including:

1. Import the AppMeasurement class
2. Import the AppMeasurement plug-in for OSMF class
3. Add the auto-track support for OSMF to an instance of AppMeasurement
4. Loading the reference plug-in into the OSMF MediaFactory.

Map Analytics Variables and Events

After you insert the code in your project, you need to map the conversion variables and events you are using to track video.

If the Web Analytics team filled out the *Video Implementation Worksheet*, you might have a list similar to the following:

- 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`

Complete the following to map conversion variables and events:

1. Add the `s.Media.trackUsingContextData` variable and set it to true:

```
s.Media.trackUsingContextData = true;
```

2. Add the `s.Media.contextDataMapping` variable and map the parameters it contains with the SiteCatalyst variables and events you selected:

```
s.Media.trackUsingContextData = true
s.Media.contextDataMapping = {
  "a.media.name":"eVar2,prop2",
  "a.media.segment":"eVar3",
  "a.contentType":"eVar1",
  "a.media.timePlayed":"event3",
  "a.media.view":"event1",
  "a.media.segmentView":"event2",
  "a.media.complete":"event7"
};
```

Next step: *Configure Milestones*

Configure Milestones

Video milestones determine specific points in the video that you want to track.

If the web analytics team filled out the *Video Implementation Worksheet*, you might have a list similar to the following:

Number of seconds between measurement calls (increments of 5): `30`

Milestones to track (as a % of video length): `25%, 50%, 75%`

Divide each video into segments based on: _____

Review *Video Metrics* then complete the following to define the collection interval:

1. (Optional) To track using seconds, add the following:

```
s.Media.trackSeconds = 30;
```

2. (Optional) To track using milestones, add additional milestone events to the milestones parameter of your contextDataMapping variable:

```
s.Media.trackUsingContextData = true
s.Media.contextDataMapping = {
  "a.media.name":"eVar2,prop2",
  "a.media.segment":"eVar3",
  "a.contentType":"eVar1",
  "a.media.timePlayed":"event3",
  "a.media.view":"event1",
  "a.media.segmentView":"event2",
  "a.media.complete":"event7",
  "a.media.milestones":{
     25: "event4",
     50: "event5",
     75:"event6"
     }
};
s.Media.trackMilestones = "25,50,75";
```

   This example measures an event for the 25%, 50%, and 75% milestones. Each milestone you track must also be specified in `trackMilestones`.

   You can also track using offset milestones instead:

```
"a.media.milestones":{
     30: "event4", // 30 seconds from start of video
     60: "event5", // 60 seconds from start of video
     120:"event6"  // 120 seconds from start of video
     }
};
s.Media.trackOffsetMilestones = "30,60,120";
```

This example measures an event 30, 60, and 120 seconds from the start of the video. Each offset milestone you track must also be specified in `trackOffsetMilestones`.

3. You can use `segmentByMilestones` to have the media module create segments automatically based on your milestones.
```
SegmentByMilestones = true;
```
If you do not enable `segmentByMilestones` to define segments, you must use a manual implementation (not `autoTrack`) and send in the segment details with `Media.play`. If the web analytics team has defined segments to track, you can define milestones that correspond to each segment then enable `segmentByMilestones`.

Next step: *Track Player Events Using AutoTrack*.

Track Player Events Using AutoTrack

AutoTrack automatically tracks video player events such as start, stop, and pause.

This prevents you from needing to manually track these events and call the open, play, stop, and close methods directly.

See *What is Autotrack?*.

Add the `s.Media.autoTrack` variable and set it to `true`.
```
s.Media.autoTrack = true;
```

Next step: *Test Your Video Measurement Code*.

Using OSMF Metadata to Override a Video Name

By default, the AppMeasurement OSMF plug-in uses the URL from the `MediaResource` as the video name.

You can override this with OSMF metadata by using variable binding to bind `Media.autoTrackMediaName` to the OSMF metadata of your choice.

You can do this using either an XML config file, or using ActionScript. Both of the following examples override the video name with the episodeID from the OSMF metadata.

Example config.xml (dynamic implementation)

```
<autoTrackMediaName>{media.player.metadata(http://www.corp1.com/,episodeID)}</autoTrackMediaName>
```

Example ActionScript (static implementation)

```
s.Media.addAutoTrackHandler(new AutoTrackOSMF());
s.Media.autoTrackMediaName = "{media.player.metadata(http://www.corp1.com/,episodeID)}"
s.Media.autoTrack = true;
```

Variable Binding

These examples use curly braces (`{ }`) to populate an XML tag with a variable value. If you need to include curly braces at the beginning and end of a literal tag value, set the `autoBind` attribute to `false` on the tag: `<autoTrackMediaName autoBind=false>{123}</autoTrackMediaName>`

Apple iOS

Note: *These instructions are for version 2.x of AppMeasurement for iOS, which is available in Code Manager in Admin Console and documented here. Version 3.x of the AppMeasurement Libraries for iOS are not covered in this guide, they are documented here.*

Download the Media Module for iOS

The Media Module is part of the standard AppMeasurement libraries for iOS.

1. In the Adobe Marketing Cloud, click **Admin** > **Admin Console** > **Code Manager**.
2. In the Select the type of code to generate field, select **iOS (iPhone and iPad)**, provide the requested information, and then click **Generate Code**.
3. In the **iPhone Config Text** section, save the `s.account` and `s.trackingServer` variable and value. These variables are required for implementation.
4. Select the **Component Files** Tab, then save the AppMeasurement_iOS.zip to your local system.

Add the iOS Media Module to a Project

1. Launch the Xcode IDE.
2. In the **Groups & Files** panel, right-click on the project and then click **Add Files To "Project Name"**.

   Select the following options:

   - Select **copy items into destination group's folder**.
   - Select **create groups** for any added folders.
   - Set Reference Type to **Default.**
   - Set Text Encoding to **Unicode (UTF-8)**.
   - Select **Recursively create groups for any added folders**.
   - In the **Add To Targets** section, make sure your project is selected.
3. Browse to `AppMeasurement.h`, then click **Add.**
4. Navigate to the Build Phases tab of your desired target and then Expand the **Link Binary with Libraries** item.
5. Click the + button and click **Add Other**. Browse to the `libAppMeasurement.a` file and click **Open.**

Map Conversion Variables and Events

After you insert the code in your project, you need to map the conversion variables and events you are using to track video.

If the Web Analytics team filled out the *Video Implementation Worksheet*, you might have a list similar to the following:

- 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`

Complete the following to map conversion variables and events:

1. Add the `s.Media.trackUsingContextData` variable and set it to true:

```
s.Media.trackUsingContextData = true;
```

2. Add the `s.Media.contextDataMapping` variable and map the parameters it contains with the SiteCatalyst variables and events you selected:

```
s.Media.trackUsingContextData = true
s.Media.contextDataMapping = {
  "a.media.name":"eVar2,prop2",
  "a.media.segment":"eVar3",
  "a.contentType":"eVar1",
  "a.media.timePlayed":"event3",
  "a.media.view":"event1",
  "a.media.segmentView":"event2",
  "a.media.complete":"event7"
};
```

Configure Milestones

Video milestones determine specific points in the video that you want to track.

If the web analytics team filled out the *Video Implementation Worksheet*, you might have a list similar to the following:

Number of seconds between measurement calls (increments of 5): `30`

Milestones to track (as a % of video length): `25%, 50%, 75%`

Divide each video into segments based on: _____

Review *Video Metrics* then complete the following to define the collection interval:

1. (Optional) To track using seconds, add the following:

```
s.Media.trackSeconds = 30;
```

2. (Optional) To track using milestones, add additional milestone events to the milestones parameter of your contextDataMapping variable:

```
s.Media.trackUsingContextData = true
s.Media.contextDataMapping = {
  "a.media.name":"eVar2,prop2",
  "a.media.segment":"eVar3",
  "a.contentType":"eVar1",
  "a.media.timePlayed":"event3",
  "a.media.view":"event1",
  "a.media.segmentView":"event2",
  "a.media.complete":"event7",
  "a.media.milestones":{
     25: "event4",
     50: "event5",
     75:"event6"
     }
};
s.Media.trackMilestones = "25,50,75";
```

   This example measures an event for the 25%, 50%, and 75% milestones. Each milestone you track must also be specified in `trackMilestones`.

   You can also track using offset milestones instead:

```
"a.media.milestones":{
     30: "event4", // 30 seconds from start of video
```

```
    60: "event5", // 60 seconds from start of video
    120:"event6"  // 120 seconds from start of video
    }
};
s.Media.trackOffsetMilestones = "30,60,120";
```

This example measures an event 30, 60, and 120 seconds from the start of the video. Each offset milestone you track must also be specified in `trackOffsetMilestones`.

3. You can use `segmentByMilestones` to have the media module create segments automatically based on your milestones.

```
SegmentByMilestones = true;
```

If you do not enable `segmentByMilestones` to define segments, you must use a manual implementation (not `autoTrack`) and send in the segment details with `Media.play`. If the web analytics team has defined segments to track, you can define milestones that correspond to each segment then enable `segmentByMilestones`.

Track Player Events Using AutoTrack

AutoTrack automatically tracks video player events such as start, stop, and pause.

This prevents you from needing to manually track these events and call the open, play, stop, and close methods directly.

See *What is Autotrack?*.

Add the `s.Media.autoTrack` variable and set it to `true`.

```
s.Media.autoTrack = true;
```

iOS Sample Code

This section contains a sample implementation for iOS.

```
#import "ADMS_MediaMeasurement.h"
s.Media.trackUsingContextData = YES;
s.Media.contextDataMapping = [NSDictionary dictionaryWithObjectsAndKeys:
          @"eVar2,prop2",@"a.media.name",
          @"eVar3",@"a.media.segment",
          @"eVar1",@"a.contentType",
          @"event3",@"a.media.timePlayed",
          @"event1",@"a.media.view",
          @"event2",@"a.media.segmentView",
          @"event7",@"a.media.complete",
           _milestoneMappingDict, @"a.media.milestones",
          _offsetMilestoneMappingDict, @"a.media.offsetMilestones",
          nil];

NSDictionary *_milestoneMappingDict = [NSDictionary dictionaryWithObjectsAndKeys:
                                          @"event4",@"25",
                                          @"event5", @"50",
                                          @"event6", @"75",
                                          nil];
NSDictionary *_offsetMilestoneMappingDict = [NSDictionary dictionaryWithObjectsAndKeys:
                                           @"event8",@"20",
                                           @"event9", @"40",
                                           @"event10", @"60",
                                           nil];
//Track By Milestones:
s.Media.trackMilestones = @"25,50,75";

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

```
// track by seconds:
s.Media.trackSeconds = 15;

// track Milestones & segmentByMilestones & seconds:
s.Media.trackMilestones = @"25,50,75";
s.Media.segmentByMilestones = YES;
s.Media.trackSeconds = 15;

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

// track offsetMilestones:
s.Media.trackOffsetMilestones = @"5,15,45";
```

Android

Note: *Version 1.x of AppMeasurement for Android, which is available in Code Manager in Admin Console and documented here, does not support video measurement. Version 3.x of the AppMeasurement Libraries for Android supports video measurement and is documented here.*

Silverlight

This section contains instructions to measure video that is displayed using Silverlight.

Download the Media Module for Silverlight

The media module is part of the standard AppMeasurement Libraries for Windows Phone, Silverlight, and .NET.

To add the media module for Silverlight you download the component libraries and include them in your project.

1. In the Adobe Marketing Cloud, click **Admin** > **Admin Console** > **Code Manager**.
2. Select **Windows Phone, Silverlight, and .NET**, provide the requested information, and then click **Generate Code**.
3. From the **C# Example** Tab, copy the sample to a text file and save it. This code can be copied to your Silverlight Project later.
4. From the **Component Files** Tab, download `AppMeasurement_Silverlight.dll` and `AppMeasurement_Silverlight_Debug.dll`.

Add the Silverlight Media Module to a Project

The Media Module defines the interfaces to track video.

The following task uses the Microsoft Web Developer 2008 Express Edition.

1. In **Visual Web Developer**, Click **File** > **New Project**.
2. Select **Silverlight Application**, then click OK.
3. Deselect **Host the Silverlight application in a new Web site**, then click OK.
4. In the **Solution Explorer**, right-click **References**, then select **Add Reference**.
5. Select the **Browse** tab, browse to and select the appropriate AppMeasurement for Silverlight library, then click OK.

Add Media Module Code

1. Add the following line to the Silverlight project to enable AppMeasurement for the project.

```
using com.omniture;
```

2. Create a variable for the AppMeasurement instance in your project. For example:

```
s = new AppMeasurement();
```

3. Create an instance of the AppMeasurement object and add the necessary Marketing Cloud variables to the Silverlight application. For example:

```
using com.omniture;

InitializeComponent();

s = new AppMeasurement();

/* Specify the Report Suite ID(s) to track here */
s.account = "myreportsuiteID";
/* Turn on and configure debugging here */
s.debugTracking = true;
/* You may add or alter any code config here */
s.pageName = "";
s.pageURL = "";
s.charSet = "UTF-8";
s.currencyCode = "USD";
```

Map Conversion Variables and Events

After you insert the code in your project, you need to map the conversion variables and events you are using to track video.

If the Web Analytics team filled out the *Video Implementation Worksheet*, you might have a list similar to the following:

- 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`

Complete the following to map conversion variables and events:

1. Add the `s.Media.trackUsingContextData` variable and set it to true:

```
s.Media.trackUsingContextData = true;
```

2. Add the `s.Media.contextDataMapping` variable and map the parameters it contains with the SiteCatalyst variables and events you selected:

```
s.Media.trackUsingContextData = true
s.Media.contextDataMapping = {
  "a.media.name":"eVar2,prop2",
  "a.media.segment":"eVar3",
  "a.contentType":"eVar1",
  "a.media.timePlayed":"event3",
  "a.media.view":"event1",
  "a.media.segmentView":"event2",
```

```
  "a.media.complete":"event7"
};
```

Configure Milestones

Video milestones determine specific points in the video that you want to track.

If the web analytics team filled out the *Video Implementation Worksheet*, you might have a list similar to the following:

Number of seconds between measurement calls (increments of 5): `30`

Milestones to track (as a % of video length): `25%, 50%, 75%`

Divide each video into segments based on: _____

Review *Video Metrics* then complete the following to define the collection interval:

1. (Optional) To track using seconds, add the following:
   ```
   s.Media.trackSeconds = 30;
   ```
2. (Optional) To track using milestones, add additional milestone events to the milestones parameter of your contextDataMapping variable:
   ```
   s.Media.trackUsingContextData = true
   s.Media.contextDataMapping = {
     "a.media.name":"eVar2,prop2",
     "a.media.segment":"eVar3",
     "a.contentType":"eVar1",
     "a.media.timePlayed":"event3",
     "a.media.view":"event1",
     "a.media.segmentView":"event2",
     "a.media.complete":"event7",
     "a.media.milestones":{
        25: "event4",
        50: "event5",
        75:"event6"
        }
   };
   s.Media.trackMilestones = "25,50,75";
   ```
   This example measures an event for the 25%, 50%, and 75% milestones. Each milestone you track must also be specified in `trackMilestones`.

   You can also track using offset milestones instead:
   ```
   "a.media.milestones":{
        30: "event4", // 30 seconds from start of video
        60: "event5", // 60 seconds from start of video
        120:"event6"  // 120 seconds from start of video
        }
   };
   s.Media.trackOffsetMilestones = "30,60,120";
   ```
   This example measures an event 30, 60, and 120 seconds from the start of the video. Each offset milestone you track must also be specified in `trackOffsetMilestones`.
3. You can use `segmentByMilestones` to have the media module create segments automatically based on your milestones.
   ```
   SegmentByMilestones = true;
   ```
   If you do not enable `segmentByMilestones` to define segments, you must use a manual implementation (not `autoTrack`) and send in the segment details with `Media.play`. If the web analytics team has defined segments to track, you can define milestones that correspond to each segment then enable `segmentByMilestones`.

Track Player Events Using AutoTrack

AutoTrack automatically tracks video player events such as start, stop, and pause.

This prevents you from needing to manually track these events and call the open, play, stop, and close methods directly.

See *What is Autotrack?*.

Add the `s.Media.autoTrack` variable and set it to `true`.

```
s.Media.autoTrack = true;
```

Using the setInterface Method

This method specifies a root layout that you can use to find the root of the Silverlight application.

The `setInterface` method is available only in the Silverlight media module. Finding the root is necessary for certain advanced features such as autoTrack.

The following example shows how to set the root layout object using `setInterface`:

```
s.setInterface(interface);
```

Silverlight Sample Code

This section contains a sample implementation for Silverlight.

```
using com.omniture;

namespace AppMeasurementExample {
  public partial class Page : UserControl {
    AppMeasurement s;

    public Page() {
      InitializeComponent();

      s = new AppMeasurement();

      /* Specify the Report Suite ID(s) to track here */
      s.account = "myreportsuiteID";
      /* Turn on and configure debugging here */
      s.debugTracking = true;
      /* You may add or alter any code config here */
      s.pageName = "";
      s.pageURL = "";
      s.charSet = "UTF-8";
      s.currencyCode = "USD";
      s.Media.trackMilestones="25,50,75";
      s.Media.playerName="My Media Player";
      s.Media.trackUsingContextData = true
      s.Media.segmentByMilestones = true;
      s.Media.contextDataMapping = new Dictionary<string,object> {
       "a.media.name":"eVar2,prop2",
       "a.media.segment":"eVar3",
       "a.contentType":"eVar1",
       "a.media.timePlayed":"event3",
       "a.media.view":"event1",
       "a.media.segmentView":"event2",
       "a.media.complete":"event7",
       "a.media.milestones":{
          25:"event4",
          50:"event5",
          75:"event6"
        }
```

```
      };
      s.Media.autoTrack = true;
      s.Media.attach(mediaElement);

      ... //other page code
      }
    }
}
```

Using JavaScript to Track a Video Player

JavaScript can be used to track a wide variety of players. To track using JavaScript, you add code to the web page that contains your player and track the player using event handlers.

This section contains instructions to measure video using JavaScript.

Download the Media Module for JavaScript

The media module for JavaScript is included in the AppMeasurement for JavaScript download.

If you are using AppMeasurement for JavaScript 1.x, paste the contents of `AppMeasurement_Module_Media.js` just above the `============== DO NOT ALTER ANYTHING BELOW THIS LINE ! ==============` comment.

If you are using H Code, find the media module and remove the surrounding comments:

```
/***** UNCOMMENT TO USE THE Media MODULE *****
...
**** END Media MODULE COMMENT ***/
```

Add the JavaScript Media Module to a Web Page

The Media module code is copied into your `s_code.js` file.

1. Add a call to `s.loadModule` to load the Media module.

   ```
   /*********Media Module Calls**************/
   s.loadModule("Media")
   ```

2. Insert the Media module code into the modules section.

   ```
   /************************** MODULES *************************/
   /* Insert the media module tracking code here. */
   ```

See *JavaScript Sample Code*.

Map Analytics Variables and Events

After you insert the code in your project, you need to map the conversion variables and events you are using to track video.

If the Web Analytics team filled out the *Video Implementation Worksheet*, you might have a list similar to the following:

- 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`

Complete the following to map conversion variables and events:

1. Add the `s.Media.trackUsingContextData` variable and set it to true:

```
s.Media.trackUsingContextData = true;
```

2. Add the `s.Media.contextDataMapping` variable and map the parameters it contains with the SiteCatalyst variables and events you selected:

```
s.Media.trackUsingContextData = true
s.Media.contextDataMapping = {
  "a.media.name":"eVar2,prop2",
  "a.media.segment":"eVar3",
  "a.contentType":"eVar1",
  "a.media.timePlayed":"event3",
  "a.media.view":"event1",
  "a.media.segmentView":"event2",
  "a.media.complete":"event7"
};
```

Next step: *Configure Milestones*

Configure Milestones

Video milestones determine specific points in the video that you want to track.

If the web analytics team filled out the *Video Implementation Worksheet*, you might have a list similar to the following:

Number of seconds between measurement calls (increments of 5): `30`

Milestones to track (as a % of video length): `25%, 50%, 75%`

Divide each video into segments based on: _____

Review *Video Metrics* then complete the following to define the collection interval:

1. (Optional) To track using seconds, add the following:

```
s.Media.trackSeconds = 30;
```

2. (Optional) To track using milestones, add additional milestone events to the milestones parameter of your contextDataMapping variable:

```
s.Media.trackUsingContextData = true
s.Media.contextDataMapping = {
  "a.media.name":"eVar2,prop2",
  "a.media.segment":"eVar3",
  "a.contentType":"eVar1",
  "a.media.timePlayed":"event3",
  "a.media.view":"event1",
  "a.media.segmentView":"event2",
  "a.media.complete":"event7",
  "a.media.milestones":{
     25: "event4",
     50: "event5",
     75:"event6"
     }
};
s.Media.trackMilestones = "25,50,75";
```

This example measures an event for the 25%, 50%, and 75% milestones. Each milestone you track must also be specified in `trackMilestones`.

You can also track using offset milestones instead:

```
"a.media.milestones":{
     30: "event4", // 30 seconds from start of video
     60: "event5", // 60 seconds from start of video
     120:"event6"  // 120 seconds from start of video
     }
};
s.Media.trackOffsetMilestones = "30,60,120";
```

This example measures an event 30, 60, and 120 seconds from the start of the video. Each offset milestone you track must also be specified in `trackOffsetMilestones`.

3. You can use `segmentByMilestones` to have the media module create segments automatically based on your milestones.

```
SegmentByMilestones = true;
```

If you do not enable `segmentByMilestones` to define segments, you must use a manual implementation (not `autoTrack`) and send in the segment details with `Media.play`. If the web analytics team has defined segments to track, you can define milestones that correspond to each segment then enable `segmentByMilestones`.

Next step: *Track Player Events Using AutoTrack*.

Track Video Player Events

You can track media players by creating functions attached to the video player event handlers

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

Load: Call `Media.open` and `Media.play`

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

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

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

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

End: Call `Media.stop`, then `Media.close`. For example, at the end of a 100-second video, call `s.Media.stop("Video1",100)`, then `s.Media.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 `Media.open`, `Media.play`, `Media.stop`, and `Media.close` come from the player. The following pseudocode demonstrates how this might be done:

```
/*Call on video load*/
function startMovie(){
 s.Media.open(mediaName,mediaLength,mediaPlayerName);
 playMovie();
}

/*Call on video resume from pause and slider release*/
function playMovie(){
 s.Media.play(mediaName,mediaOffset, segmentNum, segment, segmentLength);
}

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

/*Call on video end*/
```

```
function endMovie(){
 stopMovie();
 s.Media.close(mediaName);
}
```

JavaScript AutoTrack

The JavaScript media module for identifies all `<embed>` or `<object>` tags in the page HTML. It then searches the data in each tag to determine which media player, if any, is being used. If the player is Windows Media Player, Quicktime, or Real Player, `autoTrack` can be used, though autoTrack for Windows media player works only with Internet Explorer. Manual tracking for Windows Media Player is required to support all other browsers.

You must have the `classid` attribute set on the object you want to track. The `classid` is required to expose the event handlers used by the Media Module to automatically track the video.

```
s.Media.autoTrack = true
```

JavaScript Sample Code

This section contains a sample implementation in JavaScript.

```
s.usePlugins=true
function s_doPlugins(s) {

    /* Add manual calls to modules and plugins here */
}
s.doPlugins=s_doPlugins

/*********Media Module Calls**************/
s.loadModule("Media")

/*Configure Media Module Functions */
s.Media.autoTrack= true;
s.Media.trackVars="events,prop2,eVar1,eVar2,eVar3";
s.Media.trackEvents="event1,event2,event3,event4,event5,event6,event7"
s.Media.trackMilestones="25,50,75";
s.Media.playerName="My Media Player";
s.Media.segmentByMilestones = true;
s.Media.trackUsingContextData = true;
s.Media.contextDataMapping = {
  "a.media.name":"eVar2,prop2",
  "a.media.segment":"eVar3",
  "a.contentType":"eVar1",
  "a.media.timePlayed":"event3",
  "a.media.view":"event1",
  "a.media.segmentView":"event2",
  "a.media.complete":"event7",
  "a.media.milestones":{
     25:"event4",
     50:"event5",
     75:"event6"
  }
}

  s.Media.monitor = function (s,media){ //If Needed
  }

  /* Turn on and configure debugging here */
  s.debugTracking = true;
  s.trackLocal = true;

 /* WARNING: Changing any of the below variables will cause drastic changes to how your visitor
 data is collected.  Changes should only be made     when instructed to do so by your account
 manager.*/
  s.visitorNamespace = "yourNamespace";
  s.trackingServer="metrics.mysite.com" //Use only if using first party cookies
```

```
  s.trackingServerSecure="smetrics.mysite.com" //Use only if useing first party cookies in
conjunction with SSL
  s.dc = '122';

  /************************** PLUGINS SECTION *************************/
  /* Insert any plugins code you want to use here. */

  /***************************** MODULES *****************************/
  /* Insert the media module tracking code here. */
```

HTML 5 Video

HTML 5 video is measured using JavaScript.

To measure HTML 5 video, follow the instructions provided in *Using JavaScript to Track a Video Player* to configure video tracking using JavaScript. For example, add the Media Module to your `s_code` file, configure conversion variables, events, and milestones, and so on.

As described in *Track Video Player Events*, you must manually track the HTML 5 video events. The HTML 5 specification provides a large number of predefined event handlers you can use to manually track HTML 5 video. The following site provides details on the HTML 5 video events:

http://www.w3.org/2010/05/video/mediaevents.html

Using these event handlers, you can make calls to open, play, stop, and close to measure video. The following code provides some examples of using the HTML 5 video event handlers:

```
var video = document.getElementsByTagName('video')[0];

video.play = function(e) {
s.Media.play(mediaName,mediaOffset, segmentNum, segment, segmentLength);
}

video.pause = function(e) {
s.Media.stop(mediaName,mediaOffset);
}

video.onended = function(e) {
s.Media.close(mediaName);
}
```

Other Video Players

This section contains notes on measuring video in players that are not specifically defined in this guide.

. Many 3rd party Flash-based players are built on OSMF and can use the OSMF dynamic plug-in. Other players that expose a JavaScript event interface can use the JavaScript media module.

- *Open Source Media Framework (OSMF)*
- *Using JavaScript to Track a Video Player*
- *Custom Flash NetStream Player*
- *Brightcove*

Custom Flash NetStream Player

Flash 10.3 introduced new functionality to the NetStream component that enables enhanced video tracking. If you are using a custom Flash NetStream player you can enable the following variable to enable player tracking functionality similar to autoTrack.

```
s.Media.autoTrackNetStreams = true
```

The process you follow is similar to the process described in *Flash Video Playback*. You can modify these instructions to download the correct Flash AppMeasurement library for your development library and configure video measurement.

See *Measuring video consumption in Flash*.

Brightcove

Brightcove provides an interface for 3rd party analytics solutions using an analytics SWF Flash component.

http://support.brightcove.com/en/docs/editing-settings-players-plug-ins-tab

Adobe provides an analytics SWF, `AppMeasurementExtension.swf`, for this purpose. To complete a Brightcove integration you need the following files:

File	Instructions
`AppMeasurementExtension.swf`	1. In the Adobe Marketing Cloud, click **Admin** > **Admin Console** > **Code Manager**. 2. Select **ActionScript (Flash/Flex)**, provide the requested information, and then click **Generate Code**. 3. From the **Component Files** Tab, download `AppMeasurementExtension.swf`.
XML Configuration File	An XML configuration file is used to map the SiteCatalyst variables you want to use for video, and define milestones. See the section below.

XML Configuration File

When using a dynamic OSMF implementation, you can use an XML config file to bind variables to OSMF metadata. AppMeasurement uses the following variable binding syntax:

```
<variable>{media.player.metadata(namespace,key)}</variable>
```

variable: The name of the variable you wish to set (for example, eVar6).

namespace: (Optional) The OSMF metadata namespace you want to use. If you do not specify a namespace, the AppMeasurement OSMF plug-in uses the first matching key it locates in any namespace. When looking for keys, the plug-in looks first at MediaElement metadata, then at MediaResource metadata.

key: The specific metadata value you want to use.

The following section contains a sample XML configuration file.

• The `trackSeconds` and `milestone` sections are optional. See *Video Metrics*.

If the Web Analytics team filled out the *Video Implementation Worksheet*, you might have a list similar to the following:

• 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`

Map these variables to the appropriate `a.media` variable in the `contextDataMapping` section.

```
<config>
<account>myrsid</account>
```

```
<debugTracking>true</debugTracking>
<visitorNamespace>corp1</visitorNamespace>
<trackingServer>corp1.d1.sc.omtrdc.net</trackingServer>
<Media>
    <autoTrack>true</autoTrack>
    <trackMilestones>25,50,75</trackMilestones>
    <trackVars>events,eVar1,eVar2,eVar3,prop2</trackVars>
    <trackEvents>event1,event2,event3,event4,event5,event6,event7</trackEvents>
    <segmentByMilestones>true</segmentByMilestones>
    <trackUsingContextData>true</trackUsingContextData>
    <contextDataMapping>
            <a.media.name>eVar2,prop2</a.media.name>
            <a.media.segment>eVar3</a.media.segment>
            <a.contentType>eVar1</a.contentType>
            <a.media.timePlayed>event3</a.media.timePlayed>
            <a.media.view>event1</a.media.view>
            <a.media.segmentView>event2</a.media.segmentView>
            <a.media.complete>event7</a.media.complete>
            <a.media.milestones>
                <item name="25">event4</item>
                <item name="50">event5</item>
                <item name="75">event6</item>
            </a.media.milestones>
    </contextDataMapping>
</Media>
</config>
```

Be aware of the following if you are using Brightcove Players:

- `Media.autoTrack` supports externally loaded video players, such as Brightcove, that might not be instantiated when the Flash creates the AppMeasurement instance. To recognize an external video player, `Media.autoTrack` periodically rechecks for a video player if it does not initially find one.
- AppMeasurement tracks Brightcove video playback using the Brightcove video ID, prefixed with either Brightcove 2: or Brightcove 3: (depending on the Brightcove API version). For example, a Brightcove 3 video ID of abc123667 gets a video name of Brightcove 3:abc123667.
- AppMeasurement gets the video ID from the video DTO calling getCurrentVideo or getCurrentTitle.
- To improve video name readability, you can upload classifications that assign the video a friendly name. For example, Brightcove 3:abc123667 could be classified as On-line Auto Ad #1.

If you are already building a more complex Flash movie and using the Brightcove video player with your movie, you can also implement AppMeasurement using the instructions in *Flash Video Playback*. This lets AppMeasurement detect the Brightcove video player in your movie when it loads.

Manually Tagging a Video Player

All video players that do not support Autotrack must be manually tagged.

See *What is Autotrack?*.

Track Video Player Events

You can track media players by creating functions attached to the video player event handlers

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

Load: Call `Media.open` and `Media.play`

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

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

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

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

End: Call `Media.stop`, then `Media.close`. For example, at the end of a 100-second video, call `s.Media.stop("Video1",100)`, then `s.Media.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 `Media.open`, `Media.play`, `Media.stop`, and `Media.close` come from the player. The following pseudocode demonstrates how this might be done:

```
/*Call on video load*/
function startMovie(){
 s.Media.open(mediaName,mediaLength,mediaPlayerName);
 playMovie();
}

/*Call on video resume from pause and slider release*/
function playMovie(){
 s.Media.play(mediaName,mediaOffset, segmentNum, segment, segmentLength);
}

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

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

JavaScript AutoTrack

The JavaScript media module for identifies all `<embed>` or `<object>` tags in the page HTML. It then searches the data in each tag to determine which media player, if any, is being used. If the player is Windows Media Player, Quicktime, or Real Player, `autoTrack` can be used, though autoTrack for Windows media player works only with Internet Explorer. Manual tracking for Windows Media Player is required to support all other browsers.

You must have the `classid` attribute set on the object you want to track. The `classid` is required to expose the event handlers used by the Media Module to automatically track the video.

```
s.Media.autoTrack = true
```

Measuring Additional Metrics using Media.monitor

You can define a custom `Media.monitor` method to track additional video metrics.

A custom `Media.monitor` method gives you the most granular control over video tracking. AppMeasurement automatically calls the `Media.monitor` method in the following circumstances:

- Every second while the video is playing.
- When an autoTrack implementation, such as the OSMF plug-in, captures a player event like scrubbing, pausing or resuming, end of video playback, etc.
- When a non-autoTrack (manual) implementation calls `Media.open`, `Media.play`, `Media.stop`, or `Media.close`.

Use this functionality to monitor the status of each video that is currently playing. With it, you can setup additional variables (Props, eVars, Events) and call `Media.track` based on the current state of the video as it is playing.

```
s.Media.monitor(s, media)
```

This method takes the following parameters:

s: The AppMeasurement instance.

media: An object with members providing the state of the video. These members include:

State Property	Description
`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.openTime`	A Date object set to the time video tracking started.
`media.offset`	The current offset into the video, in seconds.
`media.percent`	The current offset into the video, as a percentage of the video length ((media.offset / media.length) * 100).
`media.timePlayed`	The total video playback time, in seconds. This includes replay time due to rewind, and excludes time skipped by fast forwarding.
`media.playerName`	The name of the media player given in the call to `Media.open`.
`media.mediaEvent`	The event that initiated the call to the `Media.monitor` method. Options include the following: **OPEN**: The start of video playback. For a non-autoTrack (manual) implementation, this is the first time `Media.play` is called after calling `Media.open`. **PLAY**: Video playback restarted. **STOP**: Video playback stopped due to pause, seek forward or backwards, scrubbing started, etc. **CLOSE**: The end of video playback. For a non-autoTrack (manual) implementation, this is when `Media.close` is called. **MONITOR**: A check of the video state, which occurs every second. **MILESTONE**: A video milestone was reached, as defined by the `Media.trackMilestones` method. The `media.milestone` property identifies the specific milestone that was reached. **SECONDS**: The playback time specified by `media.trackSeconds` has been reached.
`media.eventFirstTime`	A boolean property. When set to true, indicates the `media.mediaEvent` occurred for the first time. If media.mediaEvent = "MILESTONE" a true value indicates that this is the first time to reach the video milestone identified in media.milestone.
`media.milestone`	The milestone reached during video playback, as defined in the `Media.trackMilestones` method. This property is only set when `media.mediaEvent` is set to `"MILESTONE"`.
`media.player`	Player-specific data created by some autoTrack implementations.
`media.player.metadata`	Player-specific method for retrieving video metadata from the player.

A custom `Media.monitor` method should do similar to the following:

- Check the media object to see if this is a point where you want to customize your video tracking.
- Set custom props, eVars, and events to customize your tracking.
- Set `s.Media.trackVars` and `s.Media.trackEvents` filters to match the custom props, eVars, and events you with to track along with the video data.
- Call `s.Media.track(media.name)` to send off the video data collected up to the current point along with your custom props, eVars, and events.

Define a method or function for Media.monitor

To create a custom `Media.monitor` method you can set `Media.monitor` to an anonymous function or a class method. The anonymous function or class method you define should take the AppMeasurement instance and media object as shown in the media monitor reference.

```
//JavaScript or Flash anonymous function example
s.Media.monitor = function (s,media)  {
...
}
```

```
//Silverlight class method example
s.Media.monitor = new AppMeasurement_Media_Monitor(myMediaMonitor);
```

```
//OSMF anonymous function example
s.Media.monitor = function (s:Object,media:Object) {
...
}
```

Media.monitor Code Sample

The following code sample demonstrates using Media.monitor to send custom variables.

```
/* Import line for ActionScript 3 */
import com.omniture.AppMeasurement;

/* Uncomment for ActionScript 2 with Flash Player 8+ and comment out other import lines */
/* import com.omniture.AS2.AppMeasurement; */

/* Uncomment for ActionScript 2 with Flash Player 6, 7, or Lite and comment out other import
lines */
/* import com.omniture.AS2.FPL.AppMeasurement; */

var s:AppMeasurement = new AppMeasurement();
/* Uncomment for Flex and comment out addChild(s); */
/* rawChildren.addChild(s); */
addChild(s);

/* Specify the Report Suite ID(s) to track here */
s.account = "jdoe";
/* Turn on and configure debugging here */
s.debugTracking = true;
s.trackLocal = true;
/* You may add or alter any code config here */
s.pageName = "";
s.pageURL = "";
s.charSet = "UTF-8";
s.currencyCode = "USD";

s.Media.autoTrack=true;
s.Media.segmentByMilestones=true;
s.Media.trackMilestones="25,50,75";
s.Media.trackVars="eVar2,eVar3,eVar1,events,prop51,prop50";
s.Media.trackEvents="event1,event2,event3,event4,event5,event6,event7";

/* Turn on and configure ClickMap tracking here */
s.trackClickMap = true;
```

```
s.movieID = "";

/* WARNING: Changing any of the below variables will cause drastic changes
to how your visitor data is collected.  Changes should only be made
when instructed to do so by your account manager.*/
s.visitorNamespace = "corp1";
s.trackingServer = "corp1.d1.sc.omtrdc.net";
var tracked25:Boolean
var tracked50:Boolean
var tracked75:Boolean
var fireRequest:Boolean

s.Media.trackUsingContextData = true;

s.Media.contextDataMapping = {
"a.media.name":"eVar2",
"a.media.segment":"eVar3",
"a.contentType":"eVar1",
"a.media.timePlayed":"event3",
"a.media.view":"event1",
"a.media.segmentView":"event2",
"a.media.complete":"event7",
"a.media.milestones":{
25:"event4",
50:"event5",
75:"event6"
}
};

s.Media.monitor = function (s,media){

    if ((media.event == "MILESTONE") && (media.eventFirstTime)) {
     if (media.milestone == 25) {
          s.prop51 = media.name+ " : " +"25%";
          fireRequest = true;
     }
     if (media.milestone == 50) {
          s.prop51 = media.name+ " : " +"50%";
          fireRequest = true;
     }
     if (media.milestone == 75) {
          s.prop51 = media.name+ " : " +"75%";
          fireRequest = true;
     }
     if (fireRequest) {
          fireRequest = false;
          sendRequest();
     }
}

    if(media.event=="OPEN") {
        s.prop50="Home page"
        sendRequest();
        s.prop50=""

    }

    if(media.event=="CLOSE") {
        s.prop51=media.name+ " : " +"100%"
        sendRequest();
        s.prop51=""

    }

    function sendRequest(){
        s.Media.track(media.name);
    }
}
```

The following custom `Media.monitor` method sets eVar1 to the “series” key, eVar2 to the “season” key, and eVar3 to the “episode” key in the http://www.corp1.com/ namespace using OSMF metadata:

```
s.Media.monitor = function (s:Object,media:Object) {
    s.trackVars = "events,eVar1,eVar2,eVar3";
    s.trackEvents = "event1,event2";
    s.eVar1 = media.player.metadata("http://www.corp1.com/","series");
    s.eVar2 = media.player.metadata("http://www.corp1.com/","season");
    s.eVar3 = media.player.metadata("http://www.corp1.com/","episode");
    if (media.mediaEvent == "OPEN") {
        s.events = "event1";
        s.Media.track(media.name);
    } else if (media.mediaEvent == "CLOSE") {
        s.events = "event2";
        s.Media.track(media.name);
    }
}
```

The following example shows a `Media.monitor` implementation in Silverlight:

```
s.Media.media = new AppMeasurement_Media_Monitor(myMediaMonitor);

// ...

// custom variables (Props, eVars, Events) are sent automatically on an OPEN event, and when
manually tracked here (in media monitor below)
private void myMediaMonitor(AppMeasurement s,AppMeasurement_Media_State media) {
     if (media.mediaEvent == "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 == "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);
     }
```

Media Module Variables

The following variables let you configure video measurement.

You must define values for the variables in the Required Variables table.

Additionally, to track events in your video player, you must enable autoTrack (for supported players) or implement custom player event tracking using the open, play, stop, and close methods.

Variable	Description
`Media.trackUsingContextData`	**Syntax:** `s.Media.trackUsingContextData = true;` This option enables integrated video tracking. When `trackUsingContextData = true`, the media module generates context data for media tracking, instead of the legacy pev3 value used in previous versions of video measurement. Use `Media.contextDataMapping` to map the context data to the selected eVars and Events. Defaults to `false`.
`Media.contextDataMapping`	**Syntax:** `s.Media.contextDataMapping = { "a.media.name":"eVar2,prop2", "a.media.segment":"eVar3", "a.contentType":"eVar1", "a.media.timePlayed":"event3", "a.media.view":"event1", "a.media.segmentView":"event2", "a.media.complete":"event7", "a.media.milestones":{ 25:"event4", 50:"event5", 75:"event6" } };` An object that defines variable mapping to eVars and Events that you want to use for video measurement. The object must map the following fields: **a.media.name**: (Required) Populates variables with the video name. Provide the eVar that you selected to store the video name, and the Custom Insight Video variable (s.prop) you want to use for video pathing. Provide the values in a comma-separated list. **a.media.segment**: (Optional) The eVar that you want to store the media segment name. **a.contentType**: (Optional) The eVar that you want to store the video value, which contains visit and visitor tracking enabled to generate video visit and visitor reporting. The variable you select is likely already used to store data such as article slide show or product page **a.media.view**: (Required) The Event that you want to count media views.

Variable	Description
	a.media.segmentView: (Optional) The Event that you want to count segment views. **a.media.complete**: (Optional) The Event that you want to count complete views. **a.media.timePlayed**: (Optional, highly recommended) The numeric Event that you want to store the number of video seconds played. **a.media.milestones**: (Optional) An object that maps s.Media.trackMilestones milestones to counter Events. Media.segmentByMilestones should be set to true if you define milestones. **Ad tracking** To track ads, the following context data variables are available: **a.media.ad.name**: (Required) Populates variables with the ad name. Provide the eVar that you selected to store the ad name, and the Custom Insight Video variable (s.prop) you want to use for pathing. Provide the values in a comma-separated list. **a.media.ad.pod**: The position in the primary content the ad was played. **a.media.ad.podPosition:** The position within the pod where the ad is played. **a.media.ad.CPM:** The CPM or encrypted CPM (prefixed with a "~") that applies to this playback. **a.media.ad.view:** Works the same as a.media.view. **a.media.ad.clicked:** Count the number of clicks for the ad (Media.click calls). **a.media.ad.timePlayed:** Works the same as a.media.timePlayed. **a.media.ad.complete:** Works the same as a.media.complete. **a.media.ad.segment:** Works the same as a.media.segment. **a.media.ad.segmentView:** Works the same as a.media.segmentView. **a.media.ad.milestones:** Works the same as a.media.milestones. **a.media.ad.offsetMilestones:** Works the same as a.media.offsetMilestones.
`Media.trackVars`	**Syntax:** `s.Media.trackVars="events,prop2,eVar1,eVar2,eVar3";` A comma-separated list of all variables that are set in your video tracking code.
`Media.trackEvents`	**Syntax:** `s.Media.trackEvents="event1,event2,event3,event4,event5,event6,event7"` A comma-separated list of all events that are set in your video tracking code.

The following table contains optional variables.

Optional Variables

Variable	Description
`Media.autoTrack`	**Syntax:** `s.Media.autoTrack = true` Enables automatic tracking for supported players. Supported players are as follows: • Open Source Media Framework (OSMF) • FLVPlayback (Video players created by the import video wizard in Flash Professional) • Silverlight • MediaDisplay • MediaPlayback • Brightcove API versions 2 & 3 (see *Brightcove*) • Windows Media Player, Quicktime, or Real Player using JavaScript If you are not using one of the above players you can use Media.open, Media.play, Media.stop, and Media.close to track player events.
`Media.autoTrackNetStreams`	**Syntax:** `s.Media.autoTrackNetStreams = true` Flash 10.3 introduced new functionality to the NetStream component that enables enhanced video tracking. If you are using a custom Flash NetStream player you can enable this variable to enable functionality similar to autoTrack. This method requires that videos are viewed in Flash 10.3 or later.
`Media.completeByCloseOffset`	**Syntax:** `s.Media.completeByCloseOffset = true` This setting lets you count a complete video view a few seconds before the actual end of the video. The event is sent based on the number of seconds specified in `completeCloseOffsetThreshold`. This lets you measure completes in video players that never report an offset equal to the length of the video. By default, this value is set to true and the threshold is set to 1 second. With these defaults the complete event is sent 1 second before the end of the video.
`Media.completeCloseOffsetThreshold`	**Syntax:** `s.Media.completeCloseOffsetThreshold = 1` This threshold lets you count a complete video view a few seconds before the actual end of the video. `Media.completeByCloseOffset` must be set to true to use this threshold. The integer value you supply determines how far off in seconds the offset can be from the length of the video at close and still count as a complete. This lets you measure completes in video players that never report an offset equal to the length of the video.

Variable	Description
	The default threshold is 1 second.
`Media.playerName`	**Syntax:** `s.Media.playerName = "Custom Player Name"` Specifies a custom video player name.
`Media.trackSeconds`	`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. Enabling `Media.trackSeconds` triggers only the events that are defined in Media.contextDataMapping. To send additional variables outside of those specified for video measurement, you must use Media.Monitor
`Media.trackMilestones`	Tracks milestones as percentage of the video length. **Syntax:** `s.Media.trackMilestones = "25,50,75";` Defines the interval, as a percentage of the video length, 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%). 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, the media module sends the tracking data multiple times. Similarly, if a visitor fast forwards past a milestone, the media module does not send the tracking data for that milestone. Enabling `Media.trackMilestones` triggers only the events that are defined in `Media.contextDataMapping`. To send additional variables outside of those specified for video measurement, you must use `Media.monitor`.
`Media.trackOffsetMilestones`	Tracks milestones as seconds elapsed from the beginning of the video. **Syntax:** `s.Media.trackOffsetMilestones = "20,40,60";` Defines the interval, as seconds elapsed from the beginning of the video, for sending video tracking data to Adobe data collection servers. Specify the milestones as a comma-separated list of whole numbers. For example: 20 = 20 seconds, 40 = 40 seconds). Because these milestones are fixed points in the video, if a visitor views past the 20 seconds milestone, then rewinds and passes the 20 seconds milestone again, the media module sends the tracking data multiple times. Similarly, if a visitor fast forwards past a milestone, the media module does not send the tracking data for that milestone.

Variable	Description
	Enabling `Media.trackOffsetMilestones` triggers only the events that are defined in `Media.contextDataMapping`. To send additional variables outside of those specified for video measurement, you must use `Media.monitor`.
`Media.segmentByMilestones`	**Syntax:** `s.Media.segmentByMilestones = true;` Automatically generates the segment name, segment number, and segment length data, based on the length of the media and the milestones specified in `Media.trackMilestones`. Segmenting by milestones is the only way to define segments when using autoTrack. Defaults to `false`.
`Media.segmentByOffsetMilestones`	**Syntax:** `s.Media.segmentByOffsetMilestones = true;` Automatically generates the segment name, segment number, and segment length data, based on the length of the media and the milestones specified in `Media.trackOffsetMilestones`. Segmenting by milestones is the only way to define segments when using autoTrack. Defaults to `false`.

Ad Tracking Variables

These variables are used to send ad information in conjunction with the `openAd` method. See *VAST Video Ad Tracking*.

Variable	Description
`Media.adTrackSeconds`	`s.Media.adTrackSeconds = 15` Defines the interval, in seconds, for sending video ad tracking data to Adobe data collection servers while the video is playing. The value must be set in increments of 5 seconds. Enabling `Media.adTrackSeconds` triggers only the events that are defined in `Media.contextDataMapping`. To send additional variables outside of those specified for video measurement, you must use `Media.Monitor`.
`Media.adTrackMilestones`	Tracks ad milestones as percentage of the ad length. **Syntax:** `s.Media.adTrackMilestones = "25,50,75";`

Variable	Description
	Defines the interval, as a percentage of the ad length, for sending ad tracking data to Adobe data collection servers. Specify the milestones as a comma-separated list of whole numbers. For example: 10 = 10%, 23 = 23%). Because these milestones are fixed points in the ad, if a visitor views past the 10% milestone, then rewinds and passes the 10% milestone again, the media module sends the tracking data multiple times. Similarly, if a visitor fast forwards past a milestone, the media module does not send the tracking data for that milestone. Enabling `Media.adTrackMilestones` triggers only the events that are defined in `Media.contextDataMapping`. To send additional variables outside of those specified for video measurement, you must use `Media.monitor`.
`Media.adTrackOffsetMilestones`	Tracks ad milestones as seconds elapsed from the beginning of the ad. **Syntax:** `s.Media.adTrackOffsetMilestones = "20,40,60";` Defines the interval, as seconds elapsed from the beginning of the ad, for sending ad tracking data to Adobe data collection servers. Specify the milestones as a comma-separated list of whole numbers. For example: 20 = 20 seconds, 40 = 40 seconds). Because these milestones are fixed points in the ad, if a visitor views past the 20 seconds milestone, then rewinds and passes the 20 seconds milestone again, the media module sends the tracking data multiple times. Similarly, if a visitor fast forwards past a milestone, the media module does not send the tracking data for that milestone. Enabling `Media.adTrackOffsetMilestones` triggers only the events that are defined in `Media.contextDataMapping`. To send additional variables outside of those specified for video measurement, you must use `Media.monitor`.
`Media.adSegmentByMilestones`	**Syntax:** `s.Media.adSegmentByMilestones = true;` Automatically generates the segment name, segment number, and segment length data, based on the length of the media and the milestones specified in `Media.adTrackMilestones`. Segmenting by milestones is the only way to define segments when using autoTrack. Defaults to `false`.
`Media.adSegmentByOffsetMilestones`	**Syntax:** `s.Media.adSegmentByOffsetMilestones = true;`

Variable	Description
	Automatically generates the segment name, segment number, and segment length data, based on the length of the media and the milestones specified in `Media.adTrackOffsetMilestones`. Segmenting by milestones is the only way to define segments when using autoTrack. Defaults to `false`.

Media Module Methods

The media module methods are used to manually tracking player events and to track additional metrics that are not part of the standard video reports.

If you are using `Media.autoTrack` and are not tracking additional metrics, you do not need to call any of these methods directly. All arguments are required unless specified as optional.

Method	Description
`Media.open`	**Syntax:** `s.Media.open(mediaName,mediaLength,mediaPlayerName)` Prepares the media module to collect video tracking data. This method takes the following parameters: **mediaName**: (required) The name of the video as you want it to appear in video reports. **mediaLength**: (required) The length of the video in seconds. **mediaPlayerName**: (required) The name of the media player used to view the video, as you want it to appear in video reports.
`Media.openAd`	**Syntax:** `s.Media.openAd(name,length,playerName,parentName,parentPod,parentPodPosition,CPM)` Prepares the media module to collect ad tracking data. This method takes the following parameters: **name**: (required) The name or ID of the ad. **length**: (required) The length of the ad. **playerName**: (required) The name of the media player used to view the ad. **parentName**: The name or ID of the primary content where the ad is embedded. **parentPod**: The position in the primary content the ad was played. **parentPodPosition**: The position within the pod where the ad is played. **CPM**: The CPM or encrypted CPM (prefixed with a "~") that applies to this playback.
`Media.click`	**Syntax:** `s.Media.click(name,offset)` Track when an ad is clicked in a video. This method takes the following parameters: **name**: The name of the ad. This must match the name used in `Media.openAd`. **offset**: The offset into the ad when the click occurred.
`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. This must match the name used in Media.open.

Method	Description
`Media.complete`	**Syntax:** `s.Media.complete(name,offset)` This method manually tracks a complete event. This method is used when you need to trigger events using special logic that can't be handled using `Media.completeByCloseOffset`. For example, if you are measuring a live stream that has no defined end, you might trigger a complete after a user views a live stream for X seconds. You might measure a complete using a percentage calculation based on the length and type of content. This method takes the following parameters: **mediaName**: The name of the video. This must match the name used in Media.open. **mediaOffset**: The number of seconds into the video when the complete event should be sent. Specify the offset based on the video starting at second zero. If your media player tracks using milliseconds, make sure the value is converted to seconds before you call Media.complete. If you plan to call complete manually, set `s.Media.completeByCloseOffset = false` to disable automatic triggering of the complete event.

`Media.play`	**Syntax:**
	`s.Media.play(name,offset,segmentNum,segment, segmentLength)`
	Call this method anytime a video starts playing. When using manual video measurement, you can provide the current segment data when sending video measurement data.
	If your player changes from one segment to another, for whatever reason, you should call Media.stop before calling Media.play again for the new segment.
	This method takes the following parameters:
	mediaName: The name of the video. 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. If your media player tracks using milliseconds, make sure the value is converted to seconds before you call Media.play.
	segmentNum: (Optional) The current segment number, which marketing reports use to order the display of segments in reports. The segmentNum parameter must be greater than zero.
	segment: (Optional) The current segment name.
	segmentLength: (Optional) The current segment length, in seconds.
	For example:
	`s.Media.play("My Video",1800,2,"Second Quarter",1800)`
	`s.Media.play("My Video",0,1,"Preroll",30)`

`Media.stop`	**Syntax:** `s.Media.stop(mediaName,mediaOffset)`

Method	Description
	Tracks a stop event (stop, pause, etc.) for the specified video. This method takes the following parameters: **mediaName**: The name of the video. This must match the name used in Media.open. **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.

`Media.monitor`	**Syntax:**
	`s.Media.monitor(s, media)`
	
	Silverlight Syntax:
	`s.Media.monitor = new AppMeasurement_Media_Monitor(myMediaMonitor);`
	
	The Silverlight app media monitor implements the Objective-C delegate design pattern. `myMediaMonitor` is a class method that takes the `s` and `media` parameters.
	
	Use this method to send additional video metrics. You can setup additional variables (Props, eVars, Events) and send them using `Media.track` based on the current state of the video as it is playing.
	
	See *Measuring Additional Metrics using Media.monitor*.
	
	This method takes the following parameters:
	
	s: The AppMeasurement instance (or JavaScript s object).
	
	media: An object with members providing the state of the video. These members include:
	• **media.name**: The name of the video. This must match the name used in 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**: When playback is first observed through `Media.autoTrack` or a call to `Media.play`.
	  • **CLOSE**: When playback ends at the completion of the video through `Media.autoTrack` or at a call to Media.close.
	  • **PLAY**: When playback resumes after being paused or scrubbing through `Media.autoTrack` or a second call to `Media.play`.
	  • **STOP**: When playback stops due to a pause of the beginning of scrubbing through Media.autoTrack or a call to Media.stop.
	  • **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.
	• **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.

Method	Description
	• **media.eventFirstTime**: Indicates if this was the first time this media event was called for this video.
`Media.track`	**Syntax:** `s.Media.track(mediaName)` Immediately sends the current video state, along with any `Media.trackVars` and `Media.trackEvents` you've defined. This method is used within `Media.monitor`. See *Measuring Additional Metrics using Media.monitor*. 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. This must match the name used in Media.open. This method is the only way to send additional variables while the video is playing. This method resets the seconds interval and percent milestone counters to zero to prevent multiple tracking hits.

VAST Video Ad Tracking

AppMeasurement provides support to track ads displayed in videos.

Ads are tracked similar to primary content videos, with additional parameters tracked with them to associate them with the primary content in which they are embedded. The extra parameters are:

- Primary content video name - The identifier for the primary content video in which the ad is tracked.
- Ad pod and pos position - The pod and position the ad is which the ad is being played in the primary content video.
- Ad CPM - The CPM or encrypted CPM for this ad play

Otherwise ads are tracked using the same methods used to track videos. The Ad ID replaces the video name when tracking ads.

This example shows a method of tracking a VAST ad when no ad length is available. If the ad length is available, you can remove the manual `complete` call and include the length in the call to `openAd`.

```
//stop primary video
s.Media.stop("My Primary Video", 60) //replace 60 with offset location where ad was displayed
//play the ad
s.Media.openAd("My VAST Ad",-1,"Freewheel","My Primary Video","Preroll",0);
s.Media.play("My VAST Ad",0);
// 30 seconds later or whenever the Freewheel ad player stops playing
s.Media.complete("My VAST Ad",30);
s.Media.stop("My VAST Ad",30);
s.Media.close("My VAST Ad");
//start primary video
s.Media.play("My Primary Video", 60)
```

Step	Task	Description
1	Stop the primary content and start the ad.	Call `s.Media.stop("My Primary Video"...)`, then call `s.Media.openAd("My VAST Ad"...)` as show in the code example above. Since the ad length is unknown, `-1` is passed as the length. This means that you will need to manually call complete. In the case where the video ad is an overlay, you can call `openAd` without stopping the primary video.
2	Play the ad.	`s.Media.play("My VAST Ad",0)`

3	When the ad finishes, call complete, then stop.	Include the ad length as the offset parameter for both calls. Offset represents the seconds into the content where the media event occurs, starting from 0.
		`s.Media.complete("My VAST Ad",30);`
		`s.Media.stop("My VAST Ad",30);`

4	Close the ad	`s.Media.close("My VAST Ad");`
5	Resume the primary video.	`s.Media.play("My Primary Video",60) //replace 60 with the actual offset location where the ad was displayed`

This same process can be used to track any type of ad, it is not restricted to ads that follow the VAST standard.

See `openAd` in *Media Module Methods* and the *Ad Tracking Variables*.

Additional ContextData Mapping for Ad Tracking

The following variables must be configured in the `s.Media.contextDataMapping` object to track ad metrics:

```
s.Media.contextDataMapping = {
"a.media.name":"eVar2,prop2",
"a.media.segment":"eVar3",
"a.contentType":"eVar1",
"a.media.timePlayed":"event3",
"a.media.view":"event1",
"a.media.segmentView":"event2",
"a.media.complete":"event7",
"a.media.milestones":{  //optionally replace with trackOffsetMilestones
   25:"event4",
   50:"event5",
   75:"event6"
  }
//ad tracking variables
"a.media.ad.name":"eVar4",
"a.media.ad.pod":"eVar5",
"a.media.ad.podPosition":"eVar6",
"a.media.ad.CPM":"eVar7,prop7",
"a.media.ad.clicked":"event14",
"a.media.ad.segment":"eVar8",
"a.media.ad.timePlayed":"event10",
"a.media.ad.view":"event8",
"a.media.ad.segmentView":"event9",
"a.media.ad.complete":"event14",
"a.media.ad.milestones":{  //optionally replace with trackOffsetMilestones
   25:"event11",
   50:"event12",
   75:"event13"
  }
};
```

contextData	Destination	Description
a.media.ad.name	eVar w/ full subrelations, visits, and visitors	This will be the name or ID of the ad
a.media.ad.pod	eVar	The position in the primary content the ad was played
a.media.ad.podPosition	eVar	The position within the pod where the ad is played
a.media.ad.CPM	**currency event**	The CPM or encrypted CPM (prefixed with a "~") that applies to this playback
a.media.ad.view	counter event	Works the same as a.media.view
a.media.ad.clicked	counter event	Count the number of clicks for the ad (Media.click calls)
a.media.ad.timePlayed	counter event	Works the same as a.media.timePlayed
a.media.ad.complete	counter event	Works the same as a.media.complete
a.media.ad.segment	eVar	Works the same as a.media.segment
a.media.ad.segmentView	counter event	Works the same as a.media.segmentView
a.media.ad.milestones	counter event	Works the same as a.media.milestones
a.media.ad.offsetMilestones	counter event	Works the same as a.media.offsetMilestones

Measuring Video FAQ

This topic provides answers to common questions.

What is Autotrack?

To effectively measure video, the media module needs a way to find out what is happening in your player. For example, when a user starts playing a video, the media module needs to start counting seconds viewed. If the user pauses the video, the media module must also pause the count as well.

If AutoTrack is supported for your player, it means that the code to monitor what is happening in your player is already present in the media module. For a developer, this means that you do not need to call open, play, stop, or close since the media module can already track these events.

If AutoTrack is not supported for your player, it means 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. Additional details are provided on how to do this in the Implementation Guides for video players that do not support autotrack.

What is the Media Module? Is it different from AppMeasurement?

The media module refers to the Media class that is part of the AppMeasurement libraries. All video measurement functionality is part of the media module, meaning that you reference video measurement variables and methods using the Media prefix. For example, If s is the name of your JavaScript object or AppMeasurement instance, reference media module components using the s.Media prefix.

For JavaScript, the media module must be downloaded and included separately. For all other languages the media module is part of the core AppMeasurement library.

The media module is implemented to be as identical as possible across all AppMeasurement libraries.

What is Media.monitor?

Media monitor lets you send additional metrics and perform other actions during playback. To use `Media.monitor`, you define a function that is automatically called:

- Every second while the video is playing.
- When an autoTrack implementation, such as the OSMF plug-in, captures a player event like scrubbing, pausing or resuming, end of video playback, etc.
- When a non-autoTrack (manual) implementation calls `Media.open`, `Media.play`, `Media.stop`, or `Media.close`.

Your function is provided a media object that contains details about the video state, including the event that triggered the call, where playback is occurring, and so on. You can then send additional metrics based on this information. For details see *Measuring Additional Metrics using Media.monitor*.

Can I a set video complete before the video reaches 100%?

Yes. For example, if you show credits at the end of your video, you might want to count a complete view before the end of the video file. To do this, specify a value for `s.Media.completeCloseOffsetThreshold` equal to the number of seconds before the end that you want to send a complete event. For example, if you show 10 seconds of credits at the end of your video, you could set `s.Media.completeCloseOffsetThreshold = 10`.

For live events and video streams that do not have a defined end, you can call complete manually. See *Media Module Methods*.

Do I have to dedicate a custom event to each milestone?

At a minimum, you should dedicate a custom event for video complete. Whether you send an event with the other milestones depends on what you want to track.

If you want to view fallout for a single video, the Video Detail report is populated with the segment eVar and Segment View event. This lets you see fallout without sending events for each milestone.

Sending in an event with each milestone lets you measure if a milestone is reached across multiple videos in one report. If you want to view data about a milestone in other reports or across multiple videos, you should send an event. If you are interested in viewing fallout for each video individually you typically don't need to send events for each milestone.

Migrating to Integrated Video Tracking

If you are currently tracking video using the legacy pev3 method, this section contains the information you need to migrate to integrated video tracking.

About Integrated Video Tracking

Integrated video tracking collects video data using Custom Conversion Variables (eVars) and Events. Using these variables to collect video data lets you integrate video data with other reporting data on your reports (Legacy pev3 video data was collected using a custom variable that prevented integration). This change enhances your ability to analyze the impact of video on other aspects of your on-line marketing strategy.

You can implement integrated video tracking on version 14 in preparation for version 15 migration. Integrated video tracking can be implemented on version 14 or later, and is required by version 15.

The following list explains how integrated video tracking is affected as you migrate from version 14 to 15.

If you remain on version 14:

- You can use existing version 14 video implementation for existing version 14 reports.
- You can upgrade video implementation before moving to version 15. (To upgrade, contact your Account Manager.)
- You will not have access to any of the new version 15 features, video or otherwise.

If you migrate to version 15

- To track/view any new video data, you must re-implement video module.
- You can view previous video data by logging into version 14, but after the upgrade to v.15, no video data will be processed using version 14 platform.
- No integration of version 14 video data with new data collected in version 15.

Custom solutions created by Consulting Group

- Custom solutions might not require immediate re-implementation if moving to version 15, but re-implementation is required for all new features.

Migrating for Web Analysts

For a Web analyst, migrating to integrated video tracking is similar to defining a new implementation. Complete the instructions defined in *Measuring Video for Web Analysts* and then fill out the *Video Implementation Worksheet* to give to your developer. You developer can use the information in this worksheet to migrate your existing implementation.

Flash, Silverlight, and JavaScript Migration Guide

Complete the steps in the following list to migrate your video tracking solution to the new integrated solution. After you complete these migration steps, you can recompile and test the solution. Sample implementations are available in these *Measuring Video for Developers* sections:

- *ActionScript Sample Code*
- *JavaScript Sample Code*
- *Silverlight Sample Code*

You can reference the following sections as you migrate:

- *How Video Measurement Works*

- *Media Module Methods*
- *Media Module Variables*
- *Measuring Additional Metrics using Media.monitor*

1. *Update the AppMeasurement Libraries or Media Module*
2. *Map Conversion Variables and Events*
3. *Configure Milestones*
4. *Update Method Calls*

Update the AppMeasurement Libraries or Media Module

Download the latest AppMeasurement libraries from Code Manager to replace your existing libraries. For specific instructions, see the section for your player in *Measuring Video for Developers*.

Map Conversion Variables and Events

If the Web Analytics team filled out the *Video Implementation Worksheet*, you might have a list similar to the following:

- 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`

Complete the following to map conversion variables and events:

1. Add the `s.Media.trackUsingContextData` variable and set it to true:

```
s.Media.trackUsingContextData = true;
```

2. Add the `s.Media.contextDataMapping` variable and map the parameters it contains with the SiteCatalyst variables and events you selected:

```
s.Media.trackUsingContextData = true
s.Media.contextDataMapping = {
  "a.media.name":"eVar2,prop2",
  "a.media.segment":"eVar3",
  "a.contentType":"eVar1",
  "a.media.timePlayed":"event3",
  "a.media.view":"event1",
  "a.media.segmentView":"event2",
  "a.media.complete":"event7"
};
```

Configure Milestones

If the web analytics team filled out the *Video Implementation Worksheet*, you might have a list similar to the following:

Number of seconds between measurement calls (increments of 5): `30`

Milestones to track (as a % of video length): `25%, 50%, 75%`

Divide each video into segments based on: _____

Review *Video Metrics* then complete the following to define the collection interval:

1. (Optional) To track using seconds, add the following:
```
s.Media.trackSeconds = 30;
```
2. (Optional) To track using milestones, add additional milestone events to the milestones parameter of your contextDataMapping variable:
```
s.Media.trackUsingContextData = true
s.Media.contextDataMapping = {
  "a.media.name":"eVar2,prop2",
  "a.media.segment":"eVar3",
  "a.contentType":"eVar1",
  "a.media.timePlayed":"event3",
  "a.media.view":"event1",
  "a.media.segmentView":"event2",
  "a.media.complete":"event7",
  "a.media.milestones":{
     25: "event4",
     50: "event5",
     75:"event6"
     }
};
s.Media.trackMilestones = "25,50,75";
```
   This example measures an event for the 25%, 50%, and 75% milestones. Each milestone you track must also be specified in `trackMilestones`.

   You can also track using offset milestones instead:
```
"a.media.milestones":{
     30: "event4", // 30 seconds from start of video
     60: "event5", // 60 seconds from start of video
     120:"event6"  // 120 seconds from start of video
     }
};
s.Media.trackOffsetMilestones = "30,60,120";
```
   This example measures an event 30, 60, and 120 seconds from the start of the video. Each offset milestone you track must also be specified in `trackOffsetMilestones`.
3. You can use `segmentByMilestones` to have the media module create segments automatically based on your milestones.
```
SegmentByMilestones = true;
```
   If you do not enable `segmentByMilestones` to define segments, you must use a manual implementation (not `autoTrack`) and send in the segment details with `Media.play`. If the web analytics team has defined segments to track, you can define milestones that correspond to each segment then enable `segmentByMilestones`.

Update Method Calls

If you are using `media.AutoTrack` and you do not have a custom `Media.monitor` method, you do not need to update any methods.

1. Review calls to `Media.monitor`. Compare the variables you were previously sending using `Media.monitor` with the variables you mapped in `Media.contextDataMapping`. You might be able to reduce or eliminate calls to this method if the video variables you were previously reporting are mapped in `Media.contextDataMapping`.
2. (manual tracking only) If you are manually tracking video, you can optionally update each invocation of `Media.play` to report video segment data.
   See *Media Module Methods*.

OSMF Migration Guide

Complete the steps in the following list to migrate your video tracking solution to the new integrated solution. After you complete these migration steps, you can recompile and test the solution. Sample implementations are available in these sections:

- *Dynamic Implementation*
- *Custom Dynamic Implementation*
- *Static Implementation*

You can reference the following sections as you migrate:

- *How Video Measurement Works*
- *Media Module Methods*
- *Media Module Variables*
- *Measuring Additional Metrics using Media.monitor*

Update the OSMF AppMeasurement Libraries

Download the latest AppMeasurement libraries from Code Manager to replace your existing libraries. For specific instructions, see *Download the Media Module for OSMF*.

Update the XML Configuration File

You need to add the following to update your XML configuration for integrated video tracking:

- `trackUsingContextData` and `contextDataMapping`
- `trackMilestones`

See the following section for details and an example.

XML Configuration File

When using a dynamic OSMF implementation, you can use an XML config file to bind variables to OSMF metadata. AppMeasurement uses the following variable binding syntax:

```
<variable>{media.player.metadata(namespace,key)}</variable>
```

variable: The name of the variable you wish to set (for example, eVar6).

namespace: (Optional) The OSMF metadata namespace you want to use. If you do not specify a namespace, the AppMeasurement OSMF plug-in uses the first matching key it locates in any namespace. When looking for keys, the plug-in looks first at MediaElement metadata, then at MediaResource metadata.

key: The specific metadata value you want to use.

The following section contains a sample XML configuration file.

- The `trackSeconds` and `milestone` sections are optional. See *Video Metrics*.

If the Web Analytics team filled out the *Video Implementation Worksheet*, you might have a list similar to the following:

- 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`

Map these variables to the appropriate `a.media` variable in the `contextDataMapping` section.

```
<config>
<account>myrsid</account>
<debugTracking>true</debugTracking>
<visitorNamespace>corp1</visitorNamespace>
<trackingServer>corp1.d1.sc.omtrdc.net</trackingServer>
<Media>
    <autoTrack>true</autoTrack>
    <trackMilestones>25,50,75</trackMilestones>
    <trackVars>events,eVar1,eVar2,eVar3,prop2</trackVars>
    <trackEvents>event1,event2,event3,event4,event5,event6,event7</trackEvents>
    <segmentByMilestones>true</segmentByMilestones>
    <trackUsingContextData>true</trackUsingContextData>
    <contextDataMapping>
            <a.media.name>eVar2,prop2</a.media.name>
            <a.media.segment>eVar3</a.media.segment>
            <a.contentType>eVar1</a.contentType>
            <a.media.timePlayed>event3</a.media.timePlayed>
            <a.media.view>event1</a.media.view>
            <a.media.segmentView>event2</a.media.segmentView>
            <a.media.complete>event7</a.media.complete>
            <a.media.milestones>
                <item name="25">event4</item>
                <item name="50">event5</item>
                <item name="75">event6</item>
            </a.media.milestones>
    </contextDataMapping>
</Media>
</config>
```