CONVERSANT
The power of personal.

MOJO RICH MEDIA

BANNER AD DEVELOPMENT – CONVERSANT FLASH FRAMEWORK

TABLE OF CONTENTS

INTRODUCTION

The Conversant Flash Framework will streamline the process of building Rich Media Flash banners for delivery through the Conversant Rich Media platform. Conversant maintains a number of Rich Media templates to support Flash banners and the Framework integrates your Flash assets with the Conversant templates. The Framework supports various 'Rich' features including expand/collapse, video, click tags, dynamic URLs, and tracking user interactions.

This document contains step by step instructions and recommendations for building Rich Media Flash ads with the Conversant Flash Framework. The following documentation includes instructions and guidelines for:

- Installing and implementing the Conversant Flash Framework
- Accessing the Mojo Class
- Building expandable banners (single-asset and double-asset)
- Creating banners with video
- Tracking interaction metrics
- Best practices

Screenshots and code samples are included for all critical aspects of banner development to Conversant specifications and integration with the Conversant Flash Framework. Code samples include ActionScript 2 and ActionScript 3 except for the Video modules which only support ActionScript 3.

NOTE

Individual publishers may issue additional specifications and limitations. ***Be sure to consider all publisher/site specs before ad development begins***.

A library of sample files is available as reference material for the examples contained in this document. Each section contains a reference to the relevant sample file included in the Conversant Flash Framework package.

Download the Conversant Flash Components and sample files from the following location:

http://www.mojorichmedia.com/download-framework

All code examples in the this document are illustrated in ActionScript 3 and ActionScript 2 – *except the **Video elements are supported in ActionScript 3 only***. Likewise, the sample files include examples in ActionScript 3 and ActionScript 2, except for the video components which only include ActionScript 3.

BEST PRACTICE

We update the Conversant Flash Framework and specs from time to time**. Prior to starting each new project, please download and re-install the Conversant Components** to ensure you are working from the most current code and guidelines.

THE CONVERSANT FLASH FRAMEWORK

INSTALLING THE CONVERSANT FLASH FRAMEWORK

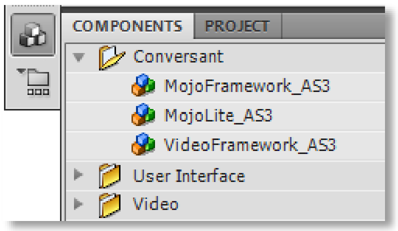
The Conversant Flash Framework is distributed as an MXP file and contains components for ActionScript 2 and ActionScript 3. Add the components to your existing Adobe Flash Professional installation by following these steps:

1. Download the Conversant Flash Framework Package ZIP file and extract the contents. http://www.mojorichmedia.com/download-framework
2. Double-click on the *MojoActionscriptFramework.mxp* file. This will launch the Adobe Extension Manager which will install the Conversant Framework components into your Flash application files.

NOTE
If you don't have the Adobe Extension Manager installed on your machine you can download it here: http://www.adobe.com/exchange/em_download.

3. If Flash is currently running on your machine, you will need to close the application and re-launch it before the Framework components are available. Alternatively, you can select 'Reload Components' from the Components panel menu.
4. In your Flash project file (.FLA), open the *Components* panel (*Window* > *Components*). You should now see the Conversant folder available. Click the triangle to the left of the folder to expand the module.

The Conversant Components module displayed in the Flash Components panel



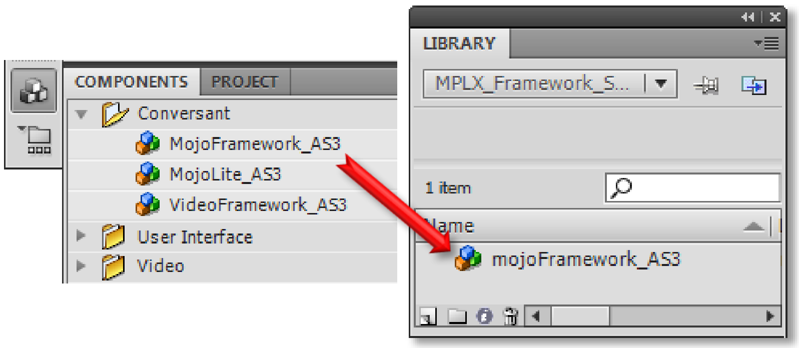
5. In your Flash project (.fla), select the first frame on the main timeline and either

   A. Double-click the mojoFramework component to add it to your project, or

   B. Open the Library panel (*Window* > *Library*) and drag the mojoFramework component from the Components window into the Library window.

Either of these methods will install the mojoFramework into your Flash project.

Dragging the mojoFramework component into the Library.

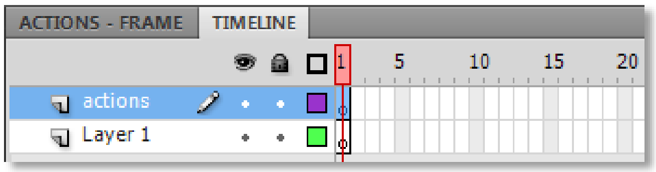


NOTE

You will see <u>either</u> the *mojoFramework_AS2* or *mojoFramework_AS3* component displayed in the *Components* pane depending on which version of ActionScript the .FLA is using. Both versions are included in the Conversant Components module. Flash automatically displays the correct component based on your ActionScript settings defined in the Flash tab of the Publish Settings dialog box.

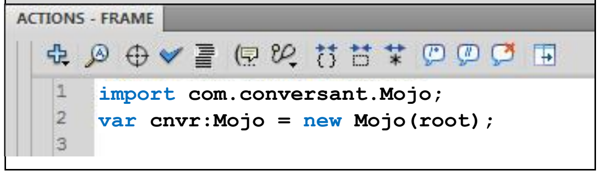
6. With the framework now added to your project, you will need to create an instance of the Mojo class.

   A. Create an *actions* layer on the main timeline.



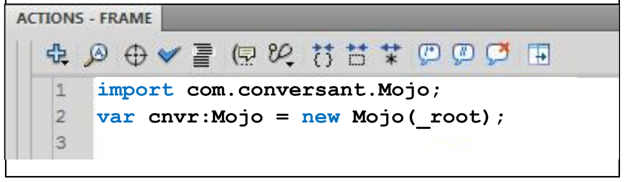
B. Enter the following ActionScript on the first frame of your Actions layer:

AS3



```
import com.conversant.Mojo;
var cnvr:Mojo = new Mojo(root);
```

AS2



```
import com.conversant.Mojo;
var cnvr:Mojo = new Mojo(_root);
```

7. Save your FLA file.

8. Now that you have created an instance of the Mojo class, you can call any of the methods of the Mojo class referenced in this document.

CREATING CLICK TAGS WITH THE MOJO CLASS

The Mojo class provides a number of pre-defined methods for performing the common functions associated with rich media creative. These include handling click tags, passing a dynamic URL, and expand/collapse functions. The Conversant Flash Framework has a separate class for Video which is detailed in section *IV – Creating Banners with Video.*

Be sure to apply the Mojo class methods to the appropriate objects/events within your Flash banner.

CLICK TAGS FOR SINGLE CLICK-THRU CREATIVE

SAMPLE FILES

AS3	clicktag/cnvr_as3_clickTag_300x250.fla
AS2	clicktag/cnvr_as2_clickTag_300x250.fla

mojo_click()

This method will call the Conversant clickTAG which will redirect the user to the landing page as defined for the creative in Conversant Adserver.

AS3

```
clickBtn.addEventListener(MouseEvent.CLICK, cnvr.mojo_click);
```

AS2

```
clickBtn.onRelease = function(){cnvr.mojo_click()};
```

CLICK TAGS FOR MULTIPLE CLICK-THRU CREATIVE

SAMPLE FILES

AS3	clicktag/cnvr_as3_multipleClickTags_300x250.fla
AS2	clicktag/cnvr_as2_multipleClickTags_300x250.fla

mojo_click(ckVal:int=null)

For creative with multiple click-thru destinations, an optional numeric parameter is used to designate and track specific click-thru URLs via the Conversant Adserver. The value passed in this optional argument will be used to construct an additional parameter which is passed on the click (*i.e. – &ck=1, &ck=2, &ck=3…*). Conversant Adserver references this parameter against a series of rules configured in the system in order to redirect to the corresponding landing page. Each unique click-thru URL must use a unique *ckVal* value.

AS3

```
clickBtn.addEventListener(MouseEvent.CLICK, Click1);
function Click1(e:MouseEvent):void { cnvr.mojo_click(1)};
otherClickBtn.addEventListener(MouseEvent.CLICK, Click2);
function Click2(e:MouseEvent):void { cnvr.mojo_click(2)};
```

AS2

```
clickBtn.onRelease = function(){ cnvr.mojo_click(1)};
otherClickBtn.onRelease = function(){ cnvr.mojo_click(2)};
```

PASSING A DYNAMIC URL ON CLICK-THRU

mojo_click(ckVal:int=null, mpre:String=null)

The optional second argument of the mojo_click method is used to designate a specific click-thru URL that will override the click-thru URL defined in Conversant Adserver. Adserver will still track the click but the final destination page will be taken from the mpre argument of this method. *The mpre parameter of the mojo_click method should be used only in situations where the final destination URL cannot be pre-defined. Otherwise the URL should be defined in* Conversant *Adserver.*

For example, in the banner ad below, the zip code is entered by the user and appended to the base URL which is passed into the mojo_click method. This argument can be used in conjunction with a ckVal argument to track multiple click-thru URLs within a single banner (see above for discussion of *Click Tags for Multiple Click-thru Creative*). When using the mpre parameter, the ckVal cannot be "null." In the sample below, the ckVal is set to 1.

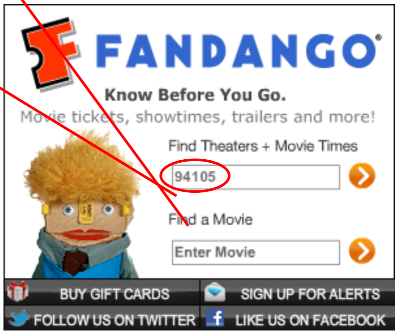
AS3

```
yourButtonName.addEventListener(MouseEvent.CLICK, zipClick);
function zipClick(e:MouseEvent):void {
     var zipCode = zipField.text;
     cnvr.mojo_click(1, "http://www.fandango.com/theaters/" + zipCode)
};
```

AS2

```
yourButtonName.onRelease = function(){
     var zipCode = zipField.text;
     cnvr.mojo_click(1, "http://www.fandango.com/theaters/" + zipCode)
};
```

Example of resulting redirect URL: http://www.fandango.com/theaters/94105



WARNING
Do NOT encode the destination URL. The mojo_click function will encode the final URL when it appends it to the actual clickTAG.

EXPANDABLE BANNER ADS

Expandable banner ads initially load on the page as standard sized banners. Upon a user-initiated action (such as a mouseover or click) the ad will expand to show one or more panels with additional messaging. Expandable banners are advantageous when the advertiser's message cannot be conveyed effectively within the confines of a static banner and where the availability of additional messaging space has a measurable impact on performance. Expandable ads can also be set to expand and collapse automatically without user interaction.

There are two methods of the Conversant Flash Framework that control the expansion and collapse of expandable banner ads; they are *mojo_show* and *mojo_hide* respectively. These methods are available in single-asset as well as double-asset expandable banners.

mojo_show()

This method initiates an ad expansion. Associate this method with events like *MouseEvent.CLICK* or *MouseEvent.ROLL_OVER* using the following construct:

AS3

```
expandBtn.addEventListener(MouseEvent.CLICK, cnvr.mojo_show);
```

AS2

```
expandBtn.onRelease = function(){ cnvr.mojo_show()};
```

mojo_hide()

This method initiates an ad collapse. Associate this method with events like MouseEvent.CLICK or MouseEvent.ROLL_OUT using the following construct:

AS3

```
collapseBtn.addEventListener(MouseEvent.CLICK, cnvr.mojo_hide);
```

AS2

```
collapseBtn.onRelease = function(){ cnvr.mojo_hide()};
```

The Flash file(s) are uploaded into Conversant Adserver and plugged into the Conversant Rich Media template which serves as the host HTML/JavaScript wrapper for the Flash file(s). This Conversant Rich Media implementation team manages this process. The Conversant Flash Framework references several Flash vars from the HTML host wrapper. The wrapper also contains various JavaScript functions that control the display of the Flash object on the page. On an expand interaction, the Conversant Flash Framework communicates with the HTML/JavaScript wrapper which swaps the SWF file to display the expanded panel and adjust the dimensions (and position if necessary) of the containing structure. Upon retraction, the reverse takes place, displaying the initial banner once again.

See the next sections for details on building expandable banners.

PRODUCING DOUBLE-ASSET EXPANDABLE FLASH CREATIVE

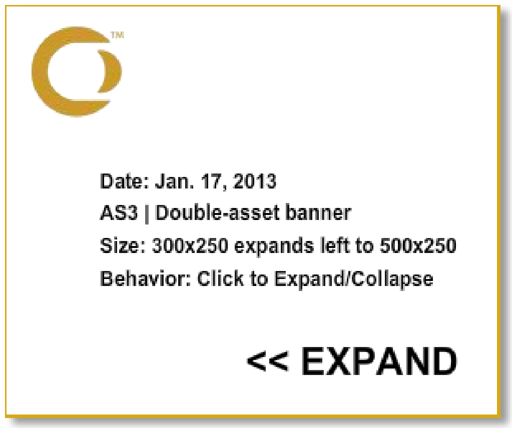
This section provides instruction for building a double-asset expandable banner using the Conversant Flash Framework and its methods. Code samples are included for both ActionScript 3 and 2.

SAMPLE FILES

AS3	expandables/cnvr_as3_doubleAssetExpand_300x250_banner.fla
	expandables/cnvr_as3_doubleAssetExpand_500x250_panel.fla
AS2	expandables/cnvr_as2_doubleAssetExpand_300x250_banner.fla
	expandables/cnvr_as2_doubleAssetExpand_500x250_panel.fla

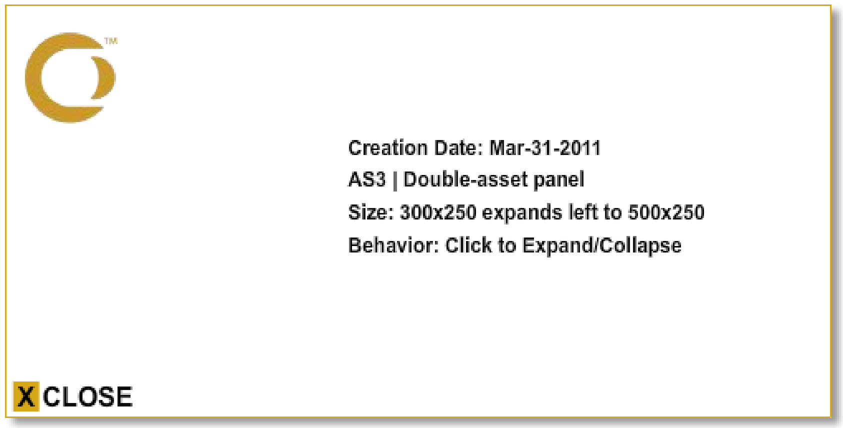
For double-asset production, you will produce two Flash files; one for the initial banner display and one for the expanded panel display. The sample below is a 300x250 banner that expands left to a 500x250 panel.

The first Flash file is the initial 300x250 banner.



Date: Jan. 17, 2013
AS3 | Double-asset banner
Size: 300x250 expands left to 500x250
Behavior: Click to Expand/Collapse

The second Flash file is the 500x250 panel.

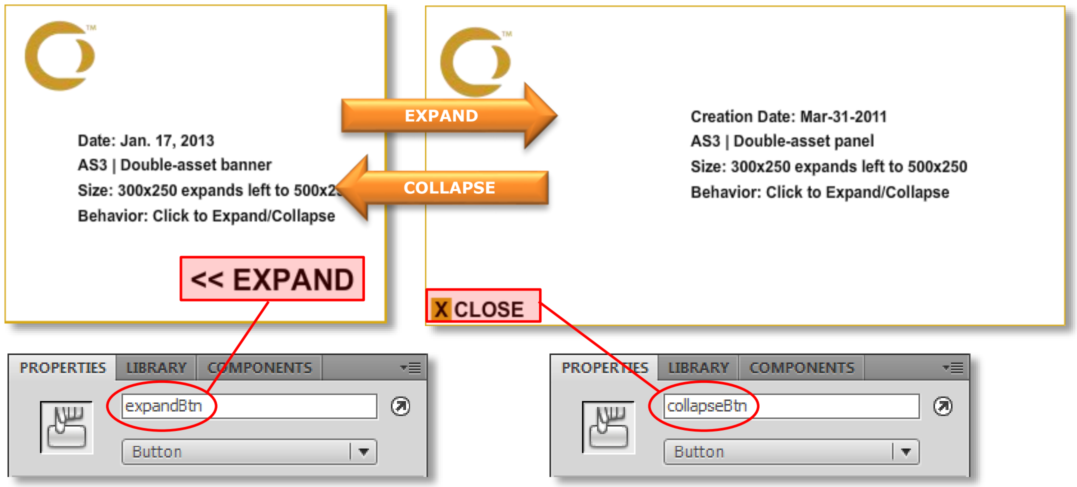


Creation Date: Mar-31-2011
AS3 | Double-asset panel
Size: 300x250 expands left to 500x250
Behavior: Click to Expand/Collapse

Both files get uploaded to the Conversant Rich Media template when the creative is submitted to Conversant Adserver. The methods in the Conversant Flash Framework communicate with the JavaScript functions in the Conversant Rich Media HTML template to display the correct file as necessary and facilitate interaction. Each Flash file contains a reference to the *mojo* class and has a dedicated button to trigger the appropriate method on click (*mojo_show* and *mojo_hide*).

The expandable ad will initially display the 300x250 pixel banner and upon click of the *Expand* button it will display the 500x250 pixel panel in place of the original banner. The *Close* button will in turn hide the expanded panel and display the initial banner once again. The instance names of the buttons in the example below are *expandBtn* and *collapseBtn*.

A double-asset banner and its expand/collapse buttons.

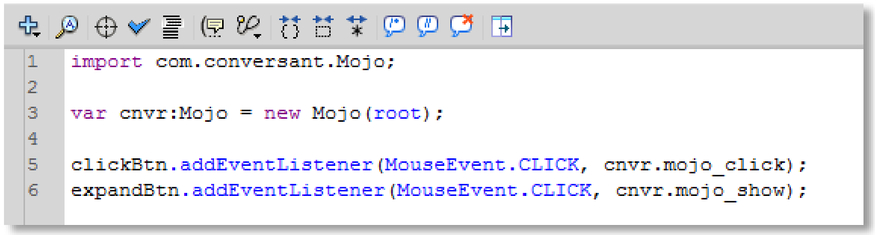

Date: Jan. 17, 2013
AS3 | Double-asset banner
Size: 300x250 expands left to 500x250
Behavior: Click to Expand/Collapse
<< EXPAND
EXPAND
COLLAPSE
Creation Date: Mar-31-2011
AS3 | Double-asset panel
Size: 300x250 expands left to 500x250
Behavior: Click to Expand/Collapse
X CLOSE
PROPERTIES
LIBRARY
COMPONENTS
expandBtn
Button
collapseBtn

The ActionScript of the 300x250 banner file does the following:

- Imports and defines the Mojo object from the Conversant Flash Framework (Lines 1-3)
- Sets the event listeners to call the appropriate methods from the Conversant Flash Framework for the Click button (Line 5) and the Expand button (Line 6)

The ActionScript of the 500x250 panel file does the following:

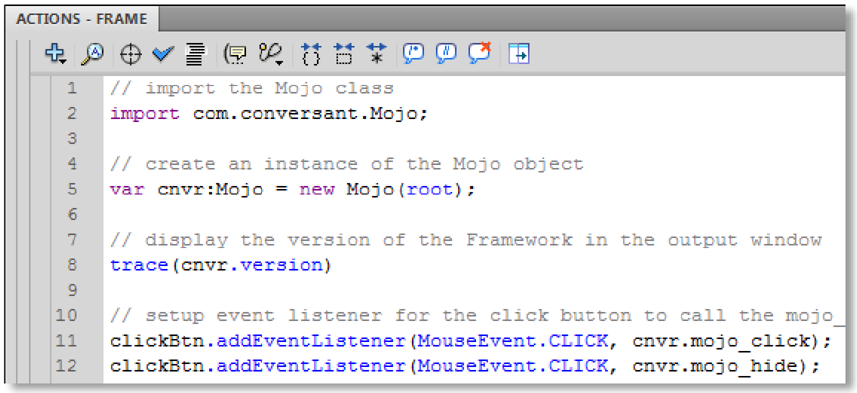
ActionScript from AS3 sample 300x250 banner.



```
1  import com.conversant.Mojo;
2  
3  var cnvr:Mojo = new Mojo(root);
4  
5  clickBtn.addEventListener(MouseEvent.CLICK, cnvr.mojo_click);
6  expandBtn.addEventListener(MouseEvent.CLICK, cnvr.mojo_show);
```

- Imports and defines the Mojo object from the Conversant Flash Framework (Lines 1-5)
- Sets the event listener for the Click button (Line 11-12).
- Sets the event listener for the Collapse button (Line 9).

ActionScript from AS3 sample 500x250 panel.



```
1   // import the Mojo class
2   import com.conversant.Mojo;
3
4   // create an instance of the Mojo object
5   var cnvr:Mojo = new Mojo(root);
6
7   // display the version of the Framework in the output window
8   trace(cnvr.version)
9
10  // setup event listener for the click button to call the mojo_
11  clickBtn.addEventListener(MouseEvent.CLICK, cnvr.mojo_click);
12  clickBtn.addEventListener(MouseEvent.CLICK, cnvr.mojo_hide);
```

NOTE

The cnvr.mojo_click method additionally causes the banner to return to its collapsed state by calling the cnvr.mojo_hide method.

COPY & PASTE ACTIONSCRIPT CODE FOR DOUBLE-ASSET EXPANDABLES

This is the minimum code needed to build a double-asset expandable banner using the Conversant Flash Framework. You can copy and paste into your .FLAs and modify as needed. This code assumes you name your buttons as referenced (*clickBtn*, *expandBtn*, *collapseBtn*).

Initial Banner

AS3

```
import com.conversant.Mojo;
var cnvr:Mojo = new Mojo(root);

clickBtn.addEventListener(MouseEvent.CLICK, cnvr.mojo_click);

expandBtn.addEventListener(MouseEvent.CLICK, cnvr.mojo_show);
```

AS2

```
import com.conversant.Mojo;
var cnvr:Mojo = new Mojo(root);

clickBtn.onRelease = function(){ cnvr.mojo_click()};

expandBtn.onRelease = function(){ cnvr.mojo_show()};
```

Expanded Panel

AS3

```
import com.conversant.Mojo;
var cnvr:Mojo = new Mojo(root);

clickBtn.addEventListener(MouseEvent.CLICK, clickBtnClick);
function clickBtnClick(e:MouseEvent):void{
     cnvr.mojo_click();
     cnvr.mojo_hide();
};
collapseBtn.addEventListener(MouseEvent.CLICK, cnvr.mojo_hide);
```

AS2

```
import com.conversant.Mojo;
var cnvr:Mojo = new Mojo(root);

clickBtn.onRelease = clickBtnClick;
function clickBtnClick() {
     cnvr.mojo_click();
     cnvr.mojo_hide();
};

collapseBtn.onRelease = function(){ cnvr.mojo_hide()};
```

PRODUCING SINGLE-ASSET EXPANDABLE FLASH CREATIVE

This section provides instruction for building single-asset expandable banners using the Conversant Flash Framework and its methods. Code samples are included for both ActionScript 3 and 2.

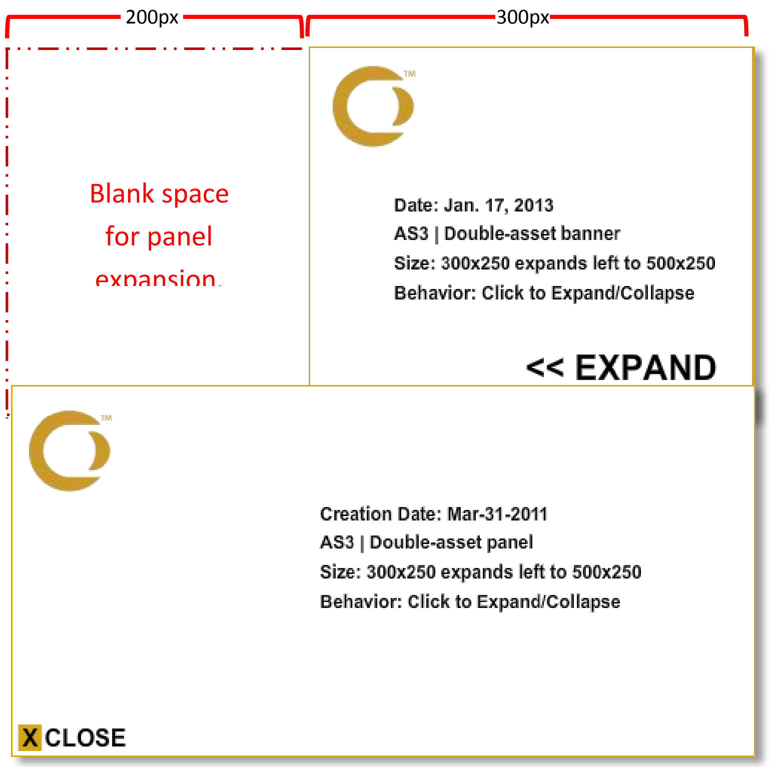
SAMPLE FILES

AS3	expandables/cnvr_as3_singleAssetExpand_300x250_500x250.fla
AS2	expandables/cnvr_as2_singleAssetExpand_300x250_500x250.fla

For single-asset production, you will produce a single SWF file which contains all content, animation, and coding for both the initial banner and expanded panel. The overall dimension of a single-asset creative will be that of the expanded panel. In this sample, the expandable ad will initially display as a 300x250 pixel banner and upon user interaction expand to display the 500x250 pixel panel. The single Flash asset will have an overall dimension of 500x250 pixels. The initial frame(s) will only display the left or right 300 pixels, depending on the direction it will expand (in this case right to left), with the remaining 200 pixels displaying blank space (which will be transparent in the final execution). On the expand interaction, the timeline will advance and the creative content will expand into the full 500 pixel panel to display the entire content of the ad. Upon retraction, the creative content will collapse back to the initial 300 pixel width banner.

Sample single-asset expandable displaying initial 300x250 pixels.

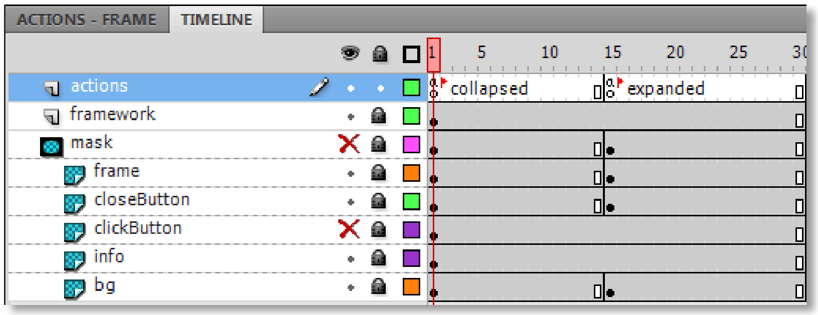
Sample single-asset expandable displaying expanded 500x250 pixels.


200px
300px
Blank space for panel expansion.
TM
Date: Jan. 17, 2013
AS3 | Double-asset banner
Size: 300x250 expands left to 500x250
Behavior: Click to Expand/Collapse
<< EXPAND
TM
Creation Date: Mar-31-2011
AS3 | Double-asset panel
Size: 300x250 expands left to 500x250
Behavior: Click to Expand/Collapse
X CLOSE

The creative will be loaded into the Conversant host HTML wrapper (configured and managed by Conversant). The host wrapper will pass in the necessary Flash vars used by the Conversant Flash Framework. The wrapper also contains various JavaScript functions that control the display of the wrapper and Flash object on the page. The Conversant Flash Framework references the relevant JavaScript functions from its methods. *See section 1 above for details on the Conversant Flash Framework.*

The timeline of this single-asset banner has two content frames (labeled 'collapsed' and 'expanded'). The 'collapsed' frame displays the 300x250 banner with the left 200 pixels masked. The 'expanded' frame displays the entire 500x250 panel.

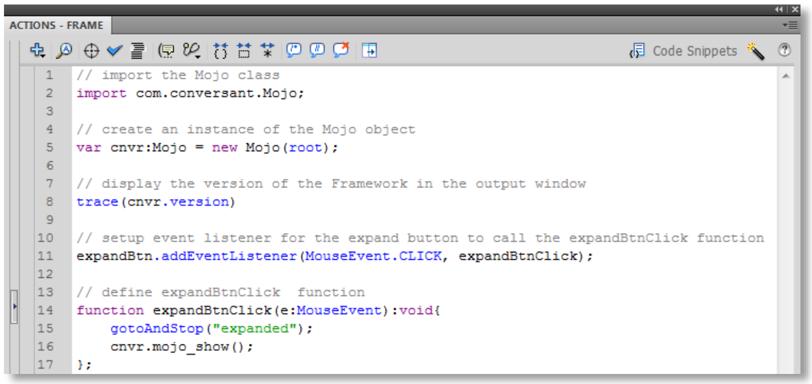
Sample timeline from a single-asset expandable.



The ActionScript on the first frame ('collapsed') of the main timeline does the following:

- Imports and defines the Mojo object from the Conversant Flash Framework (Lines 2-5)
- Sets the event listener for the Expand button (Line 11) and defines the function to trigger the expand (Lines 14-17)

ActionScript from frame 1 ('collapsed') of AS3 sample file.

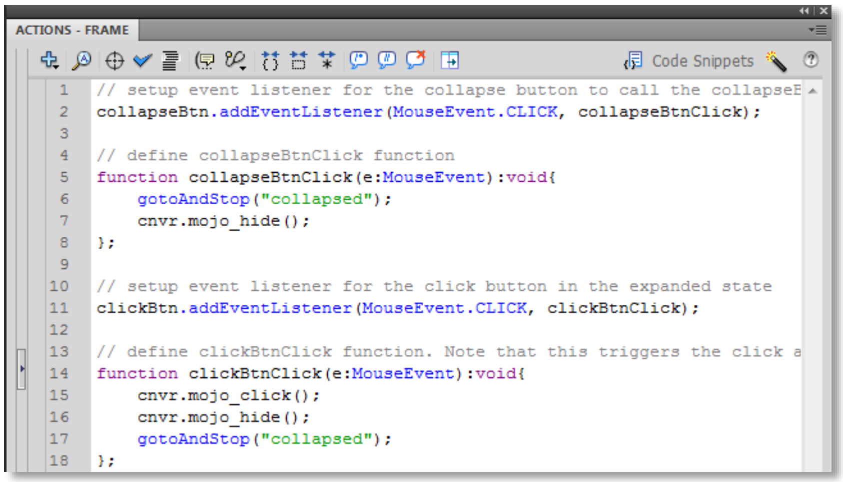


```
1   // import the Mojo class
2   import com.conversant.Mojo;
3
4   // create an instance of the Mojo object
5   var cnvr:Mojo = new Mojo(root);
6
7   // display the version of the Framework in the output window
8   trace(cnvr.version)
9
10  // setup event listener for the expand button to call the expandBtnClick function
11  expandBtn.addEventListener(MouseEvent.CLICK, expandBtnClick);
12
13  // define expandBtnClick  function
14  function expandBtnClick(e:MouseEvent):void{
15      gotoAndStop("expanded");
16      cnvr.mojo_show();
17  };
```

The ActionScript on the 15th frame ('expanded') of the main timeline does the following:

- Sets the event listener for the Collapse button (Line 2) and defines the function to trigger the collapse (Lines 5-8)
- Sets the event listener for the Click button (Line 11) and defines the function to be called on click (Lines 14-18).

ActionScript from frame 15 ('expanded') of AS3 sample file.



```
1   // setup event listener for the collapse button to call the collapseE
2   collapseBtn.addEventListener(MouseEvent.CLICK, collapseBtnClick);
3
4   // define collapseBtnClick function
5   function collapseBtnClick(e:MouseEvent):void{
6       gotoAndStop("collapsed");
7       cnvr.mojo_hide();
8   };
9
10  // setup event listener for the click button in the expanded state
11  clickBtn.addEventListener(MouseEvent.CLICK, clickBtnClick);
12
13  // define clickBtnClick function. Note that this triggers the click a
14  function clickBtnClick(e:MouseEvent):void{
15      cnvr.mojo_click();
16      cnvr.mojo_hide();
17      gotoAndStop("collapsed");
18  };
```

COPY & PASTE ACTIONSCRIPT CODE FOR SINGLE-ASSET EXPANDABLES

This is the minimum code needed to build a single-asset expandable banner using the Conversant Flash Framework. You can copy and paste into your .FLA and modify as needed to integrate with your existing objects and code. This sample code assumes you name your buttons as referenced (*clickBtn*, *expandBtn*, *collapseBtn*) and label your frames the same (*expanded*, *collapsed*). Of course you can modify your instance names as necessary.

AS3

FRAME 1 ('collapsed')

```
// import the Mojo class
import com.conversant.Mojo;

// create an instance of the Mojo object
var cnvr:Mojo = new Mojo(root);

// setup event listener for the expand button to call the expandBtnClick
function
expandBtn.addEventListener(MouseEvent.CLICK, expandBtnClick);

// define expandBtnClick  function
function expandBtnClick(e:MouseEvent):void{
     gotoAndStop("expanded");
     cnvr.mojo_show();
};
```

FRAME 15 ('expanded')

```
collapseBtn.addEventListener(MouseEvent.CLICK, collapseBtnClick);
function collapseBtnClick(e:MouseEvent):void{
     gotoAndStop("collapsed");
     cnvr.mojo_hide();
};

clickBtn.addEventListener(MouseEvent.CLICK, clickBtnClick);
function clickBtnClick(e:MouseEvent):void{
     cnvr.mojo_click();
     gotoAndStop("collapsed");
     cnvr.mojo_hide();
};
```

BANNERS WITH VIDEO

The Conversant Flash Framework video player components are supported in ActionScript 3 only; however the video metrics referenced in section 4.4 are supported in ActionScript 2 as well.

You can add a video player to the stage in a number of ways. Use the MojoVideo class to add a player that fully integrates with the Conversant Flash Framework and tracks all of the requisite video interaction metrics. If you choose a different method to create and control the video object, you can still implement the tracking metrics by following the instructions in section *4.4 – Tracking Video Metrics without using the MojoVideo class*. This still requires use of the Mojo object but not the MojoVideo object.

You can add video to any Conversant Rich Media banner ad and it will display as a progressive download. The preferred format for video file delivery is .FLV, however the following MPEG-4 video formats are supported as well: .MP4, .M4A, .MOV, .MP4V, .3GP, and .3G2. All MPEG-4 video formats must utilize H.264 encoding when deploying to Flash Player 9 update 3 and above.

If you require streaming video, please contact your Conversant Account Manager for special requirements.

PRODUCING A SIMPLE VIDEO BANNER USING THE MOJOVIDEO CLASS

This section provides instruction for building a simple Flash banner with video using the MojoVideo class of the Conversant Flash Framework.

SAMPLE FILE

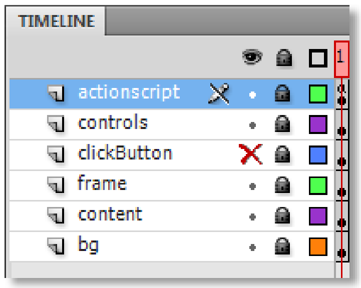
AS3 video/cnvr_as3_video_player_SIMPLE_300x250.fla /.swf

Begin by importing the Mojo class and the MojoVideo class from the Conversant Flash Framework. *See section 2.1 above for details on the Mojo class.* Create an instance of the Mojo object and the MojoVideo object. Set the width and height of the new video object to match those of the actual video that it will display (in the example below, the width and height of the video are 300px and 168px respectively).

```
import com.conversant.Mojo;  // import Mojo class
import com.conversant.MojoVideo;   // import MojoVideo class
var cnvr:Mojo = new Mojo(root);    // create Mojo object
var _mojoVid:MojoVideo = new MojoVideo(300,168);  // create MojoVideo
                                                     object (W,H)
```

With the video object (_mojoVid) defined, now add it to the stage using the *addChildAt()* method and position it as necessary.

```
addChildAt(_mojoVid, 1); // add video object to the stage
_mojoVid.x = 0;          // set x coordinate position of video object
_mojoVid.y = 82;         // set y coordinate position of video object
```



The first argument of the *addChildAt()* method is used to identify the video object to be added. The second argument identifies the layer where the video will show up. In this case, the *_mojoVid* object will display above the bottom layer of the main timeline because the second argument has a value of *1*. In reference to the timeline at the right, a value of *1* as the second argument will add the video object between the 'bg' and 'content' layers. A value of *2* would add it between the 'content' and 'frame' layers. A value of *3* would add it between the 'frame' and 'clickButton' layers, and so forth.

Next you will begin to apply methods of the MojoVideo class beginning with the *loadAndPlay()* method. Likewise, you could call the *loadAndStop()* method, depending on whether you want the video to begin playing as soon as it is loaded or not.

loadAndPlay(flvName:String, ploadPrcnt:Number=0, custMetrics:String=""),
loadAndStop(flvName:String, custMetrics:String=""),

Call the *loadAndPlay()* method of the *_mojoVid* object.

```
_mojoVid.loadAndPlay("video_clip.flv", ploadPrcnt:Number=0,
custMetrics:String="");
```

The *loadAndPlay* method of the MojoVideo class calls the video file to be played and begins playback. The *flvName* argument accepts a path to a local (or remote) video file. This reference is used for local playback only and will not be used when the ad is delivered in production. The video file itself must be included in the package of assets delivered to Conversant to be uploaded into the production environment in Conversant Adserver. The Conversant Flash Framework is configured to ensure that the video uploaded with the Flash creative into Conversant Adserver is called when the banner is displayed. The *flvName* argument is for the convenience of local testing only.

The *ploadPrcnt* argument is a numerical value that indicates the percentage of the video file buffered before the file begins to play.

The *custMetrics* argument is used to differentiate metrics for multiple videos within a single creative. When each video is loaded using *loadAndPlay* or *loadAndStop*, enter a unique identifier for that video. E.g., loadAndPlay(video_clip.flv, 1, "Testimonial1") will result in the video metric Testimonial1-Video-Plays, Testimonial1-Video-Pause, Testimonial1-Video-Stop, etc. The *custMetrics* argument is only necessary for creative with multiple videos.

COPY & PASTE ACTIONSCRIPT CODE FOR SETTING UP A SIMPLE VIDEO BANNER

```
import com.conversant.Mojo;
import com.conversant.MojoVideo;
import com.conversant.MojoEvent;

var cnvr:Mojo = new Mojo(root);
var _mojoVid:MojoVideo = new MojoVideo(300,168); // set your actual
dimensions

addChildAt(_mojoVid, 1);
_mojoVid.x = 0;
_mojoVid.y = 82;
_mojoVid.loadAndPlay("video_clip.flv", 1); // reference your video clip
```

USING ADDITIONAL FEATURES OF THE MOJOVIDEO CLASS

This section builds off of the simple banner defined in section 4.1 above and provides instruction for incorporating additional controls and features of the MojoVideo class into the video banner. These include button controls, looping, and buffering.

SAMPLE FILE

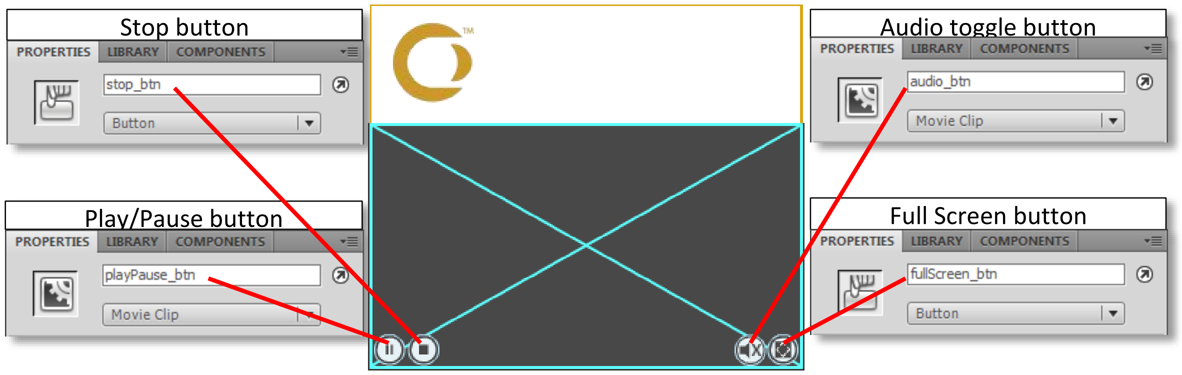
AS3 video/cnvr_as3_video_player_FULL_300x250.fla

We will begin applying features of the MojoVideo class to add functionality to our video banner.

See section 4.3 below for a complete reference of the properties, methods, and events available through the MojoVideo class.

You will need to add button objects to the stage to use as interface controls for the video. To do this, you can build your own video control bar or you can use the video controls available in the sample file for this section. Be sure to name each instance of your control buttons and reference the methods of the Conversant Flash Framework as detailed below.

300x250 banner ad with 300x168 video and video controls.



Once your button instances are set up you can begin by assigning them to the button properties of the MojoVideo class.

```
_mojoVid.playPauseBtn = playPause_btn;   // assign play/pause button
_mojoVid.stopBtn = stop_btn; // assign stop button
_mojoVid.fullscreenToggleBtn = fullScreen_btn; // assign full screen
                toggle button
_mojoVid.muteToggleBtn = audio_btn;       // assign audio button
```

The button properties have inherent listeners set up to control the video appropriately as well as track the interaction metrics through the Conversant Rich Media template which supports the Flash banner in production. You may want to set up additional listeners and functions to control the buttons themselves or other aspects of the creative. For example, the *playPause_btn* movie clip toggles video playback and pause. Also, it will display the appropriate icon based on which state the video is in (playing = show pause icon | paused = show play icon).

```
playPause_btn.addEventListener(MouseEvent.CLICK, toggleIcon);
function toggleIcon(e:MouseEvent):void {
     if (_mojoVid.isPlaying) { playPause_btn.gotoAndStop(1); }
     else { playPause_btn.gotoAndStop(2); }
};
```

A similar function may also be set up for the audio button

```
audio_btn.addEventListener(MouseEvent.CLICK, setAudio);
function setAudio(e):void {
     if (_mojoVid.mute == true) { audio_btn.gotoAndStop(2); }
     else { audio_btn.gotoAndStop(1); };
};
```

If you would like to buffer the video before displaying the stream, you can use the *buffer* property.

```
_mojoVid.buffer = .5;
```

In this case *'.5'* is the amount of buffering time you wish to apply (in seconds).

You can determine if the video will loop or not and if so, how many times.

```
_mojoVid.loopPlayback = 2;
```

In this case *'2'* is the number of times the video will loop after initial playback. A value of 2 means the video will play a total of 3 times (initial playback + 2 loops). If you do not want the video to loop, omit this property from your code. The default is no looping.

Additionally, you can control the size and position of the replay icon. By default a replay icon will appear when the video playback and looping complete and when the banner is clicked during video playback. You can set its size and position with the following:

```
_mojoVid.replayIconScaling = .8;   // scales size of icon (1 = 100%,
   0 = none)
_mojoVid.replayIconOffset = [0,-20];     // X,Y coordinates of icon
   relative to target
```

NOTE
For full screen mode to display properly, the video needs to have an x and y position of 0. For easiest use, put the video into a container, set the video to have a x and y position to 0 within the container, then place the container on the stage where you would like the video to appear.

You can also remove the video from the stage when playback looping completes or on any event you choose.

```
_mojoVid.clearIt();
```

The MojoVideo class provides options for a wide array of video functionality. *See the next section for a complete reference of all of the MojoVideo properties, methods and events.*

INDEX OF BUTTONS, PROPERTIES, METHODS AND EVENTS OF THE MOJOVIDEO OBJECT

The following tables list all properties, methods, and events available with the MojoVideo class. Use the appropriate options to control the video as needed. You can reference these in your ActionScript as you would any standard Flash properties and methods. All of the MojoVideo properties and methods will automatically fire the appropriate metrics to track in Conversant Reports. You do not need to use all of the properties or methods of the Conversant Framework in any given project. Choose the features that you need.

Using the Button Properties is the easiest way to incorporate the basic video interaction with the least amount of coding. The subsequent properties and methods give you more specific control as needed but may be redundant. For example, if you assign the *playPauseBtn* property to your button Play/Pause button, it will inherently attach the *togglePlayback* method to the button. Thereby you bypass the need to assign any of the play or pause properties or methods.

BUTTON PROPERTIES OF THE MOJOVIDEO OBJECT

The following properties provide the simplest way to assign functionality to your video control buttons. Simply reference the appropriate control button instance name as the value of the property.

Name	Syntax	Description

Property Name	*`Full syntax of property`*
	`Example`
		Description of property

clickTagBtn	`clickTagBtn:DisplayObject`
	`_mojoVid.clickTagBtn = click_btn;`
		[write only] Identifies the button assigned to the clickTAG. Reference the Click button by instance name. Calls the *mojo_click* method of the Mojo class.
		Note: the video will pause after a clickTAG fires.

playBtn	`playBtn:DisplayObject`
	`_mojoVid.playBtn = play_btn;`
		[write only] Identifies the button used to play the video. Reference the Play button DisplayObject by instance name. Calls *playIt()* method when the Play button is clicked.

pauseBtn	`pauseBtn:DisplayObject`
	`_mojoVid.pauseBtn = pause_btn;`
		[write only] Identifies the button used to pause the video playback. Reference the Pause button DisplayObject by instance name. Calls *pauseIt()* method when the Pause button is clicked.

playPauseBtn	`playPauseBtn:DisplayObject`
	`_mojoVid.playPauseBtn = playPause_btn;`
		[write only] Identifies the button used to toggle play/pause of the video. Reference the Play/Pause button DisplayObject by instance name. Calls *togglePlayback()* method when the Play/Pause button is clicked.
		Typically used instead of separate play and pause buttons.

stopBtn	*`stopBtn:DisplayObject`*
	`_mojoVid.stopBtn = stop_btn;`
		[write only] Identifies the button used to stop the video playback. Reference the Stop button DisplayObject by instance name. Calls *stopIt()* method when the Stop button is clicked.

muteToggleBtn	`muteToggleBtn:DisplayObject`
	`_mojoVid.muteToggleBtn = audio_btn;`
		[write only] Identifies the button used to toggle the video audio on/off. Reference the Audio/Mute button DisplayObject by instance name. Calls *toggleMute()* method when the Audio/Mute button is clicked.

fullscreenToggleBtn	`fullscreenToggleBtn:DisplayObject`
	`_mojoVid.fullscreenToggleBtn = fullScreen_btn;`
		[write only] Identifies the button used to toggle full screen mode. Reference the Full Screen button DisplayObject by instance name. Calls *toggleFullScreen()* method when the Full Screen button is clicked.

OTHER PROPERTIES OF THE MOJOVIDEO OBJECT

Name	Syntax	Description

Property Name	*`Full syntax of property`*
	`Example`
		Description of property

videoDuration	*`videoDuration:Number`*
	`_mojoVid.videoDuration;`
		[read only] Returns the total length of the loaded video file

buffer	*`buffer:Number`*
	`_mojoVid.buffer = .5;`
		[read/write] Specifies how long to buffer video before starting to display the stream. The default value is 0.1 (1/10 of a second).

disableMetrics	*`disableMetrics:Boolean`*
	`_mojoVid.disableMetrics = true;`
		[read/write] Disables all video metrics. *Predominately used for testing ads on non-Conversant web servers. Prevents the mpcrid error.*

loadPrgrss	*`loadPrgrss:Number`*
	`_mojoVid.loadPrgrss;`
		[read only] Returns the percentage of the video file that has been downloaded

isPlaying	*`isPlaying:Boolean`*
	`_mojoVid.isPlaying;`
		[read only] Returns true if the video is playing, otherwise returns false.

playPrgrss	*`playPrgrss:Number`*
	`_mojoVid.playPrgrss;`
	`_mojoVid.playPrgrss = 50;`
		[read/write] Accepts/returns a percent of the video played

playTime	*`playTime:Number`*
	`_mojoVid.playTime = 10;`
		[read/write] A playback time in the video defined in seconds

loopPlayback	*`loopPlayback:Number`*
	`_mojoVid.loopPlayback = 2;`
		[read/write] Set the number of times the video should loop after initial playback. Default is zero.

volumeLevel	*`volumeLevel:Number`*
	`_mojoVid.volumeLevel = .5;`
		[read/write] Sets/returns values 0 through 1. Default value is 1.

mute	*`mute:Boolean`*
	`_mojoVid.mute = true;`
		[read/write] Mute/unmute video or detect if video has been muted

fullScreen	*`fullScreen:Boolean`*
	`_mojoVid.fullScreen = false;`
		[read/write] Enter/exit full screen mode or detect if currently in full screen mode

replayIcon	*`replayIcon:String`*
	`_mojoVid.replayIcon = replayBtn;`
		[write only] Defines a DisplayObject (by instance name) to be used to override the default replay button. The instance name is referenced as the value of the replayIcon property (i.e. – replayBtn).

replayIconScaling	*`replayIconScaling:Number`* `_mojoVid.replayIconScaling = .5;`	*[read/write]* Scale the size of the video Replay button. Default value is 1 (100%). Set value for desired size (i.e. – .5 = 50%, 2 = 200%, etc.).
replayIconOffset	*`replayIconOffset:Array[x,y]`* `_mojoVid.replayIconOffset = [0,0];`	*[read/write]* Defines X and Y positioning for the Replay button offset, specified in px. Default values are zero.
lastFrameStop	*`lastFrameStop:Boolean`* `_mojoVid.lastFrameStop = true;`	*[read/write]* If set to true, video will pause on the last frame without displaying a replay button.

METHODS OF THE MOJOVIDEO OBJECT

Name	Syntax	Description
Method Name	*Full syntax of method* `Example`	Description of method

loadAndPlay	*loadAndPlay(flvName:String, ploadPrcnt:Number=0):void*
	`_mojoVid.loadAndPlay("my_video.flv");`
	`_mojoVid.loadAndPlay("my_video.flv", 30);`
		Start playing a specified video file. Use the flvName argument to set the path of the video file (in "quotes").
		OPTIONAL: use the *ploadPrcnt* argument to determine the percentage of the video to download before playback begins (i.e. – 30 = 30%).

loadAndStop	*loadAndStop(flvName:String):void* `_mojoVid.loadAndStop("my_video.flv");`	Specifies a video file to load and play later. Use the flvName argument to set the path of the video file (in "quotes").
playIt	*playIt(e:Event=null):void* `_mojoVid.playIt();`	Starts playing a loaded video or resumes play if video is paused
pauseIt	*pauseIt(e:Event=null):void* `_mojoVid.pauseIt();`	Pauses video playback
togglePlayback	*togglePlayback(e:Event=null):void* `_mojoVid.togglePlayback();`	Play or pause video depending on current playback state
stopIt	*stopIt(e:Event=null):void* `_mojoVid.stopIt();`	Stops video playback and resets player to beginning of video
toggleMute	*toggleMute(e:Event=null):void* `_mojoVid.toggleMute();`	Mute/unmute video depending on current state
toggleFullScreen	*toggleFullScreen(e:Event=null):void* `_mojoVid.toggleFullScreen();`	Enter/exit full screen mode depending on current state

clearIt	*clearIt(e:Event=null):void*
	`_mojoVid.clearIt();`
		Deactivate and remove video player from stage.
		Currently in development.
		For a temporary workaround, use:
		_mojoVid.stopIt();
		_mojoVid.visible = false;

EVENTS OF THE MOJOEVENT OBJECT

Name	Syntax	Description
Event Name	`Full syntax of event`	Description of event
VIDEO_CLEAR	`MojoEvent.VIDEO_CLEAR`	Event object's data property indicates if video object is cleared
VIDEO_PLAYBACK_TOGGLE	`MojoEvent.VIDEO_PLAYBACK_TOGGLE`	Event object's data property indicates if play mode was entered (true) or exited (false)
VIDEO_PRELOADING	`MojoEvent.VIDEO_PRELOADING`	Event object's data property indicates if video is preloading
VIDEO_PLAYING	`MojoEvent.VIDEO_PLAYING`	Event object's data property indicates if video is playing
VIDEO_PAUSED	`MojoEvent.VIDEO_PAUSED`	Event object's data property indicates if video has been paused
VIDEO_STOPPED	`MojoEvent.VIDEO_STOPPED`	Event object's data property indicates if video has been stopped
FULLSCREEN_TOGGLE	`MojoEvent.FULLSCREEN_TOGGLE`	Event object's data property indicates if fullscreen mode was entered (true) or exited (false)
MUTE_TOGGLE	`MojoEvent.MUTE_TOGGLE`	Event object's data property indicates if mute mode was entered (true) or exited (false)
COMPLETED	`MojoEvent.COMPLETED`	Event object's data property indicates if video has played to completion
REPLAY	`MojoEvent.REPLAY`	Event occurs when user clicks on the replay button when video completes.
CONTINUE_PLAYING	`MojoEvent.CONTINUE_PLAYING`	Event occurs when user clicks on the replay button after returning from a clickTAG
VIDEO_DURATION	`MojoEvent.VIDEO_DURATION`	Event object's data property indicates the number of seconds long
MUTE	`MojoEvent.MUTE`	Event object's data property indicates if video is muted (true)
UNMUTE	`MojoEvent.UNMUTE`	Event object's data property indicates if video is un-muted (true)

TRACKING VIDEO METRICS WITHOUT USING THE MOJOVID CLASS

The MojoVideo object tracks the appropriate interaction metrics automatically through the properties and methods detailed in sections 4.1 - 4.3 above. However if you choose to use your own video player rather than the MojoVideo object, you can still track interaction metrics using the *video_event* method. To use the *video_event* method, you will first need to add the Mojo class as detailed in section 2.1 above.

video_event(evtName:String, timeVal:Number=-1)

For banners containing video (whether expandable or non-expandable), the *video_event* method is used to track various events during user interaction with the video player. For example, clicking on the play button would typically trigger the *Video-Plays* metrics. These tracking calls are subsequent to the innate functionality of the interactions (i.e. – clicking the play button will need to make the video play in addition to calling the *video_event* function).

There are two arguments available to the *video_event* method. The first argument, *evtName*, is required and passes the actual name of the metric being recorded. For example, passing a value of ”Video-Plays" as the *evtName* argument will result in recording a metric called ”Video-Plays" in Conversant Reports.

AS3

```
playBtn.addEventListener(MouseEvent.CLICK, videoPlay);
function videoPlay(e:MouseEvent):void {
     cnvr.video_event("Video-Plays");
};
```

AS2

```
playBtn.onRelease = function() { cnvr.video_event("Video-Plays") };
```

The *video_event* method will track any of the standard video metrics. Simply replace the name of the metric to be tracked within the parentheses and quotation marks following *cnvr.video_event*. These calls to the *video_event* method will need to be incorporated in your ActionScript on the appropriate events. The Conversant video metrics are limited the following standard event names.

Video-Plays	cnvr.video_event("Video-Plays");
Video-Pauses	cnvr.video_event("Video-Pauses");
Video-Stops	cnvr.video_event("Video-Stops");
Video-25pct	cnvr.video_event("Video-25pct");

Video-50pct	cnvr.video_event("Video-50pct");
Video-75pct	cnvr.video_event("Video-75pct");
Video-Completes	cnvr.video_event("Video-Completes");
Video-Replays	cnvr.video_event("Video-Replays");
Video-Rewinds	cnvr.video_event("Video-Rewinds");
Video-Mutes	cnvr.video_event("Video-Mutes");
Video-Unmutes	cnvr.video_event("Video-Unmutes");
Video-Full-Screen	cnvr.video_event("Video-Full-Screen");

Learn more about tracking custom events in the section 'Tracking Custom Events.'

The second argument of the *mojo_event* method is *timeVal*. This is optional and is used only when a specific value needs to be attached to the metric. Without the *timeVal* argument, the default value is always 1 (i.e. – *Video-Plays=1).* For example, if you wanted to track the amount of time in seconds that the video has played when the user clicks the Stop button, you may set a variable that records the time stamp of the video and appends that to the *video_method* as the *timeVal* argument.

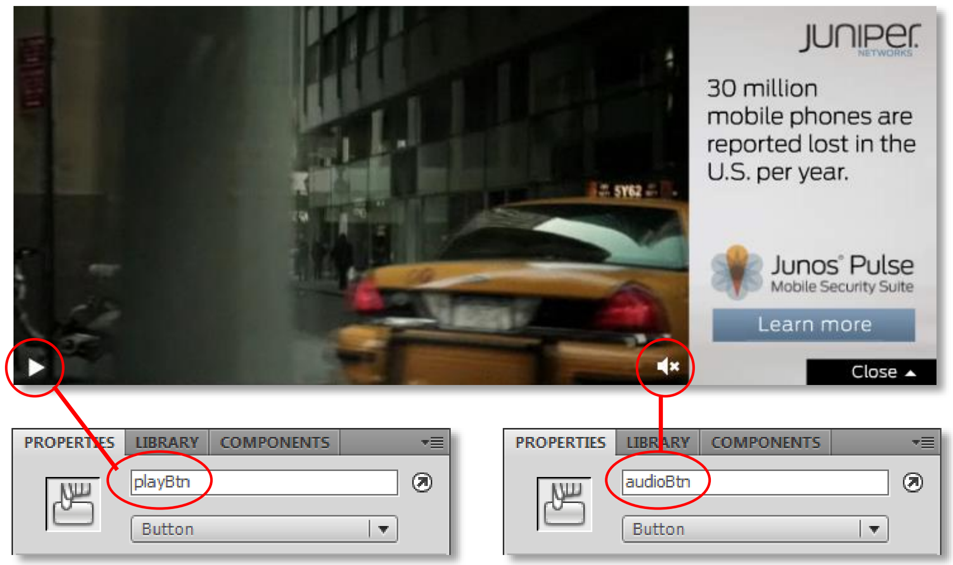
AS3

```
stopBtn.addEventListener(MouseEvent.CLICK, trackTime);
function trackTime(e:MouseEvent):void {
     var vidTime = [return video timestamp];
     cnvr.video_event("Time-Played", vidTime);
};
```

AS2

```
stopBtn.onRelease = function() {
     var vidTime = [return video timestamp];
     cnvr.video_event("Time-Played", vidTime);
};
```

In the example below, the Play button also serves as a Pause button when toggled. Likewise, the Audio button toggles the audio on and off. In a situation like this, the function's code checks the current state of the button, performs the appropriate action, and fires the correct metric.



```
var playBtnState:String = 'play'; //video is paused initially
if (playBtnState == 'play') {
//[PLAY VIDEO];
cnvr.video_event("Video-Plays");
playBtnState = 'pause';
}
else if (playBtnState == 'pause') {
//[PAUSE VIDEO];
cnvr.video_event("Video-Pauses");
playBtnState = 'play';
};
```

TRACKING EVENTS

STANDARD EVENTS

The Conversant Flash Framework automatically generates a number of standard metrics to track user interactions through Conversant Reports. This is an overview of the standard metrics that Conversant tracks and what they represent.

	Name	Description
Common Events	**Display-Time**	The amount of time the entire ad is displayed
	Mouse-In	Indicates a user moving their mouse onto the ad
	Mouse-Out	Indicates a user moving their mouse off of the ad
	Interactions	Indicates any user interaction with the ad (includes expand, collapse, video interactions)
	Hover-Time	The amount of time the user's mouse was positioned over the ad
	JavaScript Failure Impressions	Indicates user does not have JavaScript activated in their browser
	JavaScript Failure Clicks	Indicates a click by a user after JavaScript Failure Impression
	Flash Failure Impressions	Indicates the user did not have a sufficient Flash Player installed to display the ad
	Flash Failure Clicks	Indicates a click by a user after Flash Failure Impression
Expandable Events	**Expand-Time**	The amount of time the ad is displayed in the expanded state
	Expand[n]	The number of times the ad is expanded where [n] counts multiple expands on a single impression. For example, the first expand on an impression is tracked as Expand1. If the user closes the banner and expands it again during the same impression, Expand2 is tracked. This allows you to determine the number of users who expanded once, twice, three times, etc.
	Manual-Closes	Indicates user-initiated collapse
	Auto-Expand	Indicates auto-initiated expand
	Auto-Closes	Indicates auto-initiated collapses
Video Events	**Video-Plays**	Indicates the video was played (includes play being resumed after pause)
	Video-Pauses	Indicates the video was paused during playback
	Video-Stops	Indicates the video was stopped during playback
	Video-25pct	Indicates 25% of video has played
	Video-50pct	Indicates 50% of video has played
	Video-75pct	Indicates 75% of video has played
	Video-Completes	Indicates the video played to completion
	Video-Replays	Indicates the video was replayed after completion
	Video-Rewinds	Indicates video rewind button was clicked
	Video-Mutes	Indicates the video was muted
	Video-Unmutes	Indicates the video was un-muted
	Video-Full-Screen	Indicates the user activated the video full screen feature

TRACKING CUSTOM EVENTS

SAMPLE FILES

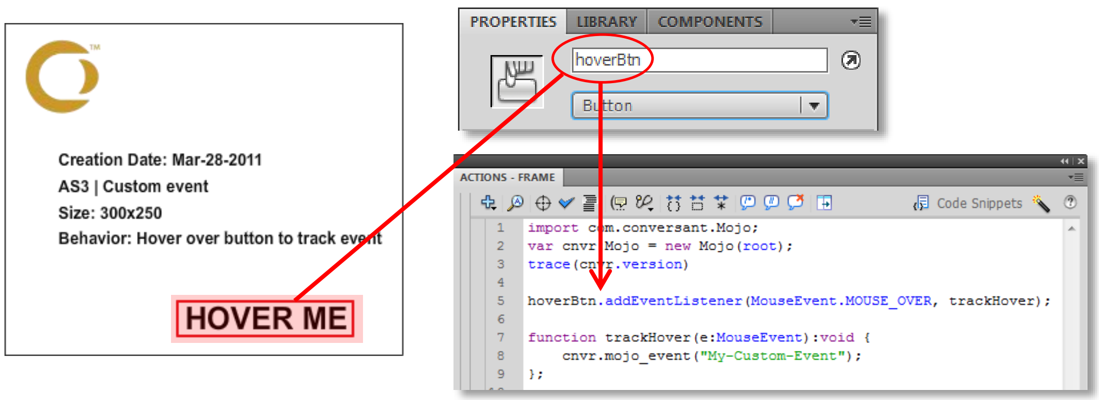
AS3	events/cnvr_as3_customEvent_300x250.fla
AS2	events/cnvr_as2_customEvent_300x250.fla

mojo_event(rmTag:String, rmVal:String)

The *mojo_event* method tracks custom event metrics that are not included with Conversant's standard metric reporting. You can use this to define a unique metric name and have it appear in Conversant Reports as a new entry.

You can track custom metrics either through the Conversant Flash Framework or by using a direct call to the *mojo_event* JavaScript function in the Conversant rich media template.

TRACKING CUSTOM EVENTS USING THE CONVERSANT FLASH FRAMEWORK



Creation Date: Mar-28-2011
AS3 | Custom event
Size: 300x250
Behavior: Hover over button to track event

```
import com.conversant.Mojo;
var cnvr:Mojo = new Mojo(root);
trace(cnvr.version)

hoverBtn.addEventListener(MouseEvent.MOUSE_OVER, trackHover);

function trackHover(e:MouseEvent):void {
    cnvr.mojo_event("My-Custom-Event");
};
```

In the above example, moving your mouse over the "Hover Me" button triggers an event called "expandBtn_hover" tracked by Conversant Adserver. If the "*expandBtn_hover*" metric does not already exist in Conversant Adserver, the system will automatically create it on the first request. All subsequent requests will aggregate. The event name "*expandBtn_hover*" is one example of custom names you can create. You can define any name you like for your custom metrics. Keep in mind that the name you indicate is the name that will show up in Conversant reports to track that metric. Custom event names cannot include spaces or special characters (i.e. - $, &, /, %, etc.). Stick with letters, numbers, and the underscore_.

AS3

```
hoverBtn.addEventListener(MouseEvent.MOUSE_OVER, customEvent);
function customEvent(e:MouseEvent):void {
     cnvr.mojo_event("expandBtn_hover");
};
```

AS2

```
hoverBtn.onRollOver = function() {
     cnvr.mojo_event("expandBtn_hover")
};
```

The default value assigned to any metric is 1 (i.e. – *expandBtn_hover=1*). The value aggregates for each metric in Conversant Reports. You can pass a specific value (other than 1) if you choose when you call the *mojo_event* function by adding the desired value as a second argument. This must be a numeric value only. For example if you wanted to track the amount of time (in seconds) a user spend hovering over the "Hover Me" button, you might track that time value in a variable and then pass it along with the "expandBtn_hover" event:

```
var hoverTime = 30;

cnvr.mojo_event("expandBtn_hover", hoverTime);
```

This would result in a value of *expandBtn_hover=30* to be recorded by Conversant Adserver.

NOTE
The second argument is optional and will default to a value of 1 if not included.

TRACKING TIMED EVENTS

SAMPLE FILE

AS3	events/CNVR_as3_timedEvent_300x250.fla /
AS2	events/CNVR_as2_timedEvent_300x250.fla /

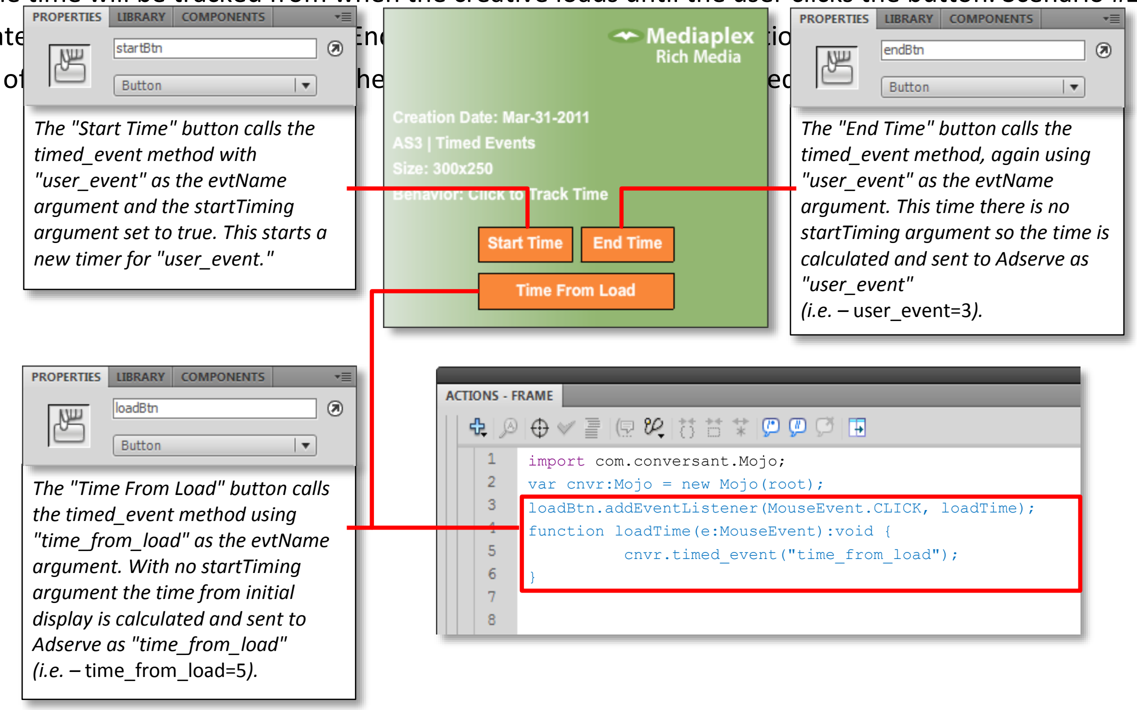
timed_event(evtName:String, startTiming:Boolean=false)

This method is used for tracking custom time metrics that are not included with the standard Conversant metric reporting. This method can be used in two ways:

1. By default the *timed_event* method tracks the number of seconds from the time the creative was loaded until calling the method. For example, if you want to know how long an ad displays before the user clicks on a certain button, call this method when the user clicks the button and the time will be tracked using the name you define in the *evtName* argument.
2. The *timed_event* method can also track the interval between two designated interactions. For example, you may want to track how long after a user hovers (event 1) over a creative before they click (event 2) on it. To do this, you would call the method on the first event (mouse over) and define the *evtName* argument. You will also include the *startTiming* argument set to *true* to begin a new timer for the given *evtName* (no tracking metric will be fired). Now call the function again on the second event (click). Use the same *evtName* but this time with no *startTiming* argument.

The example above illustrates both configurations. The "Time From Load" represents scenario #1 above where the time will be tracked from when the creative loads until the user clicks the button. Scenario #2 is illustrate

the click o



The "Start Time" button calls the timed_event method with "user_event" as the evtName argument and the startTiming argument set to true. This starts a new timer for "user_event."

The "End Time" button calls the timed_event method, again using "user_event" as the evtName argument. This time there is no startTiming argument so the time is calculated and sent to Adserve as "user_event"
(i.e. – user_event=3*).*

The "Time From Load" button calls the timed_event method using "time_from_load" as the evtName argument. With no startTiming argument the time from initial display is calculated and sent to Adserve as "time_from_load"
(i.e. – time_from_load=5*).*

```
import com.conversant.Mojo;
var cnvr:Mojo = new Mojo(root);
loadBtn.addEventListener(MouseEvent.CLICK, loadTime);
function loadTime(e:MouseEvent):void {
        cnvr.timed_event("time_from_load");
}
```

ADDITIONAL SPECS AND BEST PRACTICES

The following is a set of best practices to consider when building expandable creative assets. Although not required by Conversant per se, these are common practices often required by publishers. We strongly recommended that the following recommendations be adhered to in order to avoid some of the common problems with publisher acceptance of your ads. Individual publishers may issue additional specifications and limitations. Be sure to review and consider all publisher/site specs before ad development begins.

COLLAPSE ON CLICK

Upon click of the creative while in the expanded state, the *cnvr.mojo_click* method triggers the redirect URL to open in a new tab or window. On this event, the ad should also collapse back to the banner state on the original page. The *cnvr.mojo_hide* method handles the JavaScript portion of the collapse.

The two arguments are used to pass a parameter for multiple click-thru destinations and to pass a dynamic click-thru URL, respectively (see *Click Tags for Multiple Click-thru Creative* and *Passing a Dynamic URL on Click-thru* in section 2.2). If you are not using multiple click-thru locations or a dynamic URL, you can omit them completely.

```
cnvr.mojo_click();
```

OR (if using multiple click-thru and/or a dynamic url)

```
cnvr.mojo_click(1, "http://www.destinationURL.com);

cnvr.mojo_hide();
```

In a single-asset expandable, you will also advance the timeline to the frame displaying the collapsed banner content..

```
cnvr.mojo_click();

gotoAndStop("collapsed");
cnvr.mojo_hide();
```

The mojo_hide(); command should always come last in any list of functions.

The examples in the section on expandables above include this functionality in the 'collapse' buttons.

LOADING EXTERNAL CONTENT

SAMPLE FILES

AS3	FlashVars/CNVR_as3_mojo_aux_300x250.fla
AS2	FlashVars/CNVR_as2_mojo_aux_300x250.fla

Conversant allows you to load external content into your Flash creative. We recommend using external variables, or flashVars, to reference the file path of the loaded content. For instance, if you would like to dynamically load a graphic you can use the flashVar mojo_aux1. The file location of the graphic will be included in the Conversant environment and passed to the flash creative as part of the HTML wrapper. To use this file path in your flash, you can use the code:

AS 3

```
var newImage:String = root.loaderInfo.parameters.mojo_aux1;
```

AS 2

```
var newImage = _root.mojo_aux1;
```

This is only the first part of the code you will need to load content from a flashVar. To see the complete list of code, please consult the sample files :

SAMPLE FILES

AS3	FlashVars/ CNVR_as3_mojo_aux_300x250.fla
AS2	FlashVars/ CNVR_as2_mojo_aux_300x250.fla

If file paths or external text are contained in an XML file, you can load the XML file through a flashVar, and then use ActionScript to parse the XML file. To see coding examples on how to do this, check the samples files :

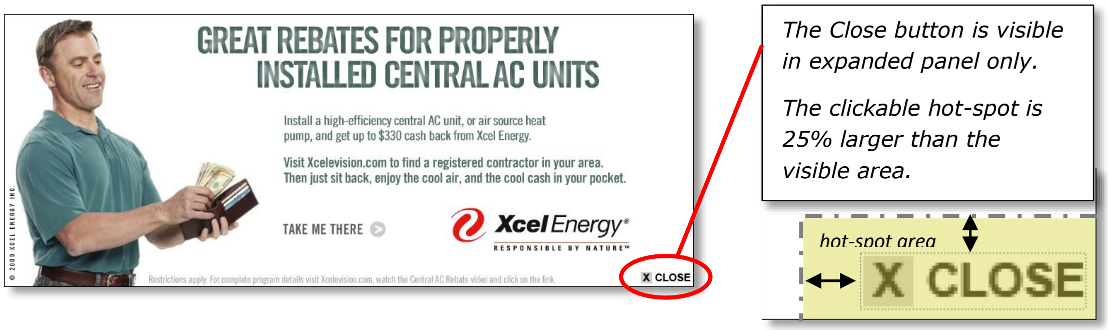
SAMPLE FILES

AS3	FlashVars/ CNVR_as3_mojo_xml_300x250.fla
AS2	FlashVars/ CNVR_as2_mojo_xml_300x250.fla

CLOSE BUTTON

Most publishers require a clearly visible close button on all expandable ads. This typically consists of an [x] and/or the word 'Close.' The actual configuration and position of the close button will vary from publisher to publisher. Here are some recommendations to help ensure your close button meets most publisher specs.

- Clicking the close button will trigger the ad to collapse back to the banner state.
- Clicking the close button will not cause the ad to redirect or open any additional windows.
- Often this button is required in addition to a mouseout-to-collapse trigger.
- The close button should not be visible on the collapsed banner.
- The hot-spot (clickable area) of the button should be 25% larger than the visible area of the button.


GREAT REBATES FOR PROPERLY INSTALLED CENTRAL AC UNITS
Install a high-efficiency central AC unit, or air source heat pump, and get up to $330 cash back from Xcel Energy.
Visit Xcelevision.com to find a registered contractor in your area. Then just sit back, enjoy the cool air, and the cool cash in your pocket.
TAKE ME THERE
Xcel Energy
RESPONSIBLE BY NATURE
Restrictions apply. For complete program details visit Xcelevision.com, watch the Central AC Rebate video and click on the link.
X CLOSE
hot-spot area
X CLOSE

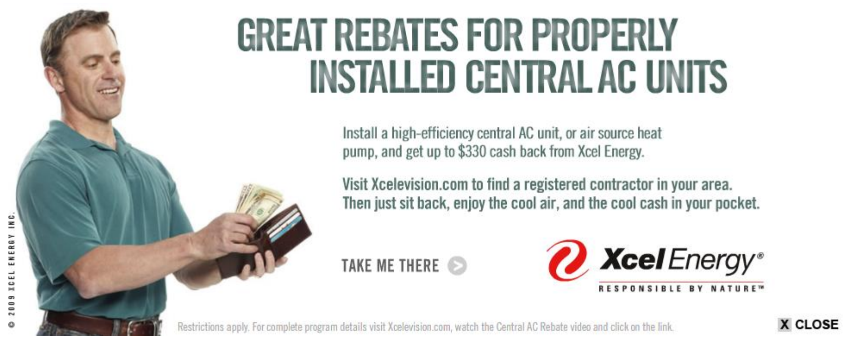
The Close button is visible in expanded panel only.

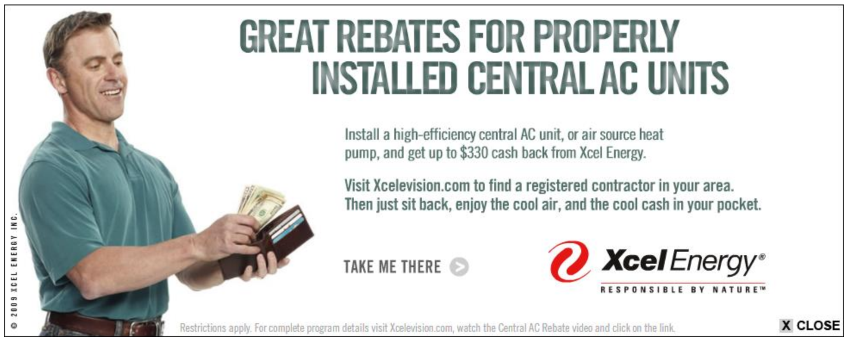
The clickable hot-spot is 25% larger than the visible area.

BORDER OUTLINE

A one pixel border should define the outer edges of the ad. This is particularly important if the ad uses a white or light background color. Without a border the ad may appear to blend in with the rest of the page. Many publishers require a visible border on ads with a light background; however this may be optional on ads with a dark background. If in doubt, it is a good idea to add the border.

The same ad with and without a visible border. Note the lack of definition in the ad without a border.


GREAT REBATES FOR PROPERLY INSTALLED CENTRAL AC UNITS
Install a high-efficiency central AC unit, or air source heat pump, and get up to $330 cash back from Xcel Energy.
Visit Xcelevision.com to find a registered contractor in your area. Then just sit back, enjoy the cool air, and the cool cash in your pocket.
TAKE ME THERE
Xcel Energy®
RESPONSIBLE BY NATURE™
Restrictions apply. For complete program details visit Xcelevision.com, watch the Central AC Rebate video and click on the link.
X CLOSE
© 2009 XCEL ENERGY INC.


GREAT REBATES FOR PROPERLY INSTALLED CENTRAL AC UNITS
Install a high-efficiency central AC unit, or air source heat pump, and get up to $330 cash back from Xcel Energy.
Visit Xcelevision.com to find a registered contractor in your area. Then just sit back, enjoy the cool air, and the cool cash in your pocket.
TAKE ME THERE
Xcel Energy®
RESPONSIBLE BY NATURE™
Restrictions apply. For complete program details visit Xcelevision.com, watch the Central AC Rebate video and click on the link.
X CLOSE
© 2009 XCEL ENERGY INC.

COLLAPSE ON MOUSEOUT

For expandable banners that expand when the user moves their mouse over the banner, publishers often require the banner to likewise collapse as soon as the user moves their mouse off of the banner (in addition to displaying a close button as detailed in section 6.2). ActionScript 3 offers the MOUSE_LEAVE event which, when attached to the stage object, works effectively for this purpose.

```
stage.addEventListener(Event.MOUSE_LEAVE, cnvr.mojo_hide);
```

NOTE
This is not available in ActionScript 2.

CODING FOR FULL SCREEN

When switching between normal and full screen mode, the video controls need to be positioned relative to the full screen size, instead of their position in normal mode. Currently the framework does not handle the repositioning of video controls between full screen and normal modes. To move the buttons to the bottom center of the screen in full screen mode, and then back to their former normal mode positions, please see the sample file.

SAMPLE FILES

AS3	video/CNVR_as3_video_player_FULL_300x250.fla

NOTE
This is not available in ActionScript 2.

DEVELOPER CHECKLIST

Submit all creative assets, once produced, to Conversant for QA and configuration in Conversant Adserver. The submission should include:

- All exported SWF file(s)
- All FLA and development files (i.e. - .as, .xfl, etc.) for QA
- Back up image(s)
- Additional assets (i.e. – .gif, .jpg, .png, .xml, .flv files to be loaded dynamically into SWF – if applicable)
- Indicate minimum Flash Player version (i.e. – 9.0)
- Click-thru URL (or URLs if ad unit contains more than one click-thru)
- List of Custom Events to be tracked by Conversant – if any

GLOSSARY

banner	A banner ad displayed in its native dimensions. For an expandable ad, the unexpanded state of the ad. i.e. – the banner portion.
panel	The expanded display portion of an expandable ad.
mojo_show	The JavaScript function in the host wrapper template that expands the <div> container. mojo_show must be called when the ad is triggered to expand.
mojo_hide	The JavaScript function in the host wrapper template that closes the <div> container. mojo_hide must be called when the ad is triggered to collapse.
mojo_event	The JavaScript function in the host wrapper used to track custom metrics.
Flash var	A variable contained in the host wrapper for a SWF file which is accessible to the ActionScript code with the SWF.
clickTAG	This Flash var passes the click tag URL for banner clicks and tracks the click events from the Conversant host wrapper template.
mojo_mpcrid	This Flash var passes a unique value to the ActionScript from the host wrapper. This variable is referenced as an argument for all JavaScript function calls.
mojo_flv	This Flash var passes the path of the external video file into the Flash movie, allowing the assignment of a video file without the need to hard-code its path into the ActionScript.

ADDITIONAL SUPPORT

For all technical questions or further guidance and support, please contact Conversant customer support:

customersupport@conversantmedia.com

1.866.417.1271