# Creating a Custom Flash Video (FLV) Player - Part 1

## By:

Flash video has arrived. You see it everywhere from ABCNews.com (http://abcnews.go.com/) to Amazon (http://www.amazon.com/) . One reason for its rapid adoption is that it enables you to deliver video that works on all browsers and platforms. No more hassling with competing formats such as Windows Media, QuickTime, and Real Player. Now you can convert your video to a single format — a Flash Video (FLV) file — and play it back on any computer that has Flash Player 6 or higher installed.

**NOTE:** Flash Player supports two Flash video formats: *streaming video* and *progressive download video*. Streaming FLVs require Flash Communication Server and Flash Player 6 or higher. Progressive FLVs can be served over standard HTTP and require Flash Player 6,0,65,0 or higher. For more information, see Macromedia's "Flash Video Learner's Guide (http://www.macromedia.com/devnet/mx/flash/articles/video_primer.html) ."

To add Flash video to your web site, you have a number of options. If you have Flash Communication Server, you can use Peldi's FLV Player (http://www.peldi.com/blog/FLVPlayer.html) . For ease of use, you can't beat the Flash Video Kit (http://www.communitymx.com/content/article.cfm?cid=79081) , a Macromedia solution that takes the drudgery out of adding progressive Flash video to your web site.

But the Flash Video Kit has its limitations. Ultimately, if you want to exercise complete control over the appearance and behavior of your Flash video player, you have to create it yourself. That's what this article is all about. A preview of the completed movie appears below (requires Flash Player 6,0,65,0 or higher):

This three-part article consists of the following sections:

- Part 1 — Creating a Flash video player using the **MediaDisplay** component
- Part 2 — Creating a Flash video player *without* components
- Part 3 — Creating a Flash video player component

In Part 1, you'll create a player that incorporates the Flash **MediaDisplay** component. You'll also learn how to modify or "skin" the player's appearance by replacing its movie clip symbols.

In Part 2, you'll create the same player using the `NetConnection` and `NetStream` classes, rather than the **MediaDisplay** component. This results in a much smaller SWF file (2K versus 58K).

In Part 3, you'll create an ActionScript class file, and associate that class with the player controls, to create a reusable Flash video player component.

**We'd like to hear from you.** Since we haven't created the Flash component yet, your feedback is important. What features would you like to see in the final extension? For example, would you like the player to auto-size? Buffer? Should it display the elapsed time and total time of the video? What about a version that uses Flash Remoting to generate a playlist? Please use the feedback form at the end of this article to submit your requests.

All three installments include the complete FLA and ActionScript source files, and enable you to edit basic settings using an external XML file (i.e., without republishing the SWF file).

## Using the Custom Flash Video Player

The starting point for this tutorial is Macromedia's Flash Video Templates (http://www.macromedia.com/devnet/mx/flash/video_templates.html) . These free templates were created by Macromedia to demonstrate how to incorporate Flash video into web sites, advertisements, and Slide-based presentations. In the case of the Sports Video Template (http://www.macromedia.com/devnet/mx/flash/articles/vidtemplate_vidshowcase.html) , the ActionScript is attached to movie clip symbols throughout the FLA. We've revised the code to adhere to ActionScript 2.0 best practices, such as strict typing, and moved the ActionScript out of the FLA and into a single, external ActionScript file. We also modified the appearance of the controls.

To use the custom Flash video player in your projects, simply edit **config.xml** in Dreamweaver or your favorite text editor:

```
<config>
  <autoPlay>1</autoPlay>
  <flvURL>videos/respighi.flv</flvURL>
  <loop>0</loop>
  <volume>20</volume>
</config>
```

This XML file is parsed by the Flash movie to define four variables:

- **autoPlay** — Play video when movie starts (0=false; 1=true)
- **flvURL** — The relative or absolute URL of the Flash video (FLV) file
- **loop** — Replay video when it finishes (0=false; 1=true)
- **volume** — The initial volume setting (a number from 0 to 100)

To get started, click **Download Support Files** at the bottom of this page and extract the zip file to a

folder on your computer. You'll find two samples Flash video files, brahms.flv and respighi.flv, in the **videos** folder. To play back the Brahms video, for example, change the `<flvURL>` node in config.xml to the following and open **flv_md.htm** in a browser:

```
<flvURL>videos/brahms.flv</flvURL>
```

To create your own FLVs, you can download a trial version of Sorenson Squeeze (http://www.sorenson.com/) , Cleaner XL (http://www.discreet.com/cleanerxl/) , or Wildform Flix (http://www.wildform.com/) .

**TIP:** If you cannot get the Flash video player to work on your web site, try browsing directly to the URL of the FLV file (e.g., http://www.yoursite.com/yourVideo.flv). If you receive a 404 File Not Found error, you (or your host) may need to define the FLV MIME type (http://livedocs.macromedia.com/flash/mx2004/main_7_2/00001107.html) (video/x-flv) on the server. If you're using IIS 6.0, consult the following Flash TechNote (http://www.macromedia.com/support/flash/ts/documents/flv_mime.htm) .

For more on creating FLVs, see "Streaming Video in Flash MX Professional 2004 (http://www.communitymx.com/abstract.cfm?cid=7574E) " by Tom Green.

## How the Player Works

If you open **flv_md.fla** in Flash MX 2004, you'll see that it consists of one frame and four layers. The MediaDisplay component is on the **video** layer. Currently, this Flash video player is designed to work with FLVs that are 320x240 pixels. If your FLV is smaller or larger, you can select the MediaDisplay component and change its dimensions using the Property inspector. You'll probably want to resize the **bg** movie clip and modify the scale and placement of the controls (play/pause button, scrubber bar, and volume slider). To change the size of the Flash document itself, choose **Modify > Document** (**Ctrl/Cmd + J**). The default size is 320x267.

### Understanding the MediaDisplay Component

The nuts and bolts of the FLV player is the Flash MediaDisplay component. This component provides an API that simplifies interacting with streaming media such as MP3s and FLVs. Table 1 lists the methods, properties, and events used by our Flash video player. For the complete `Media` class API, see **Using Components>Components Dictionary>Media components>Media class** in Flash help.

| Method | Description |
|---|---|
| pause | Pauses the playhead at its current location in the media timeline |
| play | Plays the media associated with the component instance. Accepts one parameter, **startingPoint**, that indicates the starting point (in seconds) at which the media should begin playing |
| stop | Stops the playhead and moves it to position 0, which is the beginning of the media |
| **Property** | **Description** |
| autoPlay | Determines if the component instance immediately starts to buffer and play |
| bytesLoaded | The number of bytes loaded that are available for playing (read-only) |
| bytesTotal | The number of bytes to be loaded into the component instance (read-only) |
| contentPath | A string that holds the relative or absolute URL of the media to be streamed and played |
| playheadTime | Holds the current position of the playhead (in seconds) for the media timeline that is playing |
| playing | Returns a Boolean value to indicate whether a given component instance is playing media (read-only) |
| totalTime | An integer that indicates the total length of the media (in seconds) |
| volume | An integer from 0 (minimum) to 100 (maximum) that represents the volume level |
| **Event** | **Description** |
| complete | Notification that the playhead has reached the end of the media |
| progress | Generated continuously until the media has downloaded completely |

**Table 1** *Methods, properties and events implemented by the Flash video player*

The MediaDisplay component, and its methods, properties, and events, make designing the Flash video player relatively easy. The tradeoff is that all these built-in ActionScript 2.0 classes result in a lot of weight: the completed SWF file, **flv_md.swf**, is 58 kilobytes. (In Part 2 of this article, we'll use the `NetConnection` and `NetStream` classes to bring this down to a svelte 2 kilobytes.) Because of this, you may decide to create a preloader for the player. This involves moving the existing layers to Frame 2 of the main Timeline, and placing the preloader on Frame 1. For complete instructions, see "Preloading Components in Flash MX 2004 (http://www.communitymx.com/abstract.cfm?cid=EBB61) ."

## Understanding the ActionScript

To see how the methods, properties and events of the MediaDisplay component are used in the Flash video player, open **flv_md.as** in Flash MX Professional 2004 or your favorite text editor. The ActionScript document consists of 6 main sections:

- Timeline variables
- Event listeners
- Button handlers
- Video scrubber
- Volume control
- Functions

These sections are explained in detail below.

### Timeline Variables

The Timeline variables are defined *outside* of functions, so they are accessible throughout the movie. In other words, they are *scoped* to the main Timeline.

### Event Listeners

The next section defines two listeners for events of the MediaDisplay component:

```
var flvListener:Object = new Object();


// called when Flash video playback is completed
flvListener.complete = function(evt:Object):Void{
  videoCompleted();
}


// called repeatedly while FLV download is in progress
flvListener.progress = function(evt:Object):Void{
  var bl:Number = evt.target.bytesLoaded;
  var bt:Number = evt.target.bytesTotal;
  movieScrubber.loadProgress._width =
  ➥Math.round(bl/bt * movieScrubber.clipTimeline._width);

  if(bl > 4 && bt > 4 && bl >= bt){
    trace("FLV loaded");
  }
}


videoDisplay.addEventListener("complete", flvListener);
videoDisplay.addEventListener("progress", flvListener);
```

The `complete` event is broadcast to the listener when the Flash video reaches the end of its playback. The `flvListener.complete` function calls the `videoCompleted` function, which in turn calls `MediaDisplay.stop`. The `stop` method stops playback of the video and moves the playhead to position 0.

The `progress` event is called repeatedly until the progressive FLV is completely downloaded. Each time `flvListener.progress` is called, we use the `MediaDisplay.bytesLoaded` and `MediaDisplay.bytesTotal` properties to change the width of the `loadProgress` movie clip. Once the media is downloaded, the `flvListener.progress` function is no longer called.

### Button Handlers

The button handlers merely call the `MediaDisplay.play` and `MediaDisplay.pause` methods when the corresponding buttons are clicked. The `togglePlay` function determines each button's visibility (see "Functions" below).

### Video Scrubber

This section handles user interaction with the video scrubber.

```
movieScrubber.scrubDot.onPress = function():Void{
  this.startDrag(false, 0, 0, scrubberLength-10, 0);
  scrubbing = true;
}


movieScrubber.scrubDot.onRelease = function():Void{
  this.stopDrag();
  videoDisplay.play(t);
  scrubbing = false;
}


movieScrubber.onEnterFrame = function():Void{
  togglePlay();

  if(scrubbing){
    t = (this.scrubDot._x/scrubberLength) * videoDisplay.totalTime;

    videoDisplay.play(t);
    // this pauses video until scrubber is released
    videoDisplay.pause();
  }
  else{
    var scrubFactor:Number =
videoDisplay.playheadTime/videoDisplay.totalTime;
    this.scrubDot._x = scrubRange * scrubFactor;
  }
}
```

When the scrubber handle (`scrubDot`) is pressed and released, we call the `MovieClip.startDrag` and `MovieClip.stopDrag` methods respectively. The `startDrag` method specifies a constraint rectangle so the user can only scrub within the scrubber timeline. When the scrubber handle is released, the `MediaDisplay.play` method is called and passed a `startingPoint` parameter.

The `startingPoint` parameter, `t`, is defined in the `MovieClip.onEnterFrame` function. This event handler is invoked repeatedly at the frame rate of the SWF file. If the user is scrubbing playback, we call the `MediaDisplay.play` method, followed immediately by the `MediaDisplay.pause` method. This enables us to preview the video as we scrub it. If the user is not scrubbing, we simply advance the position of the `scrubDot` movie clip during playback by changing its `_x` property.

### Volume Control

This section uses some of the same techniques as the video scrubber:

```
volumeSlider.volumeDot.onPress = function():Void{
  this.startDrag(false, 0, -4, volumeRange, -4);

  volumeSlider.onEnterFrame = function():Void{
    var v:Number = Math.round((this.volumeDot._x/volumeRange)*100);
    soundLevel = v;
    setVolume(v);
  }
}


volumeSlider.volumeDot.onRelease = function():Void{
  this.stopDrag();
  delete volumeSlider.onEnterFrame;
}


volumeSlider.speakerIcon.onRelease = function():Void{
  var muted:Boolean = videoDisplay.volume == 0;

  setVolume(muted ? soundLevel : 0);
}
```

The main difference is that the `volumeSlider.onEnterFrame` event is deleted when the user releases the `volumeDot` movie clip. The third function simply toggles the audio. If the `MediaDisplay.volume` property is 0, we know the video is muted. We then call the `setVolume` function and pass it a conditional statement: "If the audio is muted, change it back to the previous sound level. Otherwise, set the volume to 0."

### Functions

The `setVolume` function does three things: it changes the `_x` position of the `volumeDot` movie clip, it calls the `MediaDisplay.volume` method, and it calls the `MovieClip.gotoAndStop` method of the `speaker_mc` symbol. If you open this movie clip in **flv_md.fla**, you'll see that it contains four frames to represent the different states of the speaker: muted, low ), medium )), and high ))).

```
function setVolume(v:Number):Void{
  volumeSlider.volumeDot._x = Math.round((volumeRange*v)/100);
  videoDisplay.volume = v;
  var speakerLevel:Number =
  ➥Math.round(((volumeSlider.volumeDot._x/volumeRange)*100)/33)+1;
  volumeSlider.speakerIcon.speaker_mc.gotoAndStop(speakerLevel);
}
```

The `loadPreferences` function loads the XML configuration file and converts its nodes to Timeline variables.

**TIP:** For more on loading XML into Flash, see "Using the XFactorStudio XPath Classes (http://www.communitymx.com/abstract.cfm?cid=7C2AB) " and "Parsing XML in Flash (http://www.communitymx.com/abstract.cfm?cid=04A15) ."

If the XML data is successfully loaded, the `XML.onLoad` function calls `setupVideo`, which applies the

settings to the MediaDisplay component.

```
function loadPreferences():Void{
  var myXML:XML = new XML();
  myXML.ignoreWhite = true;
  myXML.onLoad = function(success:Boolean){
    if(success){
      trace("XML loaded.");

      var prefNode:XMLNode = this.firstChild.firstChild;

      while(prefNode != null){
        trace(prefNode.nodeName + ": " + prefNode.firstChild.nodeValue);
        // convert nodes to Timeline variables
        owner[prefNode.nodeName] = prefNode.firstChild.nodeValue;
        // move to next preference
        prefNode = prefNode.nextSibling;
      }
      // set MediaDisplay properties
      setupVideo();
    }
    else{
      trace("Error loading XML preferences.");
    }
  }

  myXML.load(configURL == null ? "config.xml" : configURL);
}
```

If you want to specify a different path or filename for the XML configuration file, you can use `flashvars` in flv_md.htm:

```
<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
codebase="http://fpdownload.macromedia.com/pub/shockwave/
➥cabs/flash/swflash.cab#version=6,0,65,0" width="320" height="267">
<param name="movie" value="flv_md.swf" />
<param name="quality" value="high" />
<param name="bgcolor" value="#ffffff" />
<param name="flashvars" value="configURL=preferences.xml" />
<embed src="flv_md.swf" flashvars="configURL=preferences.xml" quality="high"
bgcolor="#ffffff" width="320" height="267"
type="application/x-shockwave-flash"
pluginspage="http://www.macromedia.com/go/getflashplayer" />
</object>
```

The `togglePlay` function is called continuously by the `movieScrubber.onEnterFrame` function (see "Video Scrubber" above). It uses the `MediaDisplay.playing` property to determine if the video is currently playing. Depending on the value of `isPlaying`, the play or pause button is hidden.

```
function togglePlay():Void{
  var isPlaying:Boolean = videoDisplay.playing && !scrubbing;

  playpauseClip.pause_btn._visible = isPlaying;
  playpauseClip.play_btn._visible = !isPlaying;
}
```

The built-in properties, methods and events of the MediaDisplay component enable us to create a full-blown Flash video player in under 200 lines of code.

## Skinning the Player

Skinning the Flash video player is easy. All of the player controls can be found in the FLA library's **controls** folder.

| Symbol | Description |
|---|---|
| bg | The silver gradient that appears behind the controls |
| loadProgress | The dark gray progress bar that appears as the FLV is downloaded |
| pause | The pause button |
| play | The arrow-shaped play button |
| scrubDot | The square gray handle on the scrubber bar |
| scrubTimeline | The light gray bar that appears when the movie first loads |
| speakerIcon | The speaker movie clip that toggles volume |
| volumeDot | The rectangular gray handle on the volume slider |
| volumeWedge | The dark gray triangle that appears behind the scrubDot symbol |

There are two ways to modify these movie clips:

1. In the FLA library, double-click the movie clip you want to edit and make the modifications to the symbol directly. This is the best approach if you only want to change the fill, stroke, or size of an object.
2. Import new symbols into the FLA and save them in a new folder. From the main Timeline, drill down to the symbol you want to replace. Select the symbol on the stage and click **Swap** on the Property inspector. In the Swap Symbol dialog box, select the new symbol and click **OK**.

We tried to make the ActionScript as flexible as possible to accommodate skinning, but you may have to tweak it a little to get it just right (e.g., edit the `MovieClip.startDrag` parameters). For more player controls, check out Flash's built-in Buttons library (**Window > Other Panels > Common Libraries > Buttons**). You'll also find some great-looking controls in the source files of Peldi's FLV Player (http://www.peldi.com/blog/FLVPlayer.html) (look in the **final_skins** folder).

## Conclusion

In this article, you learned how to create a Flash video (FLV) player using the Flash MediaDisplay component. In Part 2, we'll create the same player *without* the MediaDisplay component. Instead, we'll use the `NetConnection` and `NetStream` classes. This results in a SWF file that's only 2 kilobytes in size!

Approximate download size: 3.2MB

**Keywords**
Flash video, FLV, progressive, Flash MX 2004, ActionScript 2.0, skin, Media class, MediaDisplay component, autoPlay, contentPath, volume, bytesLoaded, bytesTotal, playheadTime, complete, progress, event listener