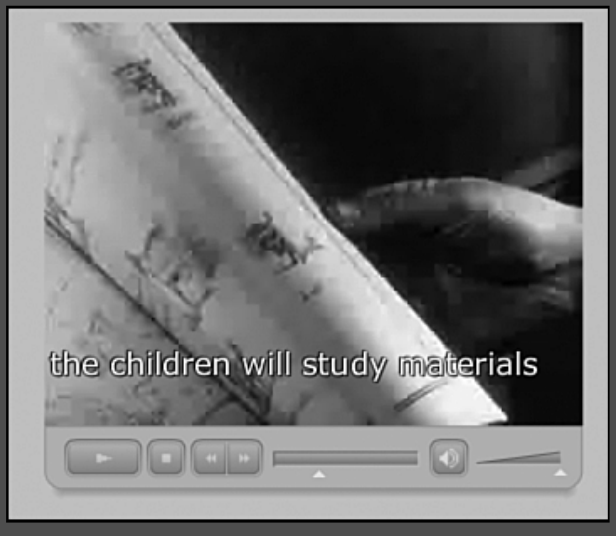
CHAPTER 3: Creating a Video with Synchronized Captions



About the Projects

The projects in this chapter are the most universally practical projects in a book full of practical projects! At the core, you'll build text *captions* for video content. You'll also learn to create any kind of *synchronization*. For example, you can display bits of trivia on top of the video (like the old VH1 music video show *Pop Up Video*). Although you can always include such content right inside the video, keeping it separate in the Flash movie lets you easily modify it or even temporarily turn it off. Everything here also applies to audio. Not only are the synchronization concepts identical for video and audio, you can use what you build here for any audio-only projects you develop.

Prerequisites

You'll need Flash Professional 8 (not Flash Basic). There are no other prerequisites for this project. Ideally, you will have your own video and audio content—a short piece with narration or dialogue is best for testing. If you understand XML, that's great, but it's not required.

@work resources

Download the `chapter_3_downloads.zip` **file for this chapter from the accompanying CD-ROM.**

Planning the Projects

Almost all the work for these projects is done in the preproduction stage. If you do a good job preparing everything, the projects will go more smoothly than a television cooking show (where everything is measured and chopped up in advance). The preparation here involves gathering the transcript of the captions that will appear and identifying the places at which those captions should appear.

After you have your captions prepared, you'll move on to creating a few text display templates. On the surface, this is nothing more than selecting a variety of fonts to use; however, you'll also add an animation effect for when the text changes. The text display templates you create can be used with any future project you build where you want captions. The rest of the chapter shows you how to implement the support class I built called `EventChannel`. It handles all the grunt work of reading in the captions and making them appear in synch with your audio or video.

I should note that captions are good for much more than simply displaying the movie's dialogue for hearing-impaired folks (although captions obviously do that well, too). I've used the techniques from this chapter to display captions in multiple languages for a museum kiosk. I've also used the techniques in this chapter to display the lyrics for music where the singer might not annunciate the words clearly. The projects you build in this chapter can apply to any situation in which you want something to appear (such as text) in synch with your audio or video.

Cue Points and Captions

There's one huge chore when captioning audio or video: transcribing—someone needs to type in all the text. Plus, that text needs to be segmented into blocks that both fit within a given screen space and match the tempo of the video. Preparing a transcript for cue points is nothing more than creating a text file with every block of text on its own line, as in this example:

```
There once was a man who loved a woman.
She was the one he ate that apple for.
and so on...
```

You simply want to make sure that each line fits in the space you're allotting for the captions and that, when synched with the audio, the captions appear on screen long enough to read. For example, if you have space for only one or two words to appear at a time, they'd have to fly by so fast no one could read them.

The final step of synchronizing the captions with the media goes very quickly after you have the transcript prepared. With the tools I built for this chapter, you'll see that you can identify *cue points* in real time while watching the video. All the investments you make to prepare the captions will make this otherwise tedious process go smoothly.

Cue Point Types

There are no fewer than five types of video cue points: Event, Navigation, ActionScript, Caption, and Marker. Although each has its respective benefits depending on your application, they're mostly equivalent. The idea of any cue point is the same: to associate a

block of information you want to appear—or, at least, get sent to your application—at a specific time in the video. Generally, the exact content for a cue point is whatever you decide. However, the exact form and where the cue point information resides are what vary in the different cue point types.

Videos produced in Flash Professional 8 (or the accompanying Flash Video Encoder) can be permanently injected with Event or Navigation cue points. The difference with Navigation cue points is that they are seekable (unlike Event cue points). That is, the Encoder places a *keyframe* in the video at the exact frame you place a Navigation cue point. This way, users can navigate directly to such keyframes by clicking the Next or Previous button in the `FLVPlayback` component shown in Figure 3.1 (which triggers the `seekToNextNavCuePoint()` and `seekToPrevNavCuePoint()` methods). You can find the `FLVPlayback` component by selecting Window, Components.

FIGURE 3.1 **The `FLVPlayback` component lets the user seek to the next or previous Navigation type cue point.**

ActionScript cue points aren't permanently embedded into the FLV, which makes them different from all the other types of cue points. I'll talk about the advantages or disadvantages of this approach in the next section. For now, just realize that ActionScript cue points are set at runtime; therefore, they require additional scripting before the video starts. (As an added bonus, the project "Implement Code for Audio-only Captions" shows you how to use ActionScript cue points with an audio source instead of a video source.)

Each of these three Macromedia cue point types (Event, Navigation, and ActionScript) have properties for time and name. The time corresponds to where the cue point appears in the video, and the name is an arbitrary string you specify. You can include any text you want, such as a caption or a description of what's happening onscreen. Plus, each cue point has room for additional *parameters*—namely, as many name/value pairs as you want. In this way, you can associate more than just a single line of text with a particular cue point. For example, you could store captions in different languages such as the following name/value pairs:

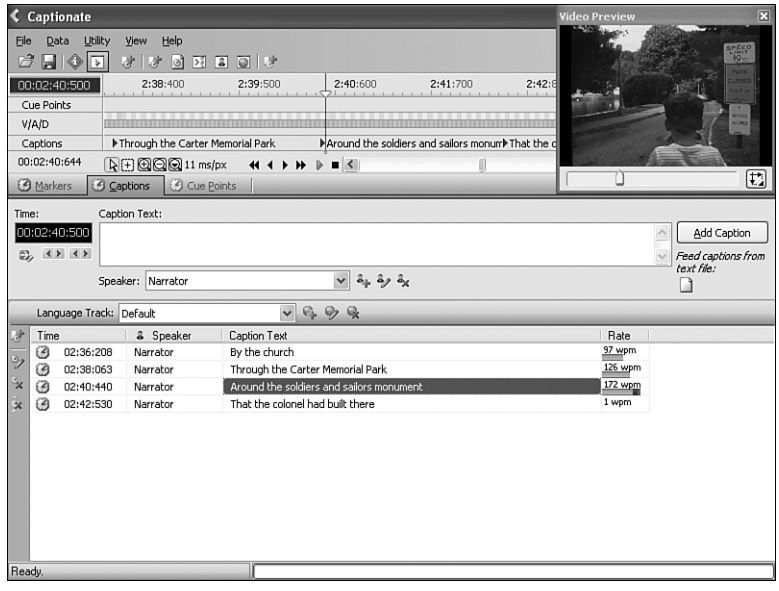
```
name:value
en:"Hello Friends"
es:"Hola Amigos"
fr:"bonjour amis"
```

Remember that you're packing information into a moment in time; you can store just a name or as many other parameters as you want.

Finally, although the third-party product Captionate (shown in Figure 3.2) is needed to

inject Caption and Marker cue points, these cue point types are definitely worth including in this list. Captionate Markers are simple: They're just text labels associated with a moment in time. However, Captionate Captions are similar to Macromedia cue points because they have room for additional, optional information (in addition to the caption itself). Captionate Captions also let you identify a speaker (that is, the person talking) with each caption.

FIGURE 3.2 **The Captionate interface lets you inject captions and markers.**



In fact, Captionate's Caption type of cue point supports multiple tracks for multilingual applications. You could probably squeeze this same kind of information into the Macromedia cue point format (Event, Navigation, or ActionScript), but the logical and convenient structure is already built in to the Captionate Caption cue point. (If nothing else, you'll want to get Captionate because it lets you modify any cue points—including Macromedia ones—embedded in the `.flv` file; otherwise, these are uneditable.)

You can use any or all of these cue point types in a single video. You can also write code that responds to each type differently, perhaps displaying Caption cue points in a text field and then using Event cue points to jump to a different frame in a movie clip. Instead of learning the different syntaxes to handle all the cue point types, the projects in this chapter channel all the cue points through a single clearinghouse: the `EventChannel` class. This class triggers events in your project for every cue point type you want to listen for.

Embedded Cue Points Versus Separate Text Files

One of the coolest features in Flash Professional 8 is that, during the video encoding stage, you can embed cue points right

into the video. As great as this feature is, though, it does have some disadvantages.

The biggest problem with embedding cue points into the video is that they're uneditable after the `.flv` file is encoded. (Well, the excellent third-party application called Captionate does let you edit cue points.) Although embedded cue points might be convenient because the data and video stay together, the data isn't left open for easy access. The alternative is to keep all the details of your cue points in an external source, such as an XML text file.

Saving cue point information in XML (a separate file from your video) can have its advantages. For example, if you have the same video encoded for different bandwidths, you can use a single XML file for the cue points. After all, the captions are the same for each bandwidth, so making an edit involves just one file instead of each `.flv` file. Plus, as you'll see in the tool I built for adding cue points, there are more convenient ways to specify cue points than through the Flash video encoder's interface. For one thing, when specifying cue points in Flash, you can't hear the audio track, which makes finding cue points very difficult indeed.

As great as storing cue point information in separate files is, if you choose that approach, you have to perform the additional step of importing and parsing the data. Also, although the `FLVPlayback` component has an `addASCuePoint()` method (to inject ActionScript cue points at runtime), you're pretty much taking a home-grown approach that might not match other developers' ways of working. I should note that, even though the only logical format for an external file containing cue point information is XML, the exact structure of that file (that is, its schema) is up to you. As long as you leave room for all the cue point types, the exact schema is completely subjective. However, for this project to work, you have to use a single format. The structure I'm using is based on the output from Captionate—which, I suppose, makes this format "better" only because it's consistent with another product.

Project: Navigation Cue Points in a .flv File

In this project, you'll create a `.flv` video file you can use in the remaining projects in this chapter. You'll also use Flash to insert Navigation cue points into the video. You'll use an excerpt from a public domain video called *The Children Must Learn*. View the source file `the_children_must_learn.mov` (located in the `source_media` folder you downloaded for this chapter) to get a sense of where the captions might appear.

STEPS▼

1. **Creating a new `.fla` file**
2. **Importing the source video**
3. **Choosing video options**
4. **Adding the first navigation cue point**
5. **Adding more navigation cue points**
6. **Skinning the video**
7. **Navigating the video**

STEP 1▼
Creating a New .fla File

Create a new Flash document and immediately save it as **working.fla**.

STEP 2▼
Importing the Source Video

Now, import a video clip into the new `.fla` file. Select File, Import, Import Video to launch the Import Video wizard, as shown in Figure 3.3.

Click the Browse button and select the `the_children_must_learn.mov` video (located in the `source_media` folder). Click Next.

STEP 3▼
Choosing Video Options

In the Deployment page of the Video Import wizard, leave the Progressive Download from a Web Server option selected, as shown in Figure 3.4, and click Next. The other options on this page of the wizard include two ways to stream a video from the Flash Media Server (formerly called Flash Communication Server) plus an option to embed the video into your `.swf` (neither of which we want to do here). Using the Flash Media Server (or a service provider) is an additional expense. Embedding the video into your `.swf` results in lower quality and audio that drifts out of synch over time.

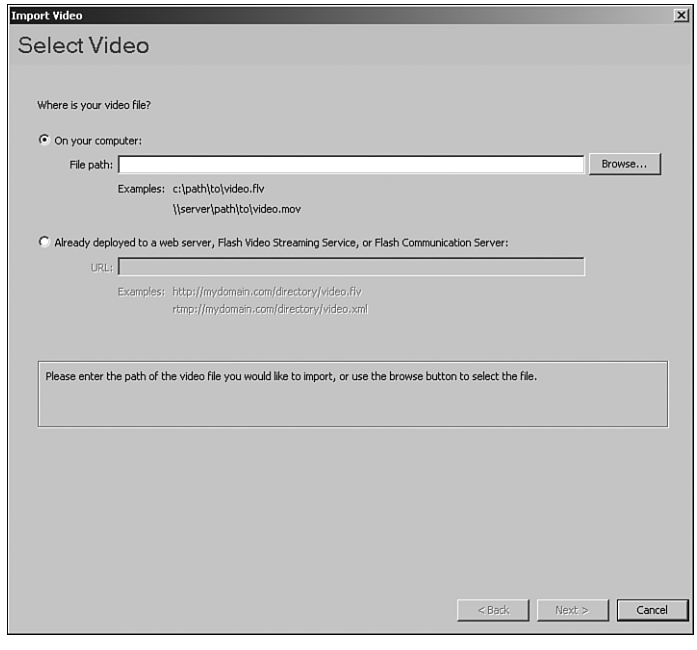


FIGURE 3.3 **The first screen in the Import Video wizard asks you to browse to your source video.**

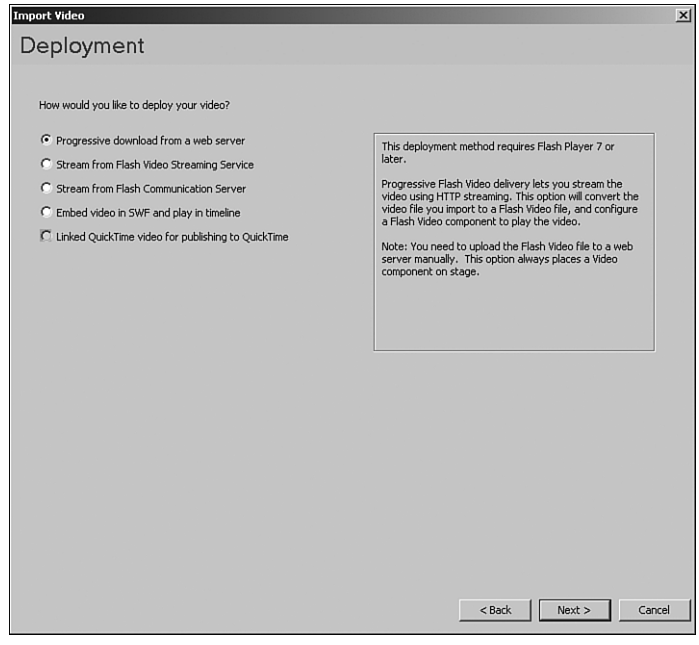


FIGURE 3.4 **Select the Progressive Download option in the Deployment page of the Import Video Wizard.**

In the Encoding page of the Import Video Wizard, select the Flash 8—Medium Quality (400kbps) encoding profile from the drop-down menu. Click the Show Advanced Settings button to expand the window so it shows additional options, and then click the Cue Points tab in the newly exposed section of the window. Figure 3.5 shows the dialog box you should be viewing.

STEP 4▼
Adding the First Navigation Cue Point

At this time, because Flash unfortunately provides no audio preview, you'd be hard pressed to set cue points for captioning.

Instead, let's just insert Navigation cue points. Specifically, we'll insert them every time the video image displays a new page of the "Let's Learn About Goats" booklet.

Scrub the video to as close to 3.187 seconds as you can (displayed as `00:00:03.187`); then click the plus button in the upper-left corner of the Cue Points list to add a cue point at this location in the video file. A new entry is created for this cue point in the Cue Points list. Set the cue point type to Navigation by selecting this option from the Type column. Next, type a name for the cue point in the Name column—for this example, use the name **The Goat Family** because that's the title of the page of the booklet displayed in the video (see Figure 3.6).

FIGURE 3.5 The Encoding dialog box, shown here with the Cue Points portion of the Advanced Settings.

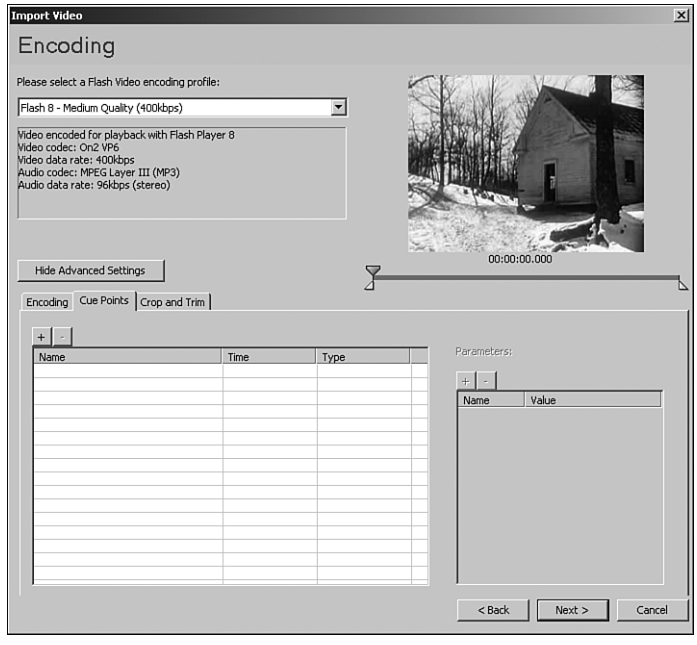
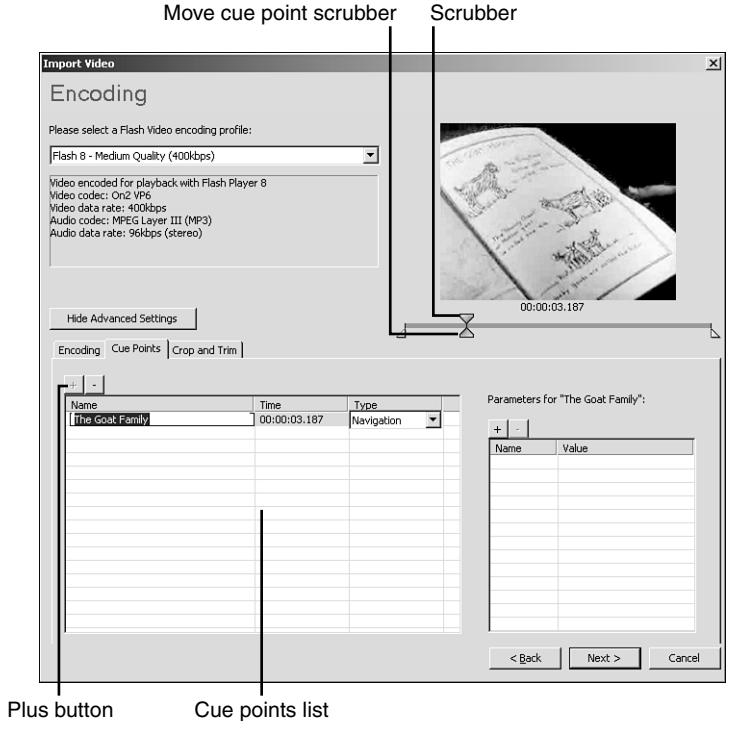


FIGURE 3.6 The advanced settings of the Import Video Wizard let you insert cue points.



STEP 5▼
Adding More Navigation Cue Points

Scrub the video to 5.247 seconds (where the booklet in the video reads *TYPES OF GOATS*), and click the plus button to add another cue point. Name this cue point **Types of Goats** and remember to select Navigation from the Type column. Add three more Navigation cue points at the times shown in Figure 3.7.

FIGURE 3.7 **These are all the Navigation cue points you'll insert.**

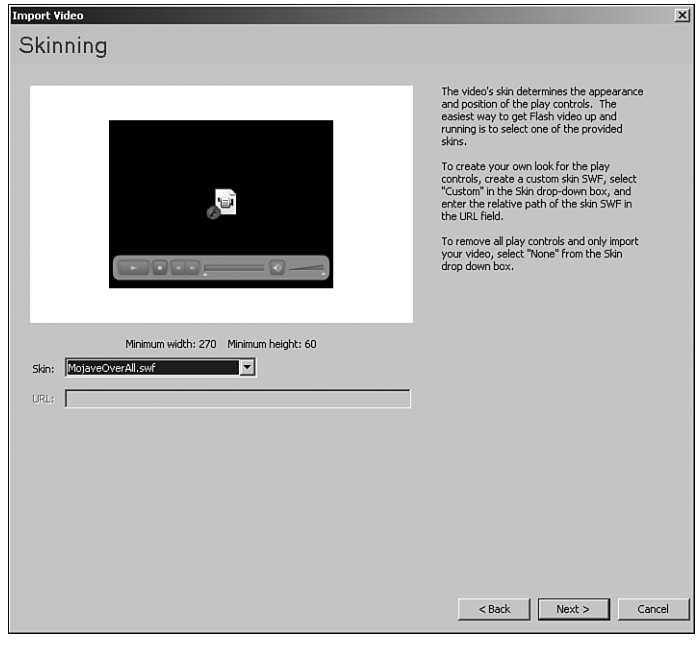
Name	Time	Type
The Goat Family	00:00:03.187	Navigation
Types of Goats	00:00:05.247	Navigation
Uses of Goats	00:00:07.307	Navigation
Uses of Goat Milk	00:00:08.778	Navigation
Tomato Poem	00:00:13.191	Navigation

STEP 6▼
Skinning the Video

Click Next to arrive at the Skinning page in the wizard, as shown in Figure 3.8.

Here you can select a general theme for the look and feel of the video controls. There are themes for Artic (cool blue), Clear, Mojave (beige), and Steel (gray). Each skin has one version where the controls appear on top of the video (these skins have *Over* in their names), and one version where the controls appear underneath the video (these skins are identified with *External* in their names). Although we're only working on producing the `.flv` file at this point, select one of the *skins* with a name that ends in *All*—just so you can see the navigation feature at work. Click Next and then, at the last step, click Finish.

FIGURE 3.8 **You can select the look and feel for your video controls in this Skinning dialog box.**



STEP 7▼
Navigating the Video

It should take a minute or so to compress the video file, depending on your computer. When it's complete, the progress bar disappears and you should see that, next to your `working.fla` file, a `.flv` file is present and named `the_children_must_learn.flv`. Select Control, Test Movie; the `.swf` will load the `.flv` you just created at runtime. It's sort of magical the way the `FLVPlayback` component gets configured automatically. You can pause and click the double-arrows in the video controller to jump to the Navigation cue points we added.

You can add the text for captions using the same technique we just used to add Navigation cue points. However, we're not going to do that because it's too difficult to select the times without an audio preview, which is simply not supported in Flash. We'll use another technique to create captions in the projects that follow. Realize that the primary goal of this project is to create a `.flv` file you can use in subsequent projects—and to see how you can use Flash to inject cue points.

Project:
ActionScript Cue Points for Captions in an XML File

In this project, we'll create an XML document containing cue point information. Namely, we'll specify the text captions and when—in the video—they should appear. The process involves using a separate application I built just for this purpose called `gathering_tool.swf`. The data you collect will become ActionScript cue points that get added at runtime simply because they won't be embedded in the `.flv` video; rather they'll be stored in an XML document.

STEPS▼

1. **Preparing to use the offline gathering tool**
2. **Loading the video and transcript into the gathering tool**
3. **Setting cue points while the video plays**
4. **Exporting the XML file**

STEP 1▼
Preparing to Use the Offline Gathering Tool

Copy the `gathering_tool.swf` file from the `gathering_tool` folder you downloaded and place a copy next to the `.flv` file produced in the first project (automatically named `the_children_must_learn.flv`). (You can also find a finished version of `the_children_must_learn.flv` in the `finished_source` folder you downloaded for this chapter.)

Copy the `video_captions.txt` file from the `starter_files` folder and place a copy of that file in your working directory.

STEP 2▼
Loading the Video and Transcript into the Gathering Tool

Double-click the `gathering_tool.swf` file to launch it in the Flash player. Type the name of the caption file (in this case, **video_captions.txt**) into the Transcription field and click the Load Text button. The contents of that text file appear in a list on the right, and the first row of that list is selected as shown in Figure 3.9.

Type the name of the `.flv` video file to which you want to add the captions (in this case, type **the_children_must_learn.flv**) into the Media field and click the Load button.

Next, you'll be using the `gathering_tool.swf` to collect cue points. Familiarize yourself with this tool (see Figure 3.10).

FIGURE 3.9 **The `gathering_tool.swf` can load the entire transcript from a text file.**

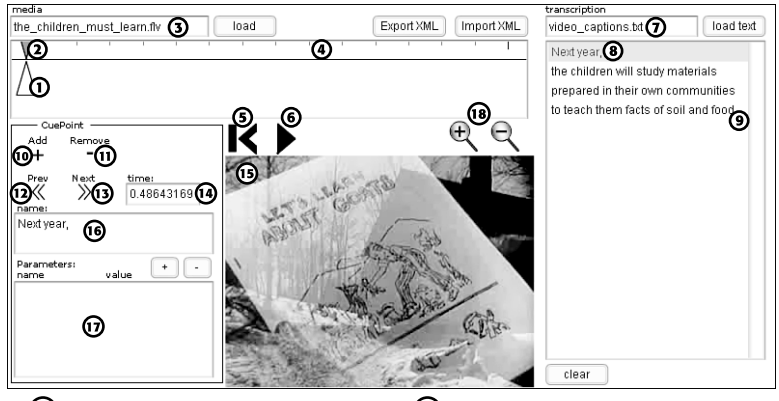
STEP 3▼
Setting Cue Points While the Video Plays

Now for the fun part. Click the Play button to watch the video as many times as needed to get an idea of when the captions are supposed to appear. You can scrub the video (drag the upward-pointing triangle), but when you let go, the video will jump to the closest keyframe. That is, when you encode a video, it automatically embeds keyframes where significant changes occur onscreen. The in-between frames contain only the parts that have changed. Therefore, you can only seek to keyframes (not to the in-between frames).

When you're ready, make sure the first row of the transcription list is still selected. Rewind the video and then click the Play button to start the video playing. Get ready to click the Add button! When the narrator says, "Next year," click the Add button to create a cue point and automatically advance to the next line in the transcription list (which is, "the children will study materials"). Then, at the right moment, click the Add button again to insert a cue point for the second line in the transcript. Because the captions appear quickly, you might need a few tries to get it right. If you accidentally add a cue point, you can click its triangle and then click the Remove button. Just remember that if you're

coming back through to add more cue points, you need to first select the row in the transcription list for the text you want to insert next.

FIGURE 3.10 **The `gathering_tool.swf` has a bunch of handy features.**



1. Current position
2. Cue point (filled when selected)
3. Media filename
4. Timeline
5. Rewind media
6. Play media
7. Transcript filename
8. Currently selected row (inserted with next cue point added)
9. Transcript contents
10. Add cue point button
11. Remove cue point button
12. Jump to previous cue point
13. Jump to next cue point
14. Time
15. Video preview
16. Name
17. Parameters (name/value pairs)
18. Zoom timeline

After you insert the cue points, rewind and then click the Play button to watch the captions appear in the cue point area. See whether the synchronization is close. For your reference, my five cue points were inserted at 3.239 seconds, 4.257 seconds, 5.825 seconds, 8.071 seconds, and 11.311 seconds. However, it's not as though you need to have those exact numbers because no one will notice if the caption appears a fraction of a second too early or late.

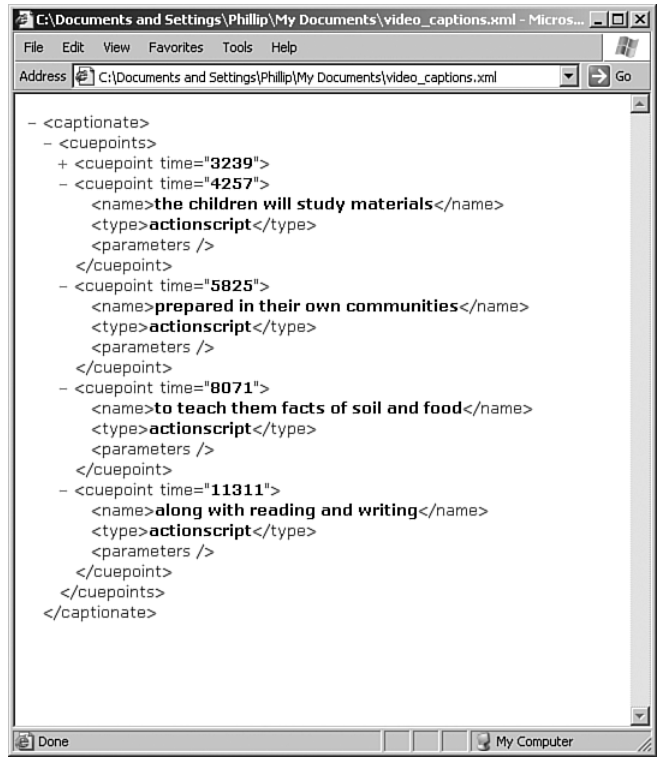
Don't close `gathering_tool.swf` because we still have to export the cue points! (Note that we haven't actually injected the caption cue points into the `.flv` file the way we did with the Navigation cue points in the project "Navigation Cue Points in a `.flv` File.") In the next step, you will take the cue point information and export it to a text file.

STEP 4▼
Exporting the XML File

In `gathering_tool.swf`, click the Export XML button. Press Ctrl+C to copy the XML string the tool just generated for you. Create a new text file using Notepad or a similar program,

and paste the contents of the XML string you just copied. Save the file as **video_captions.xml** in your working directory (we'll use it in the upcoming projects). You might want to view the XML file in a tool such as Internet Explorer (shown in Figure 3.11).

FIGURE 3.11 **The XML file is much easier to read when viewed in Internet Explorer.**



Now you have an XML file containing both the captions' values (the text) and the exact moment during the video when those captions should appear. You can use `gathering_tool.swf` for any video or audio file. Just import a different `.flv` video file (or `.mp3` file) and a corresponding transcript of the captions you want to add. The advantage of this tool is that you can identify the cue points in a natural manner—while the video plays.

Project: Basic Caption Display Template

A video can contain a lot of cue points, but you need a vehicle to display them, such as a text field. In this project, we'll build a simple text display area that serves as a template from which you can create other styles and layouts. In the next project, we'll link the

video to this .swf template file so the cue points from the video are sent to this text display area.

STEPS▼

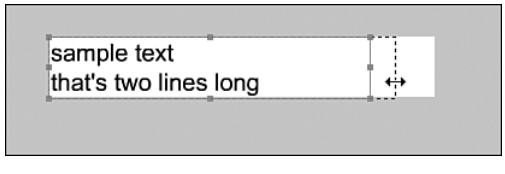
1. **Creating a captionType1.fla file**
2. **Adding the minimum code**
3. **Creating the .swf file**

STEP 1▼
Creating a captionType1.fla File

Create a new Flash file and save it in your working directory with the name **captionType1.fla**. Select Modify, Document from the menu bar and set the width to **320** and the height to **50**. These settings change the .fla file to match the video's width and give you enough room for two lines of text—which should hold the caption text nicely.

Select the Text tool and create a block of text that fills the 320 × 50 stage. Use the handles on the text block to resize the text field area, as shown in Figure 3.12 (not the Properties panel's W and H fields because they scale the text).

FIGURE 3.12 **Resize the text field using the handles, not the Properties panel.**



Next, use the Properties panel to set the Text Type to Dynamic Text. Set the instance name to **_txt** and set the Line Type to Multiline.

Finally, choose a font that will accommodate two lines of text (you can just type the longest caption in your project to see whether it fits in the newly adjusted text field). Next, set the text color to White.

To make the white text stand out on top of any background video, add a filter effect. With the block of text selected (not the characters *in* the text) select, Window, Properties, Filters and add the Drop Shadow filter—in fact, add two. For both, set the Blur X and Blur Y options to 0, the Strength to 100%, the Quality to Low (for better performance), and the Distance to 1. The default black color for the shadow will work fine. However, set one of the filter's angles to 225 and leave the other at the default angle of 45.

Finally, for the filter to really look great, return to the Properties panel and, with the text field instance selected, click the Embed button. Click the Basic Latin row and then click OK. This step adds to your .swf the font outlines for the font you chose. This way, the user won't need to have installed the font you selected. In addition, the filters have higher-quality results when the font is embedded.

STEP 2▼
Adding the Minimum Code

Each .swf file you create for this project can be used to display the text for a video. To do that, each .swf must implement the same minimum set of three functions: clear(), showText(), and getSize(). (They're formally defined in the ICaption.as interface file.) We'll add those functions now. Select the first keyframe in captionType1.fla and open the Actions panel. Type the code shown in Listing 3.1.

LISTING 3.1 This Code Displays Captions in the `captionType1.swf` File

```
function clear(){
   _txt.text = "";
}
clear();

function showText(name:String, wholeObject:Object, speed:String){
   _txt.text = name;
}

function getSize():Object{
   return {width:320, height:50};
}
```

Although you need all three of these functions in any template, you'll be able to modify how they're handled if you want to extend this project. (For now, use the code I provided here.) The `clear()` function is called immediately, and here we're just clearing the text field of any text that might already be there. The `getSize()` function returns (to the custom `CaptionHolder` class I built for this chapter) the size of this window so it can be masked out. Notice that `320` and `50` match the width and height of your movie. By providing the `getSize()` method, the `CaptionHolder` class can load any sized `captionType.swf` you create. (By the way, don't use Flash's `Stage.width` and `Stage.height` here because they won't report the correct values—the `captionType1.swf` file is ultimately loaded into a larger file that contains your video and `Stage.width` reflects the width of that file.) Finally, the `showText()` function is where you'll do the most modification in the more advanced variations of this project. Right now, it's very simple: When the video reaches a cue point, the cue point name is passed to the `showText()` function. In this case, we simply display it in the `_txt` field instance onstage. There are additional parameters that we won't use until the "Advanced Captioning Template" project, later in this chapter.

STEP 3▼
Creating the .swf File

Finally, save the `captionType1.fla` file and select Control, Test Movie to generate the file `captionText1.swf`, which we will associate with the video in the next project. (The `.swf` we generate here has a blank screen.)

Project:
Channeling Cue Points to the Caption Display

Now we'll pull things together. Namely, we'll play the `the_children_must_learn.flv` video file we generated earlier while sending the captions gathered using the `gathering_tool.swf` to the `captionText1.swf` file we just built.

STEPS▼

1. **Creating the `main.fla` file**
2. **Creating the `CaptionHolder` symbol**
3. **Assembling support class files**
4. **Implementing the final code**
5. **Alternative 1: making the video play automatically**
6. **Alternative 2: adding a second `captionText1.swf` that supports navigation cue points**

STEP 1▼
Creating the main.fla File

For this project, we need to create a simple movie file that contains an `FLVPlayback` component that points to our `.flv` file.

Create a new Flash file and save it as **main.fla** in the same directory in which you've been working—the one that contains the video file `the_children_must_learn.flv`.

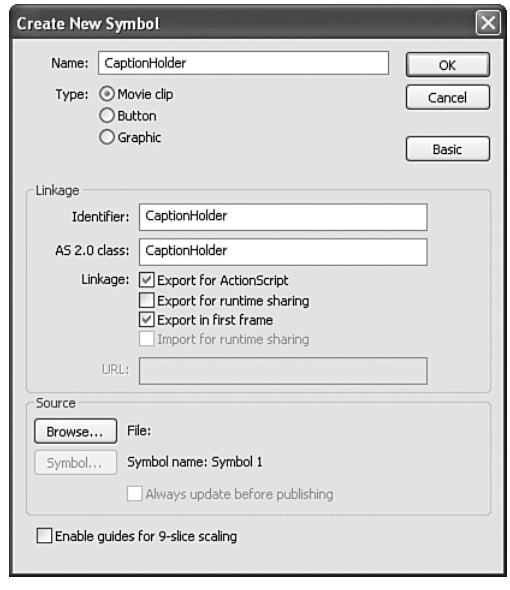
Select Window, Components and drag onto the stage an `FLVPlayback` component (from the FLVPlayback - Flash 8 category). Use the Properties panel to give the component an instance name of **playback**.

STEP 2▼
Creating the CaptionHolder Symbol

Select Insert, New Symbol. Click Advanced to expand the dialog box if isn't already expanded. Make sure the Type is set to Movie Clip and name it **CaptionHolder**. Next, enable the Export for ActionScript check box, which automatically enables the Export in First Frame check box. Then fill in both fields (Identifier and AS 2.0 Class) with the name **CaptionHolder** so your dialog box looks like the one in Figure 3.13. Click OK.

Now this empty symbol will be associated with the class file `CaptionHolder.as`.

FIGURE 3.13 **When you create the empty `CaptionHolder` movie clip, you can associate it with an ActionScript class file (`CaptionHolder.as`—but you don't type the `.as`).**



After you click OK, you are taken inside this new clip. Although you don't have to put anything here (the `captionText1.swf` file is loaded at runtime), draw a rectangle that's exactly 320 × 50 pixels (the space your captions need). Use the Info panel to ensure the rectangle's upper-left corner is aligned with the center of the clip (the plus sign), as shown in Figure 3.14.

FIGURE 3.14 **The rectangle you draw inside the `CaptionHolder` symbol should have its upper-left corner aligned with the symbol's center (where the plus sign is shown).**

Select your drawn rectangle and then select Modify, Convert to Symbol. Make sure the Movie Clip option is selected and name it **Rectangle**. After you click OK, use the Properties panel to give the Rectangle symbol an instance name of **preview**. Finally, click the first keyframe—still inside the `CaptionHolder` symbol—and type this code:

```
preview._visible = false;
```

This way, the `preview` instance is visible only while authoring.

Return to the main timeline of `main.fla` and drag an instance of `CaptionHolder` onto the stage. Because it contains the rectangle shape, you can position it below the `playback` instance or right on top—wherever you want. Use the Properties panel to give the `CaptionHolder` symbol on stage an instance name of **captions_clip**.

STEP 3▼
Assembling Support Class Files

You'll need to add two class files for the code to work. Copy the two homemade class files I created, named `CaptionHolder.as` and `EventChannel.as`, from the `starter_files` folder to your working directory. The following is a list of the minimum set of files that need to be in the same folder as `main.fla`:

- `the_children_must_learn.flv`
- `video_captions.xml`
- `captionType1.swf`
- `CaptionHolder.as`
- `EventChannel.as`

The FLVPlayback skin you selected, such as `MojaveOverNoVol.swf`, must also be adjacent to `main.fla`. (You're welcome to move the video, the XML, and the `captionType1.swf` files into a subfolder—just remember to add the path to that subfolder when specifying filenames in the next step.)

The `CaptionHolder.as` class file is needed by the `CaptionHolder` symbol you created in step 2. It handles loading the `captionType1.swf` file (or whatever template you specify). The `EventChannel.as` class does a lot of work, as detailed at the end of this chapter. It has three primary features:

- It takes all the events broadcast by the different cue point types in an `FLVPlayback` and channels them through a single, consistent event broadcast. That is, the cue point types all fire off different events (with different parameters and such), and the `EventChannel.as` class makes them all consistent.
- It handles the task of loading an optional XML file full of cue point information and injecting that information (as ActionScript cue points) at runtime.
- It supports the Sound object. This means that nearly everything you do with the `FLVPlayback` component and

video you can do with an audio clip. (And you will in the next project.)

In doing all this work, the `EventChannel.as` class insulates you from the tedious code and keeps the code you have to write for each new project to a minimum.

STEP 4▼
Implementing the Final Code

Inside `main.fla`, select the first keyframe and open the Actions panel. Type the code in Listing 3.2.

There's very little code here, but it's worth digging into. The first three lines simply set up the `FLVPlayback` instance (`playback`), the same as using the Parameters panel but with script instead. Notice that you must set `autoPlay` to `false` because you need to wait for `video_captions.xml` to fully load. (I'll show you alternative code that makes the video start playing automatically in the next step.)

Line 5 creates an instance of the `EventChannel` class (saved in the variable `myEventChannel`). Next, we pass three parameters when triggering the `init()` method on the `captions_clip` instance (the `CaptionHolder` symbol you put on the stage—and therefore an instance of the `CaptionHolder` class). Those parameters are the path to the `captionType1.swf` template file we created earlier, a reference to the `myEventChannel` instance, and an array (`eventList`) that specifies which event types you want sent to the `captionType1.swf` template. (This array can include any of the following caption types: `"event"`, `"navigation"`, `"actionscript"`, `"caption"`, or `"marker"`. In this example, though, because the array contains just `"actionscript"`, we want to display only the cue points from the XML file.)

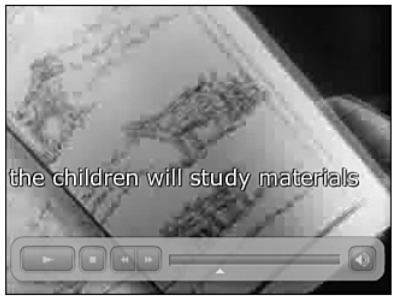
LISTING 3.2 This Code Associates the `EventChannel` Class with Your `captionType1.swf` File

```
1   var playback:mx.video.FLVPlayback;
2   playback.autoPlay = false;
3   playback.contentPath = "the_children_must_learn.flv";
4
5   var myEventChannel:EventChannel = new EventChannel();
6
7   //send init() function to the CaptionHolder instance
    //on stage (captions_clip)
8   var url = "captionType1.swf";
9   var eventList = ["actionscript"];
10  captions_clip.init(url, myEventChannel, eventList);
11
12  myEventChannel.init(playback, "video_captions.xml");
```

Finally, in the last line, we initialize `myEventChannel` by specifying the media source—in this case, it's the playback, which is an `FLVPlayback` component, but it could also be a `Sound` instance—and the XML file where ActionScript cue points can be found. The second parameter is optional, meaning you don't have to load cue points from the file if you don't want to.

Select Control Test Movie and then click the Play button on the `FLVPlayback` component to view the video. Figure 3.15 shows what mine looks like: The captions are on top of the video as if they were there the whole time.

FIGURE 3.15 **The finished video with overlaying captions.**



STEP 5▼
Alternative 1: Making the Video Play Automatically

There are two quick variations to this project worth examining. First, you might not like the fact that the video didn't automatically start playing. Add the following code *before* the last line, which triggers `init()` on `myEventChannel`:

```
function ready(){
   playback.play();
}
myEventChannel.addEventListener("ready",
                                this);
```

If this code looks familiar, that's because the `myEventChannel` instance supports every event supported by the `FLVPlayback` component (that is, our `playback` instance). Here, when the `ready` event fires, you tell the `playback` instance to `play()`. The `myEventChannel` class's `ready` event fires only after the XML has safely loaded *and* the `playback` instance has fired its `ready` event.

Select Control, Test Movie to confirm that the video begins playing automatically.

STEP 6▼
Alternative 2: Adding a Second captionText1.swf That Supports Navigation Cue Points

Just to see how versatile this application is, let's view those Navigation cue points you injected into the video back in the first project. One way you could do this is by simply passing `["navigation", "actionscript"]` (instead of just `["actionscript"]`) for the third parameter when invoking `init()` on the `captions_clip` (lines 9–10 of Listing 3.2). But seeing both cue point types in the sole `captionsType1.swf` would get messy.

Do this instead: Create a duplicate of the `captions_clip` instance on stage and give it an instance name of **captions_clip2**. Move it to another area, such as above the video. Add the following code directly below where you call `init()` on `captions_clip` (line 10 of Listing 3.2):

```
var url = "captionType1.swf";
var eventList = ["navigation"];
captions_clip2.init( url,
                     myEventChannel,
                     eventList );
```

It's the same as the original call, except you're calling `init()` on `captions_clip2` (not on `captions_clip`) and passing a different list of event types to support. (Well, just one type, but notice that it's an array, so you can specify more types. In fact, if you omit this parameter all cue point types are sent to the clip.)

NOTE

By the way, if you were showing only Navigation or Event cue points, you wouldn't need to import the XML file because the cue point information would be in the `.flv` file. In that case you'd change

```
myEventChannel.init(playback,
"video_captions.xml");
```

to simply read

```
myEventChannel.init(playback);
```

Now when you test the movie, you'll see both ActionScript cue points (the captions in the XML file) and the names of the Navigation cue points embedded in the video.

Project: Code for Audio-only Captions

To make a project like the one we just completed, but with audio this time, you use a nearly identical process. The only difference this time is that you can't inject the `.mp3` file with Navigation cue points. You can only use ActionScript cue points.

STEPS▼

1. **Capturing the cue points**
2. **Creating the `main_audio.fla` file**
3. **Assembling support files**
4. **Writing the code**
5. **Alternative: using the `MediaPlayback` component**

STEP 1▼ Capturing the Cue Points

Follow the steps for the second project, "ActionScript Cue Points for Captions in XML File," but this time grab the file `the_children_must_learn.mp3` from the `source_media` folder and the `audio_captions.txt` file from the `starter_files` folder. Enter those filenames when you run the `gathering_tool.swf` file, as shown in Figure 3.16.

When you're done gathering cue points, be sure to click the Export XML button. Then copy and paste the XML string into a text file you save as **audio_captions.xml**.

NOTE

By the way, there's a long pause after the narrator says, "Some of them live a long way off." If you want the text to go away during the pause, simply insert an extra cue point and enter an empty string into the Name field.

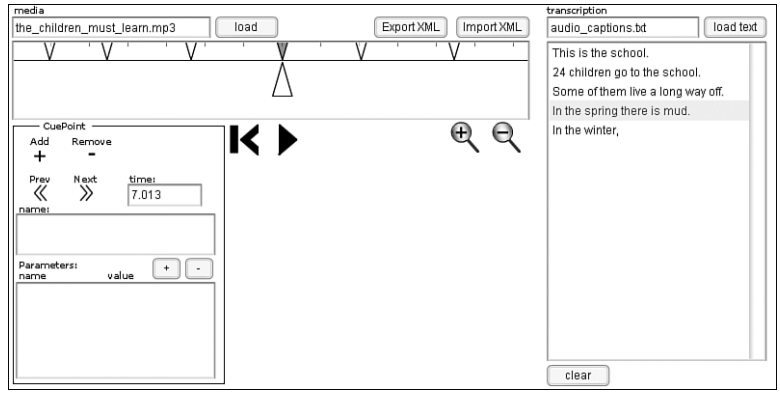


FIGURE 3.16 **You'll use the same `gathering_tool.swf` but this time for an audio track only.**

STEP 2▼
Creating the main_audio.fla File

Create a new Flash file and save it in your working directory as **main_audio.fla**. Either copy an instance of the `CaptionHolder` symbol created in the last project, "Cue Points in the Caption Display," or repeat step 1 of the preceding project.

Place on stage an instance of `CaptionHolder` and give it an instance name of **captions_clip** (just like before).

STEP 3▼
Assembling Support Files

Ensure that your working directory has, at least, the following support files in addition to `main_audio.fla` (you'll find the `.as` files in the `starter_files` folder and the `.mp3` in the `starter_media` folder—you need to create the `.swf` and `.xml` files yourself):

- `the_children_must_learn.mp3`
- `audio_captions.xml`
- `captionType1.swf`
- `CaptionHolder.as`
- `EventChannel.as`

STEP 4▼
Writing the Code

In the `main_audio.fla` file, select the first keyframe and open the Actions panel. Type the code shown in Listing 3.3.

LISTING 3.3 This Code Channels Captions for an Audio Track to the Same `captionType1.swf` Used Earlier

```
1   var myEventChannel:EventChannel = new EventChannel();
2
3   var url = "captionType1.swf";
4   var eventList = ["actionscript"];
5   captions_clip.init( url, myEventChannel, eventList );
6
7   var mySound:Sound = new Sound();
8
9   function ready(){
10     mySound.loadSound( "the_children_must_learn.mp3", true );
11  }
12  myEventChannel.addEventListener( "ready", this );
13
14  myEventChannel.init( mySound, "audio_captions.xml" );
```

In Listing 3.3, I was extra careful not to start the audio until the `ready` event fires. If you don't have any captions at the start of the sound, you can safely start playing the sound anytime you want and forgo lines 9–12. The only critical sequence issues are that `myEventChannel` is instantiated before you invoke `init()` on the `captions_clip` (because you're passing a reference to the `myEventChannel` instance) and that the `mySound` instance is instantiated before you invoke `init()` on `myEventChannel` (because you're passing a reference to the `mySound` instance).

Save and test the movie.

STEP 5▼
Alternative: Using the MediaPlayback Component

Before I show you how to use the `MediaPlayback` component for this audio-only project, I want to tell you about the two advantages it offers. The first benefit is that it works with `.mp3` audio files as well as `.flv` videos. Second, because the `FLVPlayback` component requires Flash Player 8, you'll want to use the `MediaPlayback` component if you're delivering a project to Flash Player 6 or 7.

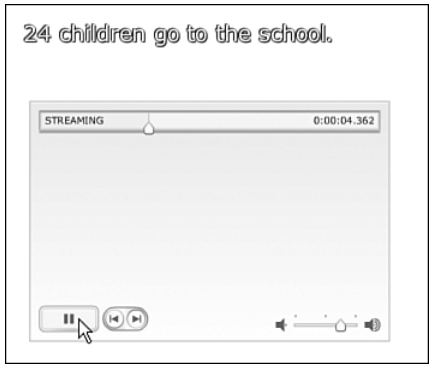
Using the `MediaPlayback` component instead of the `FLVPlayback` component is simple. Open the Components panel and drag a `MediaPlayback` instance onstage (from the Media - Player 6-7 category). Give it an instance name of **myMediaPlayback**. Replace the code you wrote in step 4 with the code in Listing 3.4.

Basically, you can pass an `FLVPlayback` instance, a `Sound` instance, or—as shown here—a `MediaPlayback` component instance. The cool part about this version shown in Figure 3.17 is that, because you're using the `MediaPlayback` component, you automatically get a bunch of features, such as pause, play, and scrubbing. Remember that, although the `FLVPlayback` component also supports these features, it works only with video.

Additional Support for Sound Objects

Originally, I thought adding support for the Sound object made the `EventChannel.as` class particularly useful, which it does. However, programming a scrub bar for audio involves a fair bit of code, in addition to requiring you to temporarily disable the `EventChannel` instance from continuing to broadcast events. I added two public methods to the `EventChannel` class to do this: `stopMonitoring()` and `startMonitoring()`. In addition, while the user is scrubbing you can call `scrubTo(`*`milliseconds`*`)` to fire off the closest cue point to a given time—my class even prevents the same cue point from firing repeatedly. I mention all this for two reasons: First, if you want to dig into the code, you can (for that matter, I included a sample file called `bonus_scrub_audio.fla` in the download files). The second reason is that I wanted you to see my rationale behind adding support for the `MediaPlayback` component (which is explained in the next step). The `MediaPlayback` component works when delivering audio or video to Flash Player 6 or 7 (whereas the `FLVPlayback` component requires Flash Player 8). The `MediaPlayback` is definitely not my favorite component because it's next to impossible to skin and it's not consistent with other components. However, it's still compelling and hard to resist, especially now that you can easily add cue points in the same manner you've done for the `FLVPlayback` component and `Sound` class.

FIGURE 3.17 **The `MediaPlayback` component is hard to resist when playing audio because it includes playback controls and is easy to use.**



LISTING 3.4 **This Code Plays a `.mp3` (and Its Captions) Through the `MediaPlayback` Component**

```
import mx.controls.MediaPlayback;
var myMediaPlayback:MediaPlayback;
myMediaPlayback.contentPath = "the_children_must_learn.mp3";

var myEventChannel:EventChannel = new EventChannel();

var url = "captionType1.swf";
var eventList = ["actionscript"];
captions_clip.init(url, myEventChannel, eventList);

myEventChannel.init(myMediaPlayback, "audio_captions.xml");
```

Project: Advanced Captioning Template

The projects in this chapter have thus far showed you various ways of collecting cue point information and then implementing the code to work with the accompanying class files. The only Flash file where you got to do any sort of layout work was the basic template from the third project, "Basic Caption Display Template." Now you'll get to see how that template can be expanded. That is, you can make as many template types as you want. For example, you can change the overall theme by modifying colors and fonts. In addition, your template doesn't have to show text at all. As long as your template includes a minimum set of features (namely, functions for `clear()`, `showText()`, and `getSize()`), you can make it perform however you want.

The features added to the basic display template in this project make the template more effective at displaying captions. In the next project, you'll make a template for a synchronized Flash display instead of captioning *per se*. The first feature you'll add here is a subtle transition animation that runs anytime the text updates. Although this might seem gratuitous, I think it's an effective way to cue the user that the text has updated because her attention might have drifted to the images in the video. The second feature you'll add is a hide/close feature. Some users might not want to view the captions, so allowing the viewer to turn off the captioning is a nice option to include.

STEPS▼

1. **Creating the `captionType2.fla` file**
2. **Nesting the text in a clip**
3. **Modifying the code to move the text**
4. **Publishing and testing**
5. **Adding code for the Hide/Reveal feature**
6. **Creating the Hide/Reveal button**

STEP 1▼
Creating the captionType2.fla File

If you have `captionType1.fla` handy from the earlier project, just open that and immediately select File, Save As. Then, name the new file **captionType2.fla**. (Use the version of `captionType1.fla` from the `finished_source` folder if you don't have your own. But remember to save it as **captionType2.fla** in your working directory.)

STEP 2▼
Nesting the Text in a Clip

To easily duplicate and move the text, select the `_txt` instance on stage and then select Modify, Convert to Symbol. Select the Movie Clip option and name the symbol **clip**. Also be sure you select the upper-left registration option. Click OK and then use the Properties panel to set the instance name to **clip**.

STEP 3▼
Modifying the Code to Move the Text

Select the first keyframe and open the Actions panel. Completely replace the existing code with the code in Listing 3.5 (which, by the way, you can copy and paste from `captionType2.fla` in the `finished_source` folder):

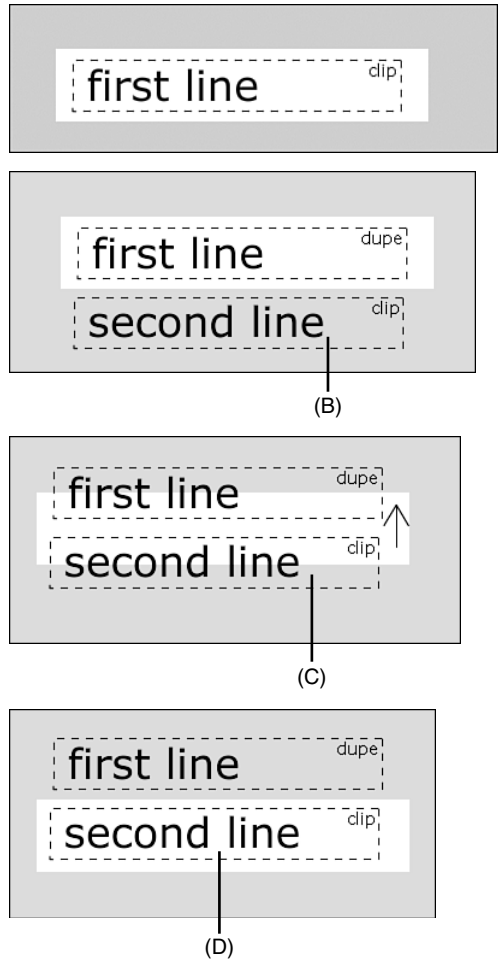
LISTING 3.5 This Code Displays Captions by Moving the Old Captions Offstage

```
1   function clear(){
2      clip._txt.text = "";
3   }
4   clear();
5
6   var dupe:MovieClip = clip.duplicateMovieClip( "dupe", 0 );
7   dupe._txt.text = clip._txt.text;
8
9   var initialLocation = clip._y;
10
11  function showText( name:String,
                       wholeObject:Object,
                       speed:String ){
12     if(speed == "fast"){
13        var duration = 0.2;
14     }else{
15        var duration = 0.5;
16     }
17     dupe._txt.text = clip._txt.text;
18     clip._txt.text = name;
19     var endTop =    initialLocation - clip._height;
20     var endBottom = initialLocation + clip._height;
21
22     clip._y = initialLocation;
23
24     new mx.transitions.Tween(dupe, "_y",
25               mx.transitions.easing.Regular.easeOut,
26               initialLocation, endTop, duration, true );
27
28     new mx.transitions.Tween(clip, "_y",
29               mx.transitions.easing.Regular.easeOut,
30               endBottom, initialLocation, duration, true );
31  }
32
33  function getSize():Object{
34     return { width:320, height:50 };
35  }
```

As different as this code seems from the original `captionType1.fla`, it's essentially doing the same thing: It's code to clear the text, code to show new text, and code that returns the stage size. Let's walk through it because it is more involved than the original. The `clear()` function is nearly identical as before, but notice that it's clearing the `text` property of the `_txt` instance nested inside the instance `clip`. Lines 6 and 7 create a duplicate of the clip so the user will see two blocks of text animate: the old text (in the `dupe` instance) going up offscreen and the new block appearing from the bottom. The `initialLocation` variable simply saves a reference to the default location for the clip with text.

Inside `showText()` is where most of the work is done. First, notice that this time we do use the third parameter (`speed`) and set a local variable, `duration`, accordingly (lines 12–16). You'll see how the `wholeObject` parameter is used in the next project, but because we want access to the third parameter, we need to leave `wholeObject` in line 11. The animation sequence goes like this: Copy the text from `clip` into `dupe` (line 17), put the new text (`name`) into `clip` (line 18), figure out the destination for text moving offscreen and ending at the top (line 19), figure the starting location below the stage for text moving up (line 20), make sure `clip` is in its initial location (line 22), and then create a new `mx.transitions.Tween()` for both `dupe` (lines 24–26) and `clip` (lines 28–30). It's easiest if you can visualize new text arriving onstage (in `clip`) and old text moving off (in `dupe`), as Figure 3.18 shows.

FIGURE 3.18 **When a new line of text arrives, the `dupe` clip is placed onscreen (with `clip`'s old text) and `clip` is moved offscreen to the bottom (b). Then they both move up (c). When the animation is over, `dupe` is offscreen and the new text appears on `clip`, which is in place (d).**



STEP 4▼
Publishing and Testing

While in your `captionText2.fla` file, select Control, Test Movie to produce `captionText2.swf`. You'll just get a blank `.swf` for now, which you can close. Reopen

one of your main files (`main.fla` or `main_audio.fla`) and change the first parameter in the `init()` method on the instance of the `CaptionHolder` symbol. You want to point to `captionText2.swf` and not to `captionText1.swf`. You'll see that change in line 7 of Listing 3.6.

LISTING 3.6 This Code Uses `captionType2.swf` Instead of `captionType2.swf`

```
1   var playback:mx.video.FLVPlayback;
2   playback.autoPlay = false;
3   playback.contentPath = "the_children_must_learn.flv";
4
5   var myEventChannel:EventChannel = new EventChannel();
6
7   var url = "captionType2.swf";
8   var eventList = ["actionscript"];
9   captions_clip.init(url, myEventChannel, eventList);
10
11  myEventChannel.init(playback, "video_captions.xml");
```

Here's one last touch before you test: Select Modify, Document and set the frame rate to 31. This makes the `mx.transitions.Tween()` methods appear much smoother. Finally, make sure all the support files, such as the actual video, class files, and `video_captions.xml` data, are present. Then select Control, Test Movie.

The captions seem to appear from below and roll up offscreen as they depart. Plus, while you're scrubbing, the captions still animate but much more quickly.

STEP 5▼
Adding Code for the Hide/Reveal Feature

To support the hide/reveal feature, go to `captionText2.fla`, select the first keyframe, open the Actions panel, and add this code below all the existing code:

```
var showing = true;
var owner = this;
hide_btn.onPress=function(){
   var duration = 1;
   if(showing){
      //move down
      dupe._txt.text = "";
      showing = false;
      var destination = getSize().height;
      var startPosition = 0;
   }else{
      //move up
      showing = true;
      var destination = 0;
      var startPosition = getSize().height;
   }
   new  mx.transitions.Tween(      owner,
                                    "_y",
   mx.transitions.easing.Regular.easeOut,
                            startPosition,
                              destination,
                                 duration,
                                     true );
}
```

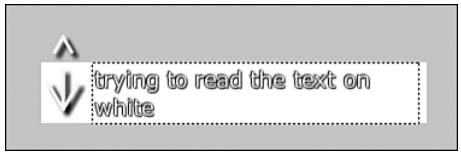
We'll create the `hide_btn` next. Anytime the user clicks this button, the entire caption

template (`owner`) moves down or back up again.

STEP 6▼
Creating the Hide/Reveal Button

Inside `captionText2.fla`, draw a down arrow that is the full height of the stage. Above it, draw an up arrow, but make sure it's just above the stage (but not on stage at all) as Figure 3.19 shows. Select both arrows and select Modify, Group so they don't get wiped away.

FIGURE 3.19 **The user will see only one arrow at a time (currently the arrow pointing down). After everything moves down, she'll see the up arrow.**



Nudge the `clip` instance over to the right if you need to make room for the arrows. You might also need to go inside the `clip` symbol to change the text margins or to modify the stage size. If you do change the stage size, remember to update the `width` and `height` properties in the object returned by the `getSize()` function that appears in the first frame's ActionScript.

Finally, draw a rectangle that's large enough to cover the two arrows. Convert the rectangle shape to a Button symbol by selecting it, pressing F8, and selecting the Button behavior. Name the new button symbol **Invisible** and click OK. Next, double-click the Invisible symbol and click once on the first keyframe; then click and drag the keyframe to the Hit frame. The button's timeline should look like the one in Figure 3.20.

FIGURE 3.20 **A Button symbol with nothing in any frame except the Hit frame will be invisible to the user but will remain clickable.**

Return to the main timeline of `captionText2.fla` and be sure to give the Invisible symbol an instance name of **hide_btn** to match the code you added in step 5.

Select Control, Test Movie and then go back to and test the main file. Because only the template has changed, you could instead simply double-click the `main.swf` generated the last time you tested.

Project:
Synchronized Images Template

This project is simply another template that works with all the code you've produced so far. Specifically, the `CaptionHolder.as` class loads the `.swf` file you'll create in this project just like it does `captionText1.swf` and `captionText2.swf`. However, the purpose of this template isn't solely for captions. Rather, you'll specify frame labels that contain images, or any Flash graphic or animation, that you want to appear at synchronized times during the video. This project also incorporates the option of adding parameters to cue points (discussed earlier) to stuff even more information into each cue point.

STEPS▼

1. **Encoding the video**
2. **Preparing to gather cue points**
3. **Creating an animated sequence**
4. **Gathering the cue points**
5. **Implementing the ActionScript in** `imagesTemplate.fla`
6. **Entering the ActionScript in** `main_coffee.fla`
7. **Testing** `main_coffee.fla`

STEP 1▼
Encoding the Video

Create a new file in Flash and save it as **main_coffee.fla**. Select File, Import to Stage and then select the `coffee_house_1969.mov` public domain excerpt, located in the `source_media` folder. Step through the video import process, accepting all the defaults to produce `coffee_house_1969.flv`. You won't be injecting Event or Navigation cue points during this phase.

STEP 2▼
Preparing to Gather Cue Points

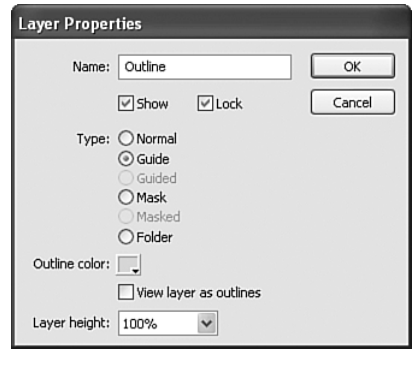
Select Control, Test Movie and then click Play to watch the video a few times, noting when the subject says the following phrases:

- "Circumscribed clientele"
- "cool people on campus"
- "not meaning derogatorily"
- "swinging" (and then "swinging" again)
- "coffee house"
- "place to be and the place to be seen"

STEP 3▼
Creating an Animated Sequence

Create a new file and save it in your working directory as **imagesTemplate.fla**. Select Modify, Document and set both the width and height to **250**. Select Insert, New Symbol. Make sure that the Movie Clip option is selected and name the symbol **Content**. Click OK and you'll be inside this clip. To see the edges of the stage, draw a rectangle that's 250 × 250 pixels. Position the rectangle inside the Content symbol so the upper-left corner of the rectangle aligns with the center of the Content symbol. Select Modify, Timeline, Layer Properties and then name the layer **Content**, click the Lock option, and change the layer type option to Guide (see Figure 3.21). Because we want the outline to guide us over several frames, click the cell in frame 25 and press F5 to insert frames.

FIGURE 3.21 **The Outline layer simply helps you see the edge of the stage from inside the** `Content` **symbol.**



Here's where you can get creative. We'll create a graphic or an animation to correspond with each of the phrases the nice young man says. Open the file `imagesTemplate.fla` in the `finished_source` folder and investigate the contents of the `Content` symbol to see how I created the sequence you're about to create.

Create a new layer for the still images; name this new layer **Stills**. Insert a keyframe into frame 2 (press F6), and then import a photograph or create a graphic to support the phrase *circumscribed clientele*, such as a circle around a drawing of a student from the '60s. Click frame 3 and then select Insert, Timeline, Blank Keyframe (or press F7). Insert a graphic to support the word *cool*. In frame 4, insert a blank keyframe and place the following text: **Derogatorily**. To animate the international "no" sign on top of it, select Insert, Timeline, Layer (name the layer **No sign**). In the No sign layer, click frame 5 and insert a blank keyframe (press F7). Draw a circle with a line through it, convert the circle to a movie clip by selecting it and pressing F8, and name it **no**. In frame 10 of the No sign layer, insert a keyframe. Open the Actions panel and type **stop();**, and in frame 11 of the No sign layer, insert a blank keyframe (F7). Return to frame 5 and, using the Properties panel, select Motion Tween. Then, select the no symbol onstage in frame 5 and use the Properties panel to set a Color style of Alpha to 1%. (The no sign will fade on from frame 5 to frame 10.)

For the two mentions of the word *swinging*, we'll show a graphic of a swing and then show that swing in motion. In frame 11, draw the swing. Next, in frame 12, start an animation of that swing that continues to frame 20, where you'll place a keyframe and a `stop()` action.

For the last two phrases (*coffee house* and *place to be and the place to be seen*), just place a photo or logo from your favorite coffee shop (in frame 21) and the text **be and be seen** in frame 22.

NOTE

You really can do anything you want, but I wanted some of the phrases to be supported with an animation and some with a still frame. In fact, there are a million other ways to do this, such as putting everything in single frames and on some frames placing a movie clip that contains multiple frames and a `stop()` on its last frame.

STEP 4▼
Gathering the Cue Points

Open `gathering_tool.swf` and enter the path to `coffee_house_1969.flv` into the Media field—don't worry about loading captions this time. Play the video and click the Add button each time the man says one of the previous phrases. You can remove cue points you added by mistake by clicking their triangles and then clicking the Remove button. Each cue point's name defaults to New Cue Point. Go through the added cue points by clicking each triangle or clicking the Prev or Next button; then enter a simple name for each one, such as **Circumscribed**, **Cool**, and so on. The user won't ever see this text the way we're building this project, but it helps to confirm that the cue points are in the right place.

When the cue points are in place, we'll add parameters to each one, such as a frame

number and an option of whether the animation should stop there or play. I came up with the idea that each cue point will have parameters for `frame` (the frame number to jump to in the `Content` symbol) and `option` (either `play` or `stop` meaning that when the user jumps to the frame, it will begin to play or simply stop there). Click the first cue point and click the plus button to add a parameter. Set the name to **frame** and the value to **2**. This means when this cue point is reached, we'll jump to frame 2 in the `Content` symbol, where the "Circumscribed Clientele" graphic appears. Click the plus button again and name the second parameter **option**; then set the value to **stop** (see Figure 3.22). The plan is that, when the user jumps to that frame, it will stop there.

Go through all the cue points so each one has parameters for `name` and `option`, as shown in the following table:

Name	Parameters	
Circumscribed	frame option	2 stop
Cool	frame option	3 stop
Derogatorily	frame option	4 play
Swinging (1)	frame option	11 stop
Swinging (2)	frame option	12 play
Coffeehouse	frame option	21 stop
Place to be	frame option	22 stop

After you have all the cue points and parameters set, click the Export XML button. Copy the XML string that appears and create a new text document with Notepad or a similar program. Next, paste in the XML text you copied and save the file as **coffee_house.xml** in your working directory.

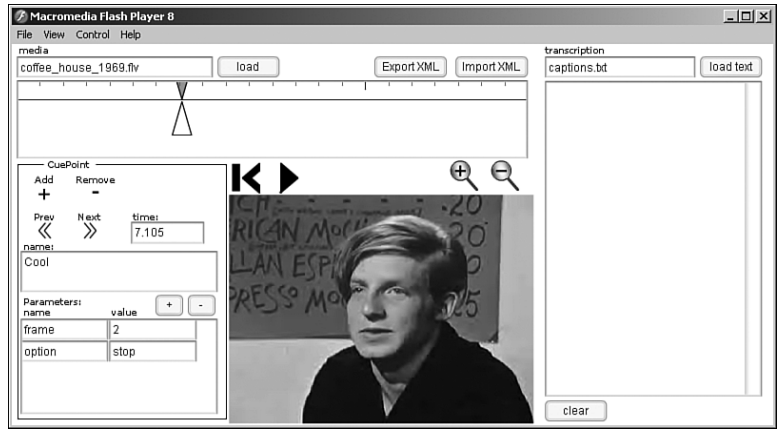


FIGURE 3.22 **This cue point has two parameters:** `frame (2)` **and** `option (stop)`**.**

STEP 5▼
Implementing the ActionScript in imagesTemplate.fla

Go to the main timeline of `imagesTemplate.fla`. Drag an instance of the Content symbol onto the stage if you haven't done so already. (It might be a bit tricky considering nothing is in the first frame of the `Content` symbol.) Use the Properties panel to set the instance name to **content** and the upper-left corner to **0, 0**.

Select the first keyframe, open the Actions panel, and type the code in Listing 3.6.

Notice that we completely ignore the first parameter received in `showText()` (that is, `name`). Instead, the code digs into `wholeObject`, which includes all the same properties that would be received from a standard `cuePoint` event. In this case, these properties are `type`, `target`, and `info`—inside of which are properties for `name`, `time`, and `parameters` (which itself contains whatever properties you injected into the `.flv` file or specified in the XML file). Here the code just grabs `info.parameters.frame` and `info.parameters.option`.

STEP 6▼
Entering the ActionScript in main_coffee.fla

Inside `main_coffee.fla`, be sure you have an `FLVPlayback` component onstage; if not, drag one from the Components panel. Give it an instance name of **playback**.

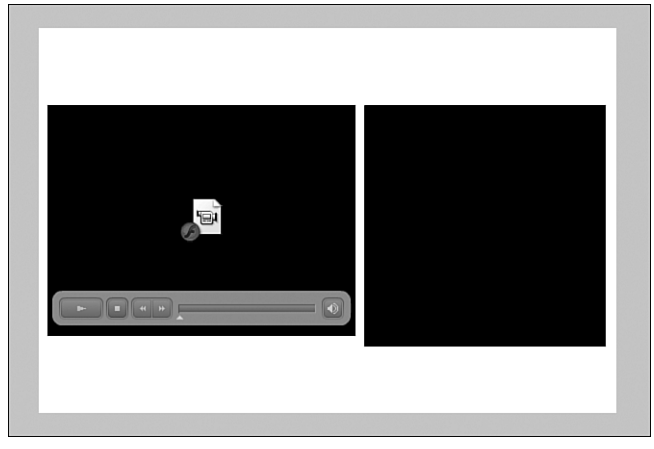
LISTING 3.6 This Code Handles New Captions by Jumping to the Appropriate Frame in the `content` Instance

```
function clear(){
   content.gotoAndStop(1);
}
clear();

function showText(name:String, wholeObject:Object, speed:String){
   content.gotoAndStop(wholeObject.info.parameters.frame);

   if( wholeObject.info.parameters.option == "play" ){
      content.play();
   }
}

function getSize():Object{
   return { width:250, height:250 };
}
```

Copy a CaptionHolder symbol from one of the other main files you've created (main.fla or main_audio.fla). Or simply select Insert, New Symbol and set the Create New Symbol dialog box as was shown previously in Figure 3.13 (step 2 of the fourth project). Place an instance of the CaptionHolder symbol onstage and give it an instance name of **captions_clip**. Arrange the screen so captions_clip is next to the video and not on top of it. The 320 × 50 rectangle shape in the CaptionHolder symbol is not an accurate representation of where the imagesTemplate.swf will appear because that file is actually 250 × 250. You can double-click the CaptionHolder and double-click again so you're inside the Rectangle symbol and then resize that shape to make it 250 × 250. Back in the main timeline, you should select Modify, Document and increase the main_coffee.fla file's width to at least 600 so the FLVPlayback component (instance name playback) and the CaptionHolder symbol (instance name captions_clip) fit side-by-side, as shown in Figure 3.23.

Finally, select the first keyframe, open the Actions panel, and type in the code in Listing 3.7.

FIGURE 3.23 **The FLVPlayback (left) and CaptionHolder (right) are arranged side-by-side.**



LISTING 3.7 **This Code Associates an Instance of the EventChannel Class with the imagesTemplate.swf File**

```
var playback:mx.video.FLVPlayback;
playback.autoPlay = false;
playback.contentPath = "coffee_house_1969.flv";

var myEventChannel:EventChannel = new EventChannel();

//send init() function to the CaptionHolder instance
//on stage (captions_clip)

var url = "imagesTemplate.swf";
var eventList = ["actionscript"];
captions_clip.init( url, myEventChannel, eventList );
```

continues

LISTING 3.7 Continued

```
//alternative code to effectively turn the FLVPlayback
//into autoPlay=true
function ready(){
   playback.play();
}
myEventChannel.addEventListener( "ready", this );
myEventChannel.init( playback, "coffee_house.xml" );
```

This code should look very familiar—the only changes are the filenames for the `.flv`, `imagesTemplate.swf`, and `.xml` files.

STEP 7▼
Testing main_coffee.fla

Make sure all the following support files are present in the same folder where your `main_coffee.fla` file is:

- `coffee_house_1969.flv`
- `coffee_house.xml`
- `imagesTemplate.swf`
- `CaptionHolder.as`
- `EventChannel.as`

That folder should also contain the FLVPlayback skin you selected, such as `MojaveOverNoVol.swf`.

In `main_coffee.fla` select Control, Test Movie.

I hope that this template shows you another way to use cue points for more than just captions and also gives you the ability to design and build additional template types. Like many projects in this book, extra templates are available for download; plus, I expect readers to share their templates.

Exploring the Support Classes

In each chapter in this book, I include this "Exploring the Support Classes" section as a behind-the-scenes look at the support files. You certainly don't have to study how everything was built, but I feel responsible to at least provide an architectural overview of all the code. If you're interested in adding features to this project or just want to learn more about how I chose to program this project, this section should be interesting.

I've produced the class diagram in Figure 3.24. Keep this figure handy as you read the following overview.

The two pieces we've built for the project in this chapter are the `main.swf` (far left) and the various `captionType.swf` templates (far right). We also created the external `.flv` and `.xml` files that get loaded at runtime. Notice that all `captionType.swf` templates *implement* the interface file named `ICaption.as`. There you'll see the three required methods each `captionType.swf` must implement. Recall, too, that inside our `main.swf` we

created a symbol called `CaptionHolder`, which was associated with the `CaptionHolder` class (center). The main thing the `CaptionHolder` class does is load the particular `captionType.swf` filename specified when the `main.swf` triggers the `init()` method on its instance of the `CaptionHolder` class (shown as `captions_clip`).

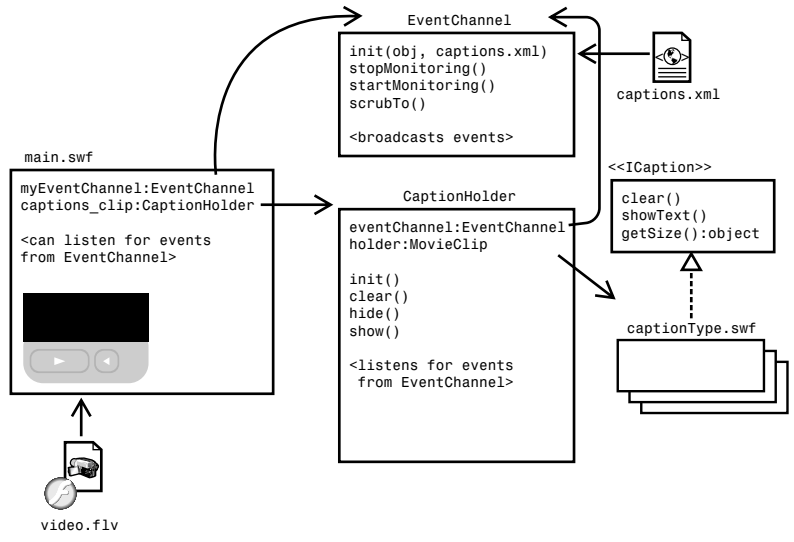


FIGURE 3.24 This diagram shows where your main file and `captionType` templates fit and how they relate to the `EventChannel` and `CaptionHolder` class files.

Before you call `init()` on the `CaptionHolder` instance, you must first create an instance of the `EventChannel` class. You'll see that the `CaptionHolder` class has an `EventChannel` instance (shown as a variable named `eventChannel` in the `CaptionHolder` class). In addition to loading the appropriate `captionType.swf`, the `CaptionHolder` also sets up listeners for events broadcast from the `EventChannel`. When the `EventChannel` broadcasts that a new caption should appear, `CaptionHolder` triggers the `showText()` method in the `captionType.swf` template.

The `EventChannel` does several things. First, it first loads an optional `.xml` file full of captions (if specified in `main.swf`). The `EventChannel` then sets up listeners for every event that can be broadcast from the media type you're using; this is because it supports the `FLVPlayback` or `MediaPlayback` components plus plain `Sound` objects. If you send a reference to an `FLVPlayback` component when you first call the `EventChannel`'s `init()` method, all the events for `FLVPlayback` components are listened for. If you pass a reference to the `MediaPlayback` component, a different set of events are listened for. This means you can use the `EventChannel` as a proxy for any media type. Normally, you'd use `addEventListener()` on an instance of the `FLVPlayback` component, but this way you can use `addEventListener()` on the `EventChannel` instance. This has several advantages: The `EventChannel` hijacks the `ready` event and fires that event only after the `.xml` is fully loaded (and therefore is *really* ready). The `EventChannel` also merges the cue points in the `.xml` into ActionScript cue

points. Finally, the `EventChannel` broadcasts a `cuePoint` event for regular Sound instances—something that's not built in to Flash's Sound class.

So, the `EventChannel` class *channels* all the event types that the different media types broadcast. Therefore, you can set up listeners if you want. In addition, the `CaptionHolder` instance is listening for all the possible events that should trigger the `showText()`—that is, so the text changes in your `captionType.swf` template.

The thinking behind the `EventChannel` class was that I wanted a unified way to listen for all the types of events related to captions. `FLVPlayback` components have a `cuePoint` event, but they're not the same as a `cuePoint` events broadcast from the `MediaPlayback` component. In addition, `.flv` files generated by Captionate broadcast events for `onCaption` and `onMarker`. Using the `EventChannel` means you don't have to think about how these syntaxes vary: You just channel them all to the `CaptionHolder` and the events you want to listen for are channeled to your `captionType.swf` display.

Final Thoughts

Now that you understand several kinds of cue points, you can select whichever one works best for your project. The framework built in this chapter lets you easily create new templates for displaying captions or responding to cue points by displaying graphics or animations. All you have to do is synchronize and design.

Let me give you a few more ideas of ways I've added synchronization beyond simply text captions. I built a kiosk for a history museum that included traditional captions of the narrator's script but that also had graphic highlights that appeared on a detailed map to supplement the audio discussion. In another project, I temporarily hid the video clip when the actor asked the user to interact with a survey question (built in Flash) that would appear in place of the video. There are so many more uses for captions and synchronization. As you can do for all the projects in this book, you can send me your ideas or any templates you build, and I can then share them with other readers on my website (www.phillipkerman.com/at-work/).