c6 CHAPTER

Flash deCONSTRUCTION

BILLABONG-USA.COM-INNOVATIVE ACTIONSCRIPT NAVIGATION

c6 CHAPTER

Flash deCONSTRUCTION

BILLABONG-USA.COM—INNOVATIVE ACTIONSCRIPT NAVIGATION

Billabong is a company that makes apparel for people who are into extreme sports and live very active lifestyles. The company's branding is not so much about certain colors and fonts; instead, it's about energy, attitude, and freshness. This branding was a driving force behind the interaction we built into the site.

Many sites I see on the web force interactivity onto a brand or content that has no real need for it. In fact, it works against that brand. Billabong-USA.com is a site that is as perfect a place as ever there was for dynamic and interesting interactivity.

In this chapter, we will be breaking down some of the interesting interactive pieces that work to make the site a total experience.

NOTE

Without the proper server and Generator environment, these files will not work when run from your desktop. However, we have provided files for you to explore and follow along with the examples.

NOTE

This chapter deals with Macromedia Generator application development. It requires that you have the Generator Extensions installed in Flash. If you do not have the extensions installed, you can download them for free from Macromedia's web site at www.macromedia.com/software/generator/download/extensions.html.

6.1 SKATE TEAM RIDERS

In the youth sports apparel market, one of the major marketing tactics used is building teams of riders in specific sports by offering sponsorships to these riders. Billabong has hundreds of pro and amateur sponsored riders spread across several sports. It is a symbiotic relationship because Billabong gives the riders gear and money, and the riders wear the gear to promote the brand. We learned firsthand in creating this site that these guys are always on the move and are hard to pin down at times. Nevertheless, they're always supporting the brand out in the water and on the streets.

6.1.1 NAME CLUSTERER

On the skate team rider page, we abstractly captured this dynamic of the riders through something we call the "name clusterer" or "the swarm." We wanted to display the riders' names in a dynamic and organic manner, as per the dynamics of the relationship we talked about previously. The concept we came up with was similar to a raging virus—a growing mass of graphical elements that spread across the screen in a random, frantic manner, as shown in Figure 6.1.

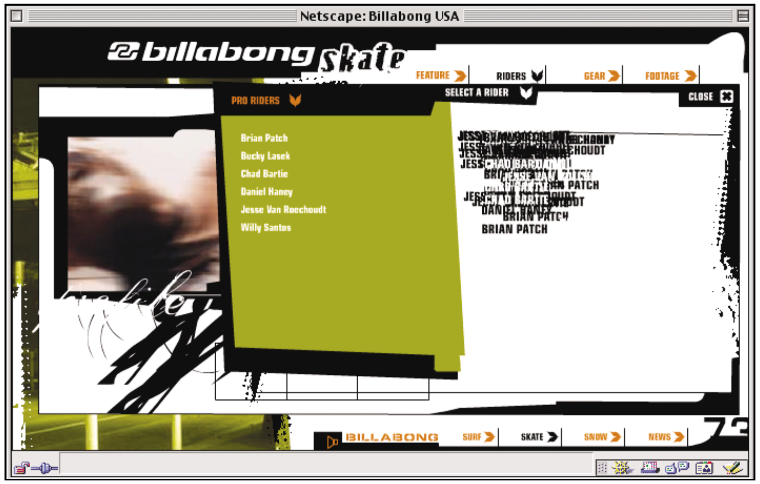
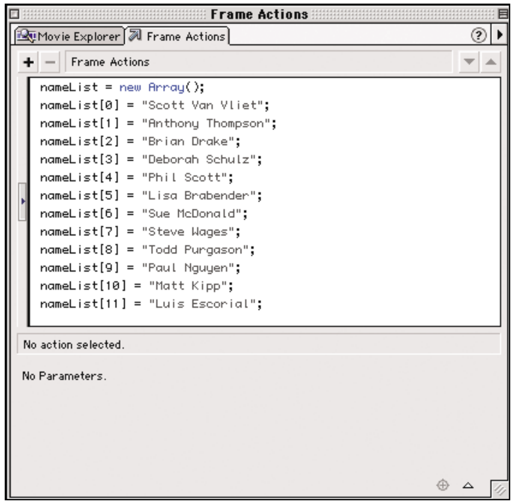


Figure 6.1 The name clusterer on the riders screen on Billabong-USA.com.

Open the file namecluster.fla (see Figure 6.2). We've isolated the clusterer to demonstrate how it was done.

Start by looking at the actions on the main timeline:

```
nameList = new Array();
nameList[0] = "Scott Van Vliet";
nameList[1] = "Anthony Thompson";
nameList[2] = "Brian Drake";
nameList[3] = "Deborah Schulz";
nameList[4] = "Phil Scott";
nameList[5] = "Lisa Brabender";
nameList[6] = "Sue McDonald";
nameList[7] = "Steve Wages";
nameList[8] = "Todd Purgason";
nameList[9] = "Paul Nguyen";
nameList[10] = "Matt Kipp";
nameList[11] = "Luis Escorial";
```



```
nameList = new Array();
nameList[0] = "Scott Van Vliet";
nameList[1] = "Anthony Thompson";
nameList[2] = "Brian Drake";
nameList[3] = "Deborah Schulz";
nameList[4] = "Phil Scott";
nameList[5] = "Lisa Brabender";
nameList[6] = "Sue McDonald";
nameList[7] = "Steve Wages";
nameList[8] = "Todd Purgason";
nameList[9] = "Paul Nguyen";
nameList[10] = "Matt Kipp";
nameList[11] = "Luis Escorial";
```

No action selected.

No Parameters.

Figure 6.2 Script on a frame in the main timeline in namecluster.fla.

In the Billabong-USA site, we obtained the skate riders' names from the database. In this similar example, we've created an array called nameList populated by the names of a few Juxt Interactive employees.

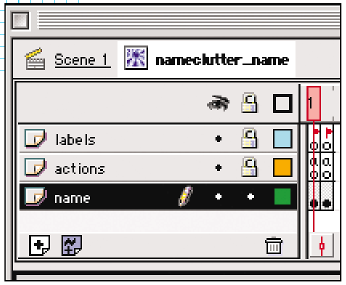
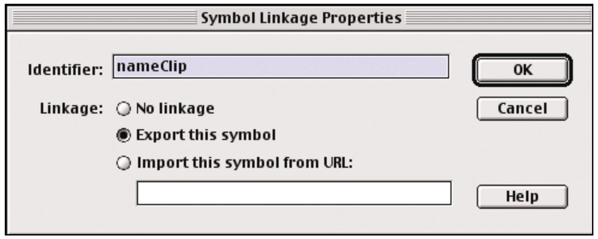


Figure 6.3 The nameclutter_name movie clip-editing mode.

Now that we've set up the names we'll use, we'll look at the symbol nameclutter_name in the library (see Figure 6.3). We've set up this movie clip with two frames labeled "black" and "white." Each frame has a stop action to keep the clip from looping. There's a dynamic text field called displayName that, if you look at the character palette, is set to black text on the black frame and white text on the white frame.

In the Billabong-USA site, we used a lot of 2-bit, black and white bitmaps to create a gritty design aesthetic. For this name clusterer, we chose black and white because it was consistent with the site's feel.



Identifier: nameClip

Linkage: No linkage [✓ Export this symbol] Import this symbol from URL:

Figure 6.4 The Symbol Linkage Properties dialog box for the nameclutter_name movie clip.

Notice that this clip isn't actually placed in the stage anywhere. Examine the linkage properties for this clip to see that we're exporting it with an identifier of nameClip (see Figure 6.4).

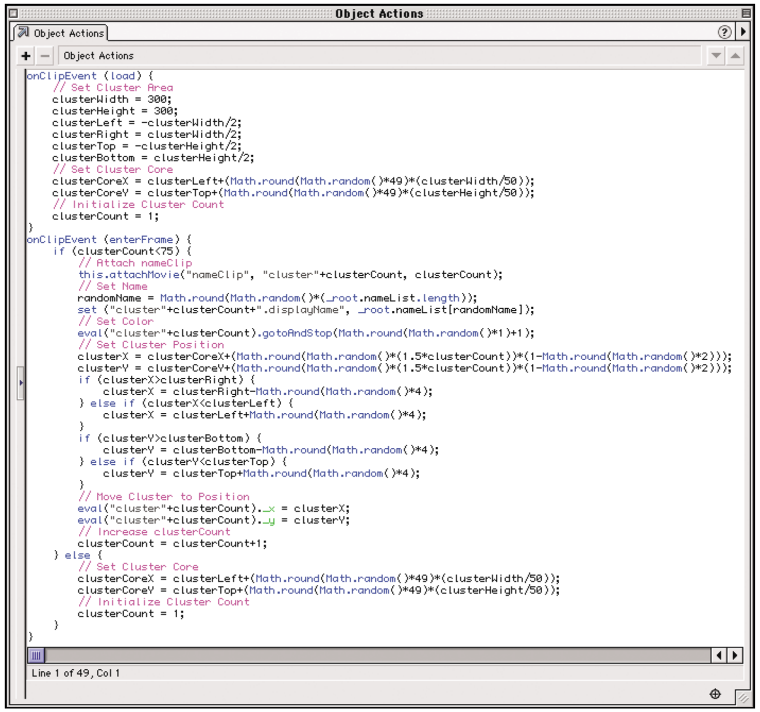
Now look at the action clip on the name clusterer layer. You'll see later that we've written the cluster script to cluster around wherever this clip is placed. So we've placed it in the center of the stage. Look at the clipEvent on this movie clip (see Figure 6.5):

```
onClipEvent (load) {
   // Set Cluster Area
   clusterWidth = 300;
   clusterHeight = 300;
   clusterLeft = -clusterWidth/2;
   clusterRight = clusterWidth/2;
   clusterTop = -clusterHeight/2;
   clusterBottom = clusterHeight/2;
   // Set Cluster Core
   clusterCoreX = clusterLeft+(Math.round(Math.random()*49)*(clusterWidth/50));
   clusterCoreY = clusterTop+(Math.round(Math.random()*49)*(clusterHeight/50));
   // Initialize Cluster Count
   clusterCount = 1;
}
onClipEvent (enterFrame) {
   if (clusterCount<75) {
      // Attach nameClip
      this.attachMovie("nameClip", "cluster"+clusterCount, clusterCount);
      // Set Name
      randomName = Math.round(Math.random()*(_root.nameList.length));
```

continues

continued

```
        set ("cluster"+clusterCount+".displayName", _root.nameList[randomName]);
        // Set Color
        eval("cluster"+clusterCount).gotoAndStop(Math.round(Math.random()*1)+1);
        // Set Cluster Position
        clusterX = clusterCoreX+(Math.round(Math.random()*(1.5*clusterCount))*(1-Math.round(Math.random()*2)));
        clusterY = clusterCoreY+(Math.round(Math.random()*(1.5*clusterCount))*(1-Math.round(Math.random()*2)));
        if (clusterX>clusterRight) {
            clusterX = clusterRight-Math.round(Math.random()*4);
        } else if (clusterX<clusterLeft) {
            clusterX = clusterLeft+Math.round(Math.random()*4);
        }
        if (clusterY>clusterBottom) {
            clusterY = clusterBottom-Math.round(Math.random()*4);
        } else if (clusterY<clusterTop) {
            clusterY = clusterTop+Math.round(Math.random()*4);
        }
        // Move Cluster to Position
        eval("cluster"+clusterCount)._x = clusterX;
        eval("cluster"+clusterCount)._y = clusterY;
        // Increase clusterCount
        clusterCount = clusterCount+1;
    } else {
        // Set Cluster Core
        clusterCoreX = clusterLeft+ (Math.round(Math.random()*49)*(clusterWidth/50));
        clusterCoreY = clusterTop+(Math.round(Math.random()*49)*(clusterHeight/50));
        // Initialize Cluster Count
        clusterCount = 1;
    }
}
```



```
onClipEvent (load) {
    // Set Cluster Area
    clusterWidth = 300;
    clusterHeight = 300;
    clusterLeft = -clusterWidth/2;
    clusterRight = clusterWidth/2;
    clusterTop = -clusterHeight/2;
    clusterBottom = clusterHeight/2;
    // Set Cluster Core
    clusterCoreX = clusterLeft+(Math.round(Math.random()*49)*(clusterWidth/50));
    clusterCoreY = clusterTop+(Math.round(Math.random()*49)*(clusterHeight/50));
    // Initialize Cluster Count
    clusterCount = 1;
}
onClipEvent (enterFrame) {
    if (clusterCount<75) {
        // Attach nameClip
        this.attachMovie("nameClip", "cluster"+clusterCount, clusterCount);
        // Set Name
        randomName = Math.round(Math.random()*(_root.nameList.length));
        set ("cluster"+clusterCount+".displayName", _root.nameList[randomName]);
        // Set Color
        eval("cluster"+clusterCount).gotoAndStop(Math.round(Math.random()*1)+1);
        // Set Cluster Position
        clusterX = clusterCoreX+(Math.round(Math.random()*(1.5*clusterCount))*(1-Math.round(Math.random()*2)));
        clusterY = clusterCoreY+(Math.round(Math.random()*(1.5*clusterCount))*(1-Math.round(Math.random()*2)));
        if (clusterX>clusterRight) {
            clusterX = clusterRight-Math.round(Math.random()*4);
        } else if (clusterX<clusterLeft) {
            clusterX = clusterLeft+Math.round(Math.random()*4);
        }
        if (clusterY>clusterBottom) {
            clusterY = clusterBottom-Math.round(Math.random()*4);
        } else if (clusterY<clusterTop) {
            clusterY = clusterTop+Math.round(Math.random()*4);
        }
        // Move Cluster to Position
        eval("cluster"+clusterCount)._x = clusterX;
        eval("cluster"+clusterCount)._y = clusterY;
        // Increase clusterCount
        clusterCount = clusterCount+1;
    } else {
        // Set Cluster Core
        clusterCoreX = clusterLeft+(Math.round(Math.random()*49)*(clusterWidth/50));
        clusterCoreY = clusterTop+(Math.round(Math.random()*49)*(clusterHeight/50));
        // Initialize Cluster Count
        clusterCount = 1;
    }
}
```

Figure 6.5 The script on the name clusterer layer on the main timeline.

First, with the onClipEvent(load), we initialize the script. We'll start by determining the cluster area. We've picked 300 width and 300 height.

We then set variables for the left, right, top, and bottom of the cluster area.

Next we'll set the "cluster core." This is the point from which the cluster will form. For the X and Y position of this core, we divide the cluster area by 50 and then randomize which 50th of the area we'll place the cluster at. You can change out 50 with a higher or lower number, depending on the density of the possible spots you want the cluster to grow out of.

Lastly, we set the clusterCount to 1.

The rest of the actions are contained in an onClipEvent(enterFrame), so they'll be continually looped. First we'll determine if the clusterCount is less than 75. This means that after there are 75 names, the cluster will move to another position and start building again. If the count is less than 75, we'll add names. We start by attaching the nameClip that we previously set to export. Next we'll choose which name from the nameList array we'll display. The variable randomName is a number chosen from the length of the array and then used to transfer the value of a name from the array to the displayName variable in the current cluster clip.

Now we'll set the color of the cluster. As we discussed before, this can either be black or white. By putting gotoandStop(random(2)+1), we'll send the cluster to either frame 1 or frame 2.

Now we'll determine where the current cluster will be positioned. We'll set two position variables, clusterX and clusterY. Let's examine the logic behind clusterX and clusterY, which is the same. We start at the clusterCoreX, which is the center of the cluster. The (1-random(3)) will return either –1, 0, or 1. This will take the result of random(1.5*clusterCount) and either make it negative, leave it positive, or counteract it by setting the X position to the core X position. We chose 1.5 because it fit within our usage. If you use a higher number, the clusters will be spaced out farther. The inverse is also true.

Before we move the cluster to the new X and Y positions, we need to check whether the new position is within the cluster area we initially set through a series of if statements. If the cluster will be out of the cluster area, we manually reposition the X and Y coordinates to the boundary of the cluster area and then break up the numbers with a little randomization.

Finally, we move the cluster to the X and Y positions we just set.

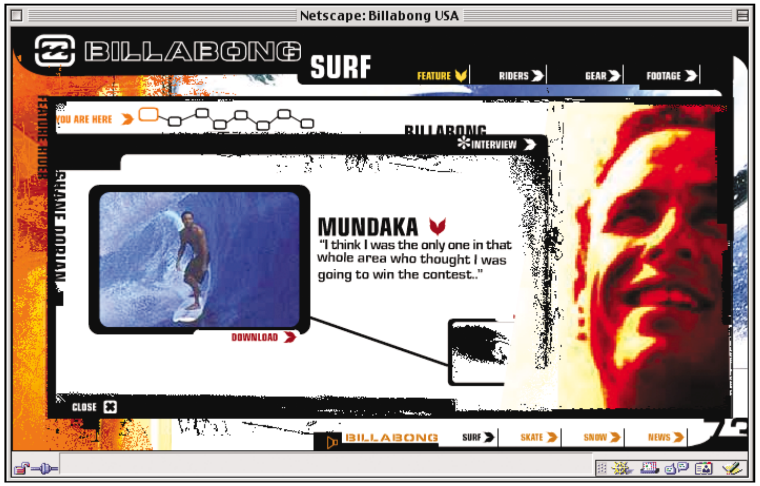
Now we come back to the initial if statement. The preceding actions were to be run if the clusterCount was less than 75. Once the clusterCount exceeds 75, we rerun the initialization script, which randomizes the position of a new "core" and sets the count back to 1.

Now we start again. Notice that we don't remove the clips we attached before starting again. In Flash, only one movie clip can occupy a specific "depth." By restarting to attach the movie clips, we "eat away" at the first cluster by attaching clips to depths that contain the nameClips from the first time the script ran. We intentionally set up the script this way because we were pleased with the way the clusters grew across the screen while slowly "dissolving" after a while.

6.2 BILLABONG-USA.COM SURF FEATURED RIDER

One of the sections of the Billabong-USA.com site that afforded us the most creative experimentation was the featured rider section. Designed to be a frequently updated section of the site, each new featured rider for the three sports received a new design treatment and content navigation experimentation. We used several interesting techniques for each featured rider section. Now we'll look at the Featured Surf Rider for Shane Dorian.

For this feature, we were conceptually playing with the dynamics of the fluid motion of surf (see Figure 6.6). The content comes into the screen much like swells come to a beach. They roll in as they get closer to shore. They peak, break, and slide back into the sea. So, in this feature, the content rolls to a peak and then falls back down and out of the scene.



"I think I was the only one in that whole area who thought I was going to win the contest."

Figure 6.6 A Featured Rider screen on the Billabong-USA site.

6.2.1 FEATURED RIDER NAVIGATION

Open the file featuredrider.fla. We'll first look at the content navigation system we devised for this particular rider spotlight.

Because this is a surf rider, we tried to develop a navigation system that loosely simulated the feel of a wave. Once the concept was finalized, there were several approaches we could have taken to accomplish the interaction we were looking for. We finally went with a hybrid approach of keyframed/tweened animation and ActionScript because we felt some of the animation control could be done easier without complex mathematical scripts.

First we'll look at the symbol called content on the main timeline (see Figure 6.7). The movie clip is also given an instance name of content. Edit this symbol and observe the timeline. There's a lot of manual animation here. Scrub the timeline to see how we animated the content nodes in and out of the view area.

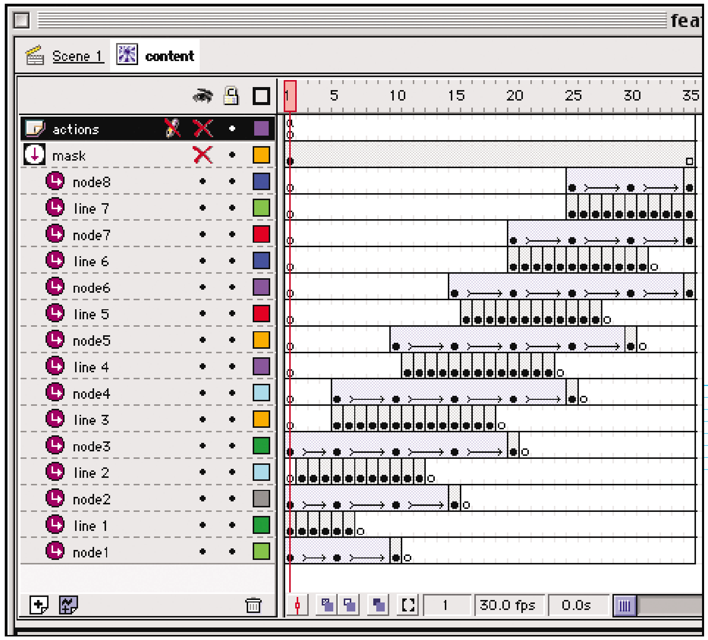


Figure 6.7 The movie clip named content on the main timeline.

There's an important issue to remember when you keyframe animation like this. Because the content movie's timeline needs to act in a fluid motion, its play movement could be ActionScripted to stop or start at intervals designated by the tween keyframes. At these tween points, the feature, or node, needs the ability to be ActionScripted to life. It is important to make sure that each keyframe of the tween retains its instance name, thus enabling it to receive actions at any of these points. For example, look at node1 on the layer node1 on all three keyframes. We've made sure the clip has an instance name of node1 on frames 1, 5, and 10.

We'll return to this clip later, but let's move on to the navigation. On the main timeline, there's a clip, featured_nav, with the instance name nav. Edit this symbol. You'll see we have eight copies of the movie clip featured_nav_node. Each has a unique instance name: node1, node2, and so on. Edit this symbol (see Figure 6.8).

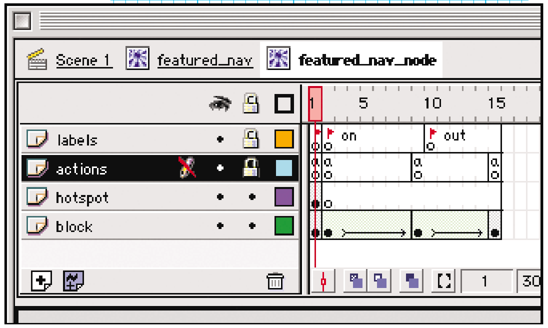
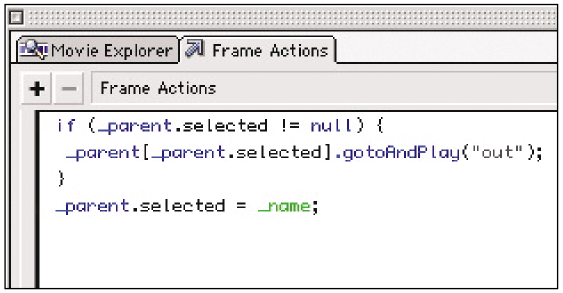


Figure 6.8 The featured_nav_node movie clip is duplicated eight times in the parent clip.

We've set up this clip with a growth animation starting on frame on and a shrink animation starting on frame off. The hotspot on the off frame tells the clip to gotoAndPlay("on"), as shown in Figure 6.9.

On the frame labeled on, we have the following actions:

```
if (_parent.selected != null) {
    _parent[_parent.selected].gotoAndPlay("out");
}
_parent.selected = _name;
```



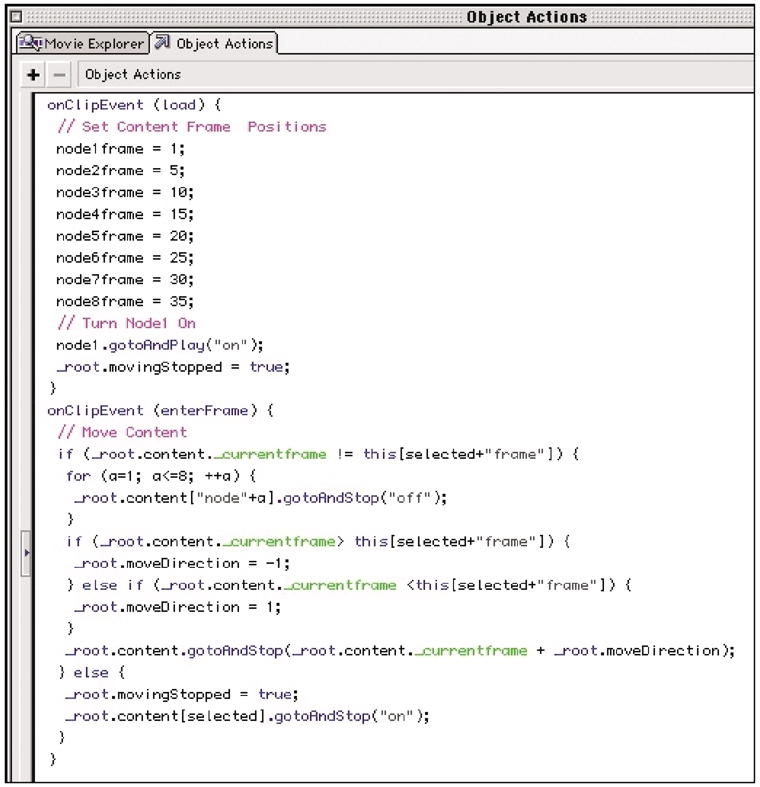
```
if (_parent.selected != null) {
 _parent[_parent.selected].gotoAndPlay("out");
}
_parent.selected = _name;
```

Figure 6.9 The script on the on frame in the featured_nav_node clip and the action on the button within the hotspot layer.

Because we only want one node to be designated as on at a time, we've set a variable called selected on the parent timeline (_root.nav) to the _name of the current on node. If this isn't the first time we've run this script (_parent.selected != null), we first tell the current selected node to go to the out sequence. We then set selected to the current node's _name.

Now that we've seen the structure of the navigation nodes, go back to the main timeline and view the clip events on the nav clip (see Figure 6.10).

```
onClipEvent (load) {
    // Set Content Frame  Positions
    node1frame = 1;
    node2frame = 5;
    node3frame = 10;
    node4frame = 15;
    node5frame = 20;
    node6frame = 25;
    node7frame = 30;
    node8frame = 35;
    // Turn Node1 On
    node1.gotoAndPlay("on");
    _root.movingStopped = true;
}
onClipEvent (enterFrame) {
    // Move Content
    if (_root.content._currentframe != this(selected+"frame")) {
        for (a=1; a<=8; ++a) {
            _root.content["node"+a].gotoAndStop("off");
        }
        if (_root.content._currentframe> this[selected+"frame"]) {
            _root.moveDirection = -1;
        } else if (_root.content._currentframe< this[selected+"frame"]) {
            _root.moveDirection = 1;
        }
        _root.content.gotoAndStop(_root.content._currentframe+
            _root.moveDirection);
    } else {
        _root.movingStopped = "true";
        _root.content[onNode].gotoAndStop("on");
    }
}
```



```
onClipEvent (load) {
 // Set Content Frame  Positions
 node1frame = 1;
 node2frame = 5;
 node3frame = 10;
 node4frame = 15;
 node5frame = 20;
 node6frame = 25;
 node7frame = 30;
 node8frame = 35;
 // Turn Node1 On
 node1.gotoAndPlay("on");
 _root.movingStopped = true;
}
onClipEvent (enterFrame) {
 // Move Content
 if (_root.content._currentframe != this[selected+"frame"]) {
  for (a=1; a<=8; ++a) {
   _root.content["node"+a].gotoAndStop("off");
  }
  if (_root.content._currentframe> this[selected+"frame"]) {
   _root.moveDirection = -1;
  } else if (_root.content._currentframe <this[selected+"frame"]) {
   _root.moveDirection = 1;
  }
  _root.content.gotoAndStop(_root.content._currentframe + _root.moveDirection);
 } else {
  _root.movingStopped = true;
  _root.content[selected].gotoAndStop("on");
 }
}
```

Figure 6.10 The clip events on the nav clip.

We start by initializing some settings with onClipEvent(load). The variables node(1-8) frame are the actual frame numbers in which the content is correctly positioned on the stage. Because the content starts out with content node1 focused on, we'll turn the navigation node1 on by telling it to gotoAndPlay("on").

And now for the actions that move the content. Because we want these actions to loop, we put them in an onClipEvent(enterFrame) located on the nav movie clip. We'll start by determining whether the content is at the correct position by comparing the current frame of the content clip against the result of the concatenation of the variable called selected (node1, node2, etc.) and the string called frame. The result of the [selected+"frame"] statement will look something like node1frame. If selected = node3, the concatenation of the variable called selected with the string frame gives us node3frame. If you recall, node3frame is a variable that has already been set to be equal to 10 within the onClipEvent(load) statement we used on the nav clip. To highlight the content for node3, the content movie clip's desired frame location would be at the value of node3frame, or in this case, 10. Using the brackets to concatenate and evaluate the variable selected with the string frame works well, but must be used with the content of a timeline identifier presented before the brackets. In addition, the dot notation usually found between two objects in a timeline reference is dropped directly before the bracket. For example, _root.content["node"+a] .gotoAndStop("off").

Next, we'll determine which "side" of the desired frame the content clip is on. If the current content frame is greater than the desired frame, we'll set the variable moveDirection to –1. Otherwise, we'll set the moveDirection to 1. Once we've determined which direction to move the content clip, we'll move the content to the currentframe plus the direction we've determined. This action will keep looping until the currentframe is the same as the desired frame. Once we've moved the content clip to the desired frame, we'll tell the correct content node to gotoAndStop("on").

Now we've set up the navigation. Go back to the content clip. Edit one of the content nodes (for example, content_node1). Look at the actions on the hotspot on the off frame:

```
on (rollOver) {
    if (_root.movingStopped == true) {
        _root.nav[this._name].gotoAndPlay("on");
        _root.movingStopped = false;
    }
}
```

Because each content node has an instance name of node1, node2, and so on, when we concatenate _root.nav with _name, Flash will return _root.nav.node1,2,3,etc. We tell this clip to gotoAndPlay("on"). By controlling the nav in this manner, the user can roll over either a content node or a navigation node to move the content. The if (_root.movingStopped == "TRUE") checks to make sure the content isn't currently animating. The if statement used here checks to make sure the content isn't currently animating. This prevents the clip action from happening more than once.

6.3 MOVIE CONTROL

In the Shane Dorian featured rider section, we kept the higher res color images and the two movies as external movie clips to help cushion the download time for the user. To demonstrate the controller, we developed for the movies. We've included one of the movies in the featuredrider.fla file.

Before we look at that file, we'll quickly discuss how to use video inside of Flash. As of Flash 5, you cannot directly play a QuickTime, AVI, or MPEG (and so on) movie inside your Flash file. There are times, however, when you need to utilize bitmap movies inside of Flash. To do this, you'll need to import your video as an image sequence. Motion pictures are basically many still images displayed rapidly in a sequence. With just about any 2D effects/compositing/NLE software available today (that is, After Effects, Premier, Final Cut Pro), you can export digital video to a folder of sequentially named, still images. See your software's manual for specific instructions on how to do this.

If you don't own any of this software, and your budget's not ample, a cheap tool that can do this function is QuickTime Pro, which sells for about $30 at www.apple.com/quicktime. QuickTime Pro can do a variety of tasks, including exporting your movies as image sequences. Regardless of what software you use, we recommend using no compression on the still images. It's advantageous to bring images into Flash as high-quality images and let Flash do the compression. This avoids even further loss of quality from redundant compression. Also, make sure your output files are named with a sequential numeric order. Most software will do this automatically or will offer it as an option.

One great feature of Flash is that it recognizes sequentially named image sequences. All you need to do is import the first image of a sequence, and Flash will recognize the other images. Flash will then prompt you to decide whether or not to import the whole sequence. If you select yes, Flash will create a new keyframe on the currently selected layer of your timeline for every image. Export this and—viola!—your video is playing in Flash. Keep in mind that Flash isn't meant for just displaying video, and you'll start fighting performance issues depending on the size of the video and how many frames it is. Remember, the user has to download every single still image, which can quickly add up.

One technique you can use to deal with this is to delete every other frame of video and leave one frame in between each of the remaining keyframes. The motion will start to be less smooth, but now your user only has to download half as many images. If this isn't enough, delete every other two frames. Experiment until you reach an acceptable equilibrium between quality of the motion and file size.

We used this technique for the two videos controlled in Flash. Let's go back to featuredrider.fla (see Figure 6.11).

Netscape: Billabong USA
BILLABONG SURF
FEATURE
RIDERS
GEAR
FOOTAGE
YOU ARE HERE
INTERVIEW
"It's nice to have such challenging waves, and it's nice to have to prove yourself in good conditions in order to win the title."
DRAG
CLOSE
SURF
SKATE
SNOW
NEWS

Figure 6.11 A screen from the feature rider page.

Open this file, edit the content clip, and then edit the symbol content_node3 (see Figure 6.12). You'll see that we structured the clip like the other content clips, with an off frame and an on frame.

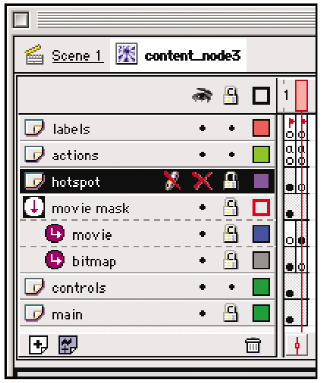


Figure 6.12 The timeline on the content_node3 clip.

On the movie layer, we've put the movie clip node3_movie with an instance name of surfmovie on the on frame. Edit this symbol. You'll see that we've imported an image sequence into a movie clip. Go back one level to the node3 clip.

We'll be concentrating on the movie clip content_video_controller, which is on the controls layer (see Figure 6.13). Edit this clip.

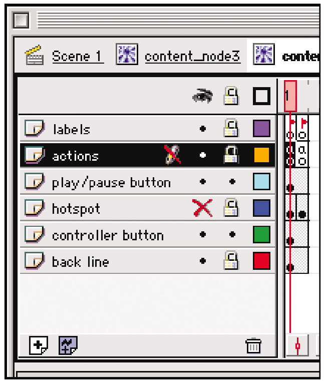


Figure 6.13 The timeline on the content_video_controller clip.

We've structured this clip with a "playing" frame and a "paused" frame. On the first frame, playing, there are the following actions:

```
stop ();
controller.gotoAndPlay("playing");
_parent.surfmovie.play();
```

We start by telling the controller clip (which we'll look at next) to gotoAndPlay("playing"). We also tell the surfmovie to start playing. On the paused frame:

```
stop ();
controller.gotoAndPlay("paused");
_parent.surfmovie.stop();
```

Look at the hotspot layer. The playing frame has a hotspot over the timeline graphic that tells the clip to go to the paused frame. On the paused frame, we have several hotspots with actions to tell the clip to go back to the playing frame.

Now edit the content_video_controller_button movie clip with an instance name of controller (see Figure 6.14).

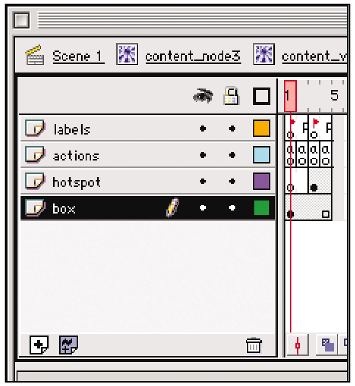


Figure 6.14 The timeline on content_video_controller_button.

You'll see that we've set up two, two-frame loops. The first loop starts at the playing frame. The first frame has the actions, and the second frame loops back to the first frame. The second loop starts at the paused frame. During the playing loop, the movie is playing, and we want the scrub bar to move in sync with the movie.

Look at the actions on the playing frame:

```
dragLeft = -32;
dragRight = 80;
frame = _parent._parent.surfmovie._currentframe;
step = (dragRight-dragLeft)/(_parent._parent.
  surfmovie._totalframes);
_x = (frame*step)+dragLeft;
```

First, we set the left and right limits of the scrub bar with the variables dragLeft and dragRight. These values were determined by manually moving the scrub bar to the left and right of the timeline and noting the X position.

We next set the frame variable to the currentframe of the surfmovie. Step is a variable determined by taking the drag area (dragRight-dragLeft) and dividing it by the number of frames in the surfmovie (_parent._parent.surfmovie._totalframes).

Lastly, we set the X position of the scrub bar to the frame variable multiplied by the step variable. We offset this variable by the left of the drag area. Now, on every loop, the scrub bar will move along with the surfmovie.

Now that we've set up the scrub bar to move along with the surfmovie, we need to set it up so that when the user rolls over the scrub area, he or she can drag the scrub bar and consequently the surfmovie.

Look at the actions on the paused frame.

```
_root.moviePosition = Math.round((_x-dragLeft)/step);
_parent._parent.surfmovie.gotoAndStop(_root.moviePosition);
```

First, we set the variable moviePosition. This is an inverse equation to the step variable we set in the previous loop. We limit this value to an integer. We tell the surfmove to gotoAndStop(_root.moviePosition).

Finally, look at the actions on the hotspot on the paused frame.

```
on (press) {
    startDrag ("", true, dragLeft, _y, dragRight, _y);
}
on (release) {
    stopDrag ();
}
```

Now the scrub bar is draggable and is constrained to the scrub area. When the user drags the scrub bar, it will evaluate its position and move the surf movie to the appropriate frame. This technique is scalable and can be used to control a movie clip of any length.

6.3.1 JAVASCRIPT POP-UP WINDOW

In our minds, the key to creating a good site is to use technology for its strengths and work around its weaknesses, if possible, with other technologies. We love Flash because it has so many strengths that work around the many weaknesses of HTML. However, Flash has its own weaknesses, and at times we use things like HTML, JavaScript, and QuickTime to provide solutions for those weaknesses. The key is to weave all these elements together to make a consistent presentation that feels seamless. You can opt to make every page a hybrid. In other words, a balance of HTML and Flash together on each page. Or you can isolate the unique elements of content that will need HTML and pop them up in JavaScript windows. This is the method we choose on the Billabong site because it allows the main focus of the content to reach a high level of concentration through Flash without the distracting breaks in experience that come with HTML page loads.

To open a new browser window with JavaScript, the window.open() method is used. The syntax of that JavaScript method is as follows:

```
window.open(theURL,winName,features);
```

The benefit of using JavaScript to open a new window rather than targeting a blank new window is that you are given control over the size and properties of the new window. To control the features, you would use a method similar to this example:

```
window.open('window_url.html','window_name','toolbar=no,
  location=no,status=no,menubar=no,scrollbars=no,
  resizable=no,width=400,height=400');
```

All of the features set to no can be set to yes. At Juxt, if a new window will be filled with Flash (such as Pickled.tv), we'll set status=yes and the other features to no. We do this so that the user can see the browser's status bar, which will display the download status of the files being loaded (see Figure 6.15).

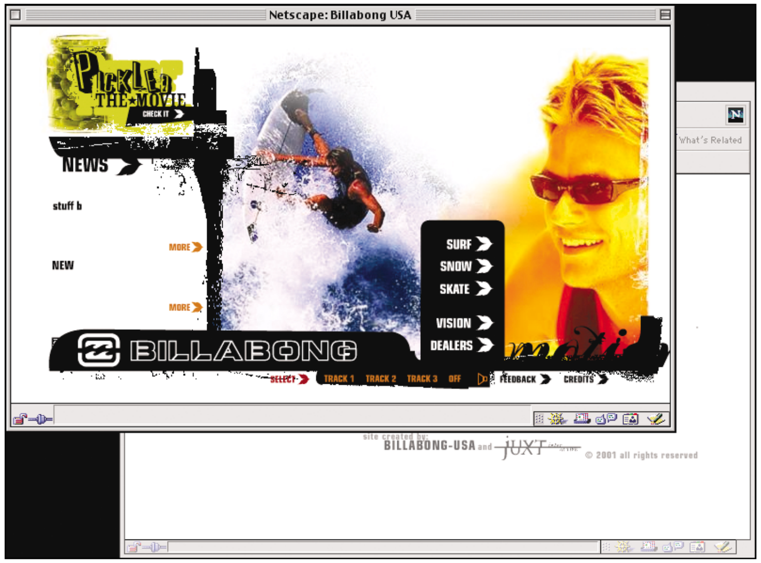


Figure 6.15 We set the status to yes for new windows opening with Flash so that the user can monitor the browser's status.

There are several ways to invoke this JavaScript method from Flash. The most compatible way is to define a JavaScript function in the HTML page where you've embedded your Flash file. Put this in the head of your document:

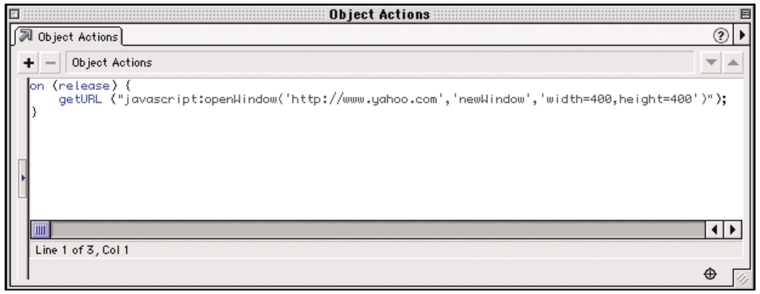
```
<SCRIPT LANGUAGE="JavaScript">
<!--

function openWindow(theURL,winname,features) {
window.open(theURL,winname,features);
}

//-->
</SCRIPT>
```

NOTE

This is the only method that works for Internet Explorer 4.5 on the Macintosh.

In Flash, create a button with a getUrl action that calls the JavaScript function (see Figure 6.16). For example:

```
on (release) {
   getURL ("javascript:openWindow('http://www.yahoo.com',
      'newWindow','width=400,height=400')");
}
```



```
on (release) {
    getURL ("javascript:openWindow('http://www.yahoo.com','newWindow','width=400,height=400')");
}
```

Figure 6.16 A button in Flash with a getUrl action that calls a JavaScript function.

There's another technique we use that involves a hidden frame. When a Flash site is contained in a pop-up window, usually the HTML file that the window contains is actually an HTML frameset with one or more hidden frames. A hidden frame is a frame that isn't allowed any viewable space. Here's a sample frameset for you to see this principle:

```
<html>
<head>
<title></title>
</head>

<frameset rows="100%,*" frameborder="NO" border="0"
  framespacing="0">
<frame src="maincontent.html" name="mainframe"
  marginwidth="0" marginheight="0" scrolling="NO"
  noresize frameborder="NO">
<frame src="blank.html" name="hidden" marginwidth="0"
  marginheight="0" scrolling="NO" noresize frameborder="NO">
</frameset>
<noframes><body bgcolor="#FFFFFF">

</body></noframes>
</html>
```

You'll see that we've given the mainframe frame100% of the viewable size, while the hidden frame is limited to whatever's left (*), which in this case isn't anything.

Through the browser, the only frame you will see is the mainframe. But the hidden frame is still there, and its existence allows us to load data into that frame without the user seeing it. We can use this hidden frame to launch JavaScript functions as well as opening JavaScript windows. Create an HTML file called launch_yahoo.html with the following contents:

```
<HTML>
<HEAD>

<TITLE></TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!--

function openWindow(theURL,winname,features) {
window.open(theURL,winname,features);
}

function openYahoo() {

openWindow('http://www.yahoo.com','yahooWindow','
  toolbar=yes,location=yes,status=yes,menubar=yes,
  scrollbars=yes,resizable=yes,width=400,height=400');
}

//-->
</SCRIPT>
</HEAD>
<BODY onLoad="openYahoo()">

</BODY>
</HTML>
```

In your Flash file, the getURL action will look like this:

```
on (release) {
    getURL ("launch_yahoo.html", "hidden");
}
```

When you invoke this action, the launch_yahoo.html file will be loaded into the hidden frame, and the JavaScript function will run, opening a new window.

6.4 EMBEDDING VIDEOS

Another technology that has strengths in its own right is video. Billabong has lots of great video footage that is great content for the site. As we showed you earlier in this chapter, you can export video as stills and import the stills into Flash as frames to simulate video. But this takes a fair amount of development time to do and would mean we would have to be involved every time Billabong wanted to add a video to the site. For the Footage sections of the site, it was a much better choice to rely on the widely distributed QuickTime and Windows Media players to augment the Flash experience. This enabled us to build a system for Billabong to upload videos at any time on their own without having to call on us. We will cover the content management aspect of this in the next chapter. For now, let's focus on the embedding video formats.

On the Billabong-USA site, we gave the user the option of viewing the surf/skate/snow videos in either QuickTime or Windows Media formats. Embedding these video formats in an HTML page is very easy. For the Windows Media format, the following HTML is used:

```
<OBJECT ID="MediaPlayer" classid="CLSID:22d6f312-b0f6-
   11d0-94ab-0080c74c7e95" width=200 height=160
   CODEBASE="http://activex.microsoft.com/activex/
   controls/mplayer/en/nsmp2inf.cab#Version=5,1,52,701"
   standby="Loading Media Player components..."
   TYPE="application/x-oleobject">
  <PARAM NAME="FileName" VALUE="MEDIA FILE URL">
  <PARAM NAME="AutoStart" VALUE="true">
  <PARAM NAME="ShowControls" VALUE="0">
</OBJECT>
<EMBED TYPE="application/x-mplayer2"
   PLUGINSPAGE="http://www.microsoft.com/Windows/
   MediaPlayer/" SRC="MEDIA FILE URL" NAME="MediaPlayer1"
   SHOWCONTROLS="0" WIDTH="200" HEIGHT="160"
   AUTOSTART="FALSE"></EMBED>
```

Just replace MEDIA FILE URL with the actual URL of the Windows Media file in both the OBJECT and EMBED tags.

For a QuickTime movie, use the following:

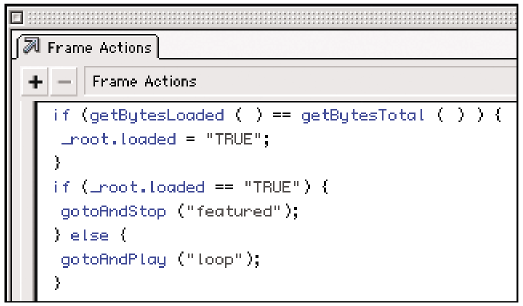
```
<EMBED SRC=”QUICKTIME FILE URL” WIDTH=”200”
    HEIGHT=”160” TYPE=”video/quicktime”>
```

Again, replace QUICKTIME FILE URL with your URL and you're ready to go.

6.4.1 VIDEO LIST

The Footage section of Billabong-USA utilizes Generator to create a dynamic list of the videos available for each sport. We've included the source file for this, but please note that Generator is a web server application for creating dynamic Flash content. Because we have a unique server environment, these files won't "work" upon export. We're examining them to give you insight into how we utilize Generator. We'll be delving more into Dynamic Flash content management in the next chapter.

That being said, look at the file video_list.fla. There aren't many actions on the main timeline. Other than two stops on the last two frames, there is a simple preloader on frame 3 (see Figure 6.17).



```
if (getBytesLoaded ( ) == getBytesTotal ( ) ) {
 _root.loaded = "TRUE";
}
if (_root.loaded == "TRUE") {
 gotoAndStop ("featured");
} else {
 gotoAndPlay ("loop");
}
```

Figure 6.17 The script on frame 3 of the file named video_list.fla.

```
if (getBytesLoaded ( ) == getBytesTotal ( ) ) {
    _root.loaded = "TRUE";
}
if (_root.loaded == "TRUE") {
    gotoAndStop ("featured");
} else {
    gotoAndPlay ("loop");
}
```

The getBytesTotal() function returns the value in bytes of the current timeline. The getBytesLoaded() function returns the value of the bytes loaded. We compare these values and, if they're equal, set the loaded variable to true. Then there is an if statement that continues playing if loaded is true or continues looping if it isn't. The reason for the additional if statement is that it gives the flexibility to add additional requirements before continuing.

An alternative method for this would be to use the movie clip object methods getBytesLoaded() and getBytesTotal() to perform the same function. The getBytesLoaded method returns the number of bytes loaded on a movie clip. The getBytesTotal method returns the size of a movie clip. These methods are used for both internally and externally loaded movie clips.

Go to the frame featured. This is the list of the three featured videos in this sport section. Because we've constrained the featured video section to three videos, there's no need for a scrolling list. We went with a static, hard-coded list (see Figure 6.18).

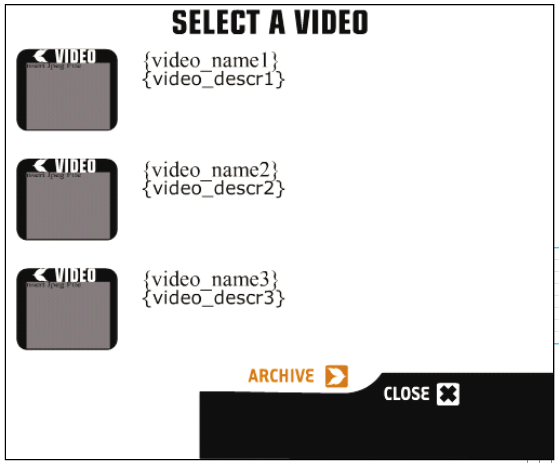
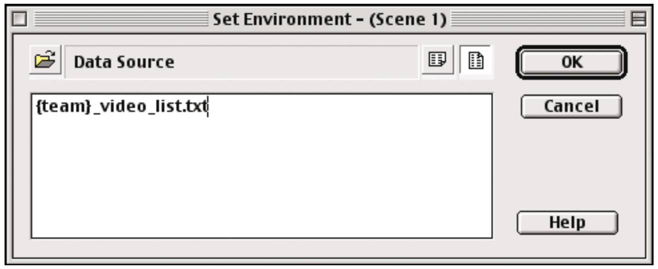


Figure 6.18 The featured videos list on the featured frame.

The data from this list comes from the environment data. The environment data is the first icon of the three icons at the top right of the Flash application window. You'll see that we're using the data source {team}_video_list.txt, as shown in Figure 6.19.



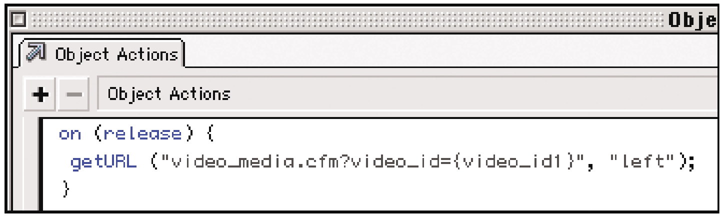
```
{team}_video_list.txt
```

Figure 6.19 The Set Environment dialog box in Flash.

The Generator variable {team} is provided through the URL of the .swt. We'll discuss URL variables and offline Generator in the next chapter. All we need at this point is to know that {team}_video_list.txt will evaluate to surf_video_list.txt (or skate or snow). Here's the contents of snow_video_list.txt:

```
name,value
video_src1,"crawford.jpg"
video_name1,"Crawford"
video_descr1,"Crawford ripping on anything with snow on it"
video_id1,"8"
video_src2,"kevin.jpg"
video_name2,"Kevin"
video_descr2,"Kevin Jones knows how to snowboard"
video_id2,"10"
video_src3,"kevin_2.jpg"
video_name3,"Kevin"
video_descr3,"Kevin at Mammoth 12/01"
video_id3,"16"
```

Look at the stage. If you look at the Generator variables {}, you'll see where the data flows. Select one of the video buttons (see Figure 6.20). Look at the actions:



```
on (release) {
 getURL ("video_media.cfm?video_id={video_id1}", "left");
}
```

Figure 6.20 The actions on one of the video buttons.

```
on (release) {
    getURL ("video_media.cfm?video_id={video_id1}", "left");
}
```

You'll see that we use Generator variables here to construct a URL string to pass to Cold Fusion.

In the library, look at the symbols video_thumb1,2,3. We've got an Insert JPG with a source of ../../images/video/{video_src1}.

Now that we've seen the featured section using an environment data source, go to the archive frame, as shown in Figure 6.21. This is the archived list of footage.

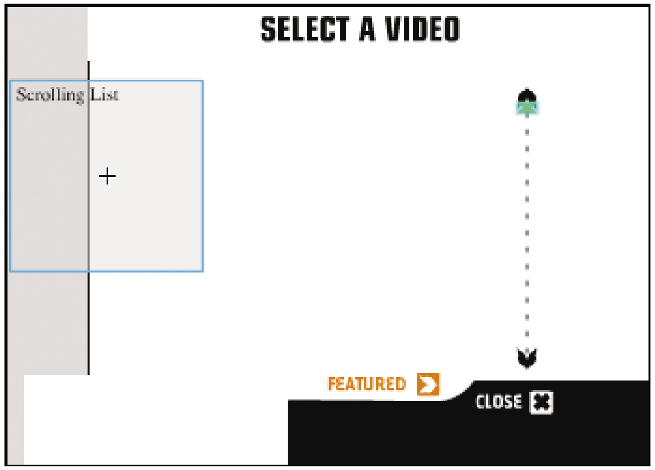
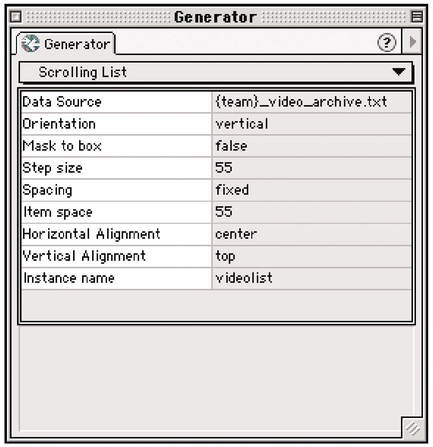


Figure 6.21 The archived list of footage on the archive frame in the main timeline.

The central part of this section is a Generator scrolling list (see Figure 6.22).



Data Source	{team}_video_archive.txt
Orientation	vertical
Mask to box	false
Step size	55
Spacing	fixed
Item space	55
Horizontal Alignment	center
Vertical Alignment	top
Instance name	videolist

Figure 6.22 The Generator template named Scrolling List displays the Generator scrolling list window.

The Data Source is {team}_video_archive.txt, which, like the previous environment data source, will evaluate to snow_video_archive.txt (or skate, surf). Here's the content of snow_video_archive.txt:

```
clip,video_name,video_descr,video_id
archive_item,"Crawford","Crawford ripping on anything with snow
    on it",8
archive_item,"Kevin","Kevin Jones knows how to snowboard",10
archive_item,"Kevin","Kevin at Mammoth 12/01",16
```

A Generator scrolling list uses the clip parameter to decide which movie clip in the library to use for this list. Look at the clip archive_item in the library (see Figure 6.23).

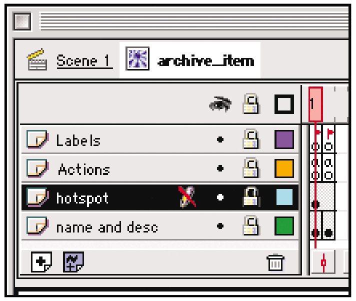


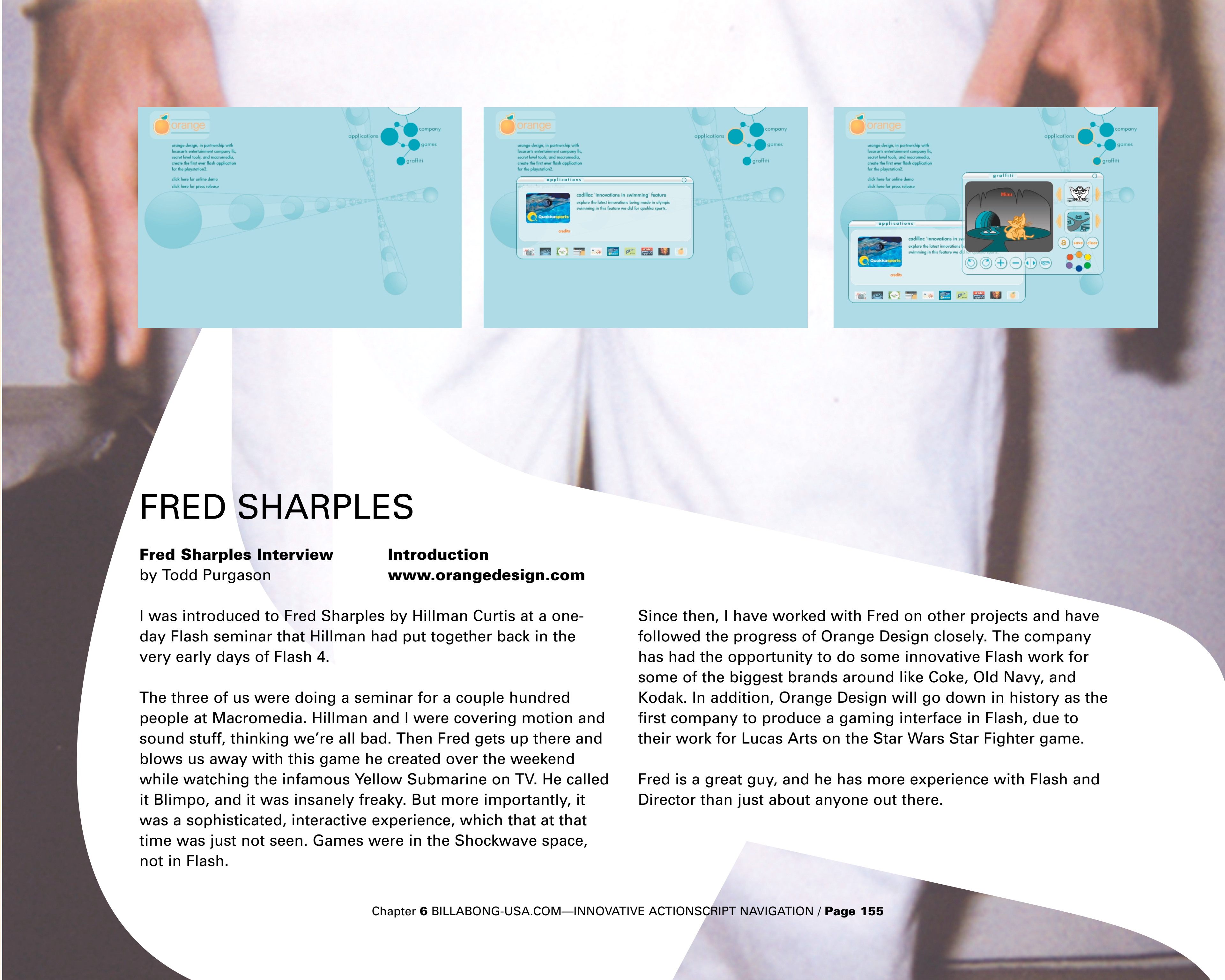
Figure 6.23 The timeline of the clip named archive_item.

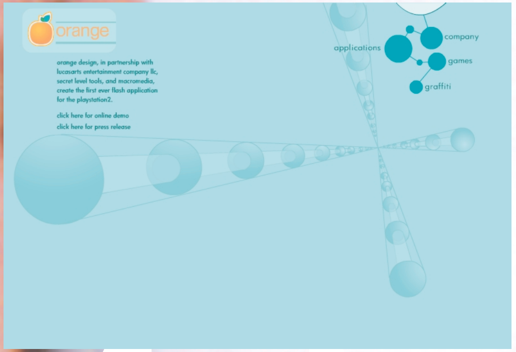
The clip is set up like a rollover with off and on frames. The content of the clip is inserted through Generator variables.

6.5 SUMMARY

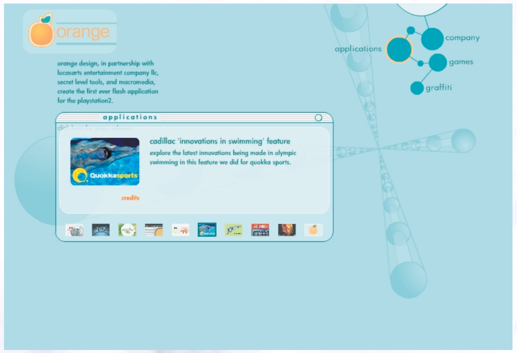
We are really excited about all of the dynamic, interesting interaction and content we included in the Billabong site. You can see from this example how interactivity can be used not only to create an interesting experience, but also to reinforce creative concepts in your designs. This is the stuff that really excites us. We feel that it makes a site successful and makes us successful as designers and developers.

To clients, however, what really makes a site successful is that they can flawlessly and easily update the content of the site, keeping it current with their business. Even in this hard-core market segment of extreme sports apparel, that is a main concern for these companies. So, in the next chapter, we are going to show you how we married the immersive Flash experience on Billabong-USA.com with a custom-built web application that gives clients the control they want and need for their content so the site can be successful.





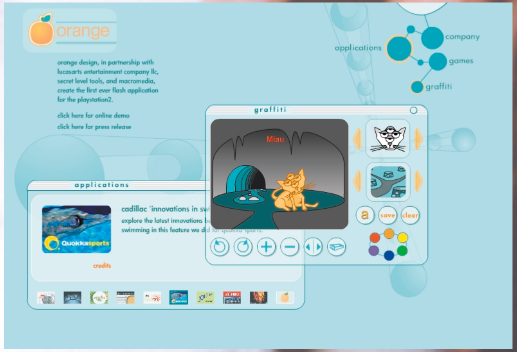
orange design, in partnership with lucasarts entertainment company llc, secret level tools, and macromedia, create the first ever flash application for the playstation2.



orange design, in partnership with lucasarts entertainment company llc, secret level tools, and macromedia, create the first ever flash application for the playstation2.

cadillac 'innovations in swimming' feature

explore the latest innovations being made in olympic swimming in this feature we did for quokka sports.



FRED SHARPLES

Fred Sharples Interview
by Todd Purgason

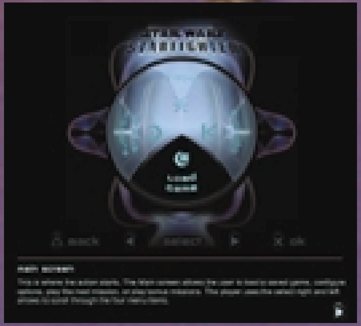
Introduction
www.orangedesign.com

I was introduced to Fred Sharples by Hillman Curtis at a one-day Flash seminar that Hillman had put together back in the very early days of Flash 4.

The three of us were doing a seminar for a couple hundred people at Macromedia. Hillman and I were covering motion and sound stuff, thinking we're all bad. Then Fred gets up there and blows us away with this game he created over the weekend while watching the infamous Yellow Submarine on TV. He called it Blimpo, and it was insanely freaky. But more importantly, it was a sophisticated, interactive experience, which that at that time was just not seen. Games were in the Shockwave space, not in Flash.

Since then, I have worked with Fred on other projects and have followed the progress of Orange Design closely. The company has had the opportunity to do some innovative Flash work for some of the biggest brands around like Coke, Old Navy, and Kodak. In addition, Orange Design will go down in history as the first company to produce a gaming interface in Flash, due to their work for Lucas Arts on the Star Wars Star Fighter game.

Fred is a great guy, and he has more experience with Flash and Director than just about anyone out there.



Teaser from the Interview

TP: So, we're here with Fred Sharples from Orange Design. So, who is Fred Sharples?

FS: Fred Sharples, the enigma.

That's a hard one, Todd.

TP: He's the guy with the bird on his shoulder.

FS: Yeah, I'm the guy with the bird on his shoulder in Hillman's book.

TP: Ok, who is Orange Design? We'll put it that way.

FS: Yeah, that's probably better. Orange Design is a small Flash studio, actually a multimedia studio, I'd say. But we do all Flash stuff, and we just started it so we could have a small company that would be fun to work at, that would make a little money, and you know, stay focused on doing interesting projects. Kind of an escape from working in really big agencies and big companies. That's sort of an antidote to that. And it was an experiment by Pam and myself that turned out really well. We were able to believe that we could have our own sort of creative group that could be independent, that would be able to say no when we needed to, and that would be able to reap the benefits when it was good to say yes. And we were able to, you know, really control what projects we'd take, how we would do them, how we would treat people that worked with us, and how we would be treated by our clients, it worked out really well. So that was the main focus. And then, additionally, it was just to make sure that we had really, really good people that worked for, with us and to ensure that we can deliver in a stress-free environment, I'd say.

TP: Wait, wait, stress free? What's that?

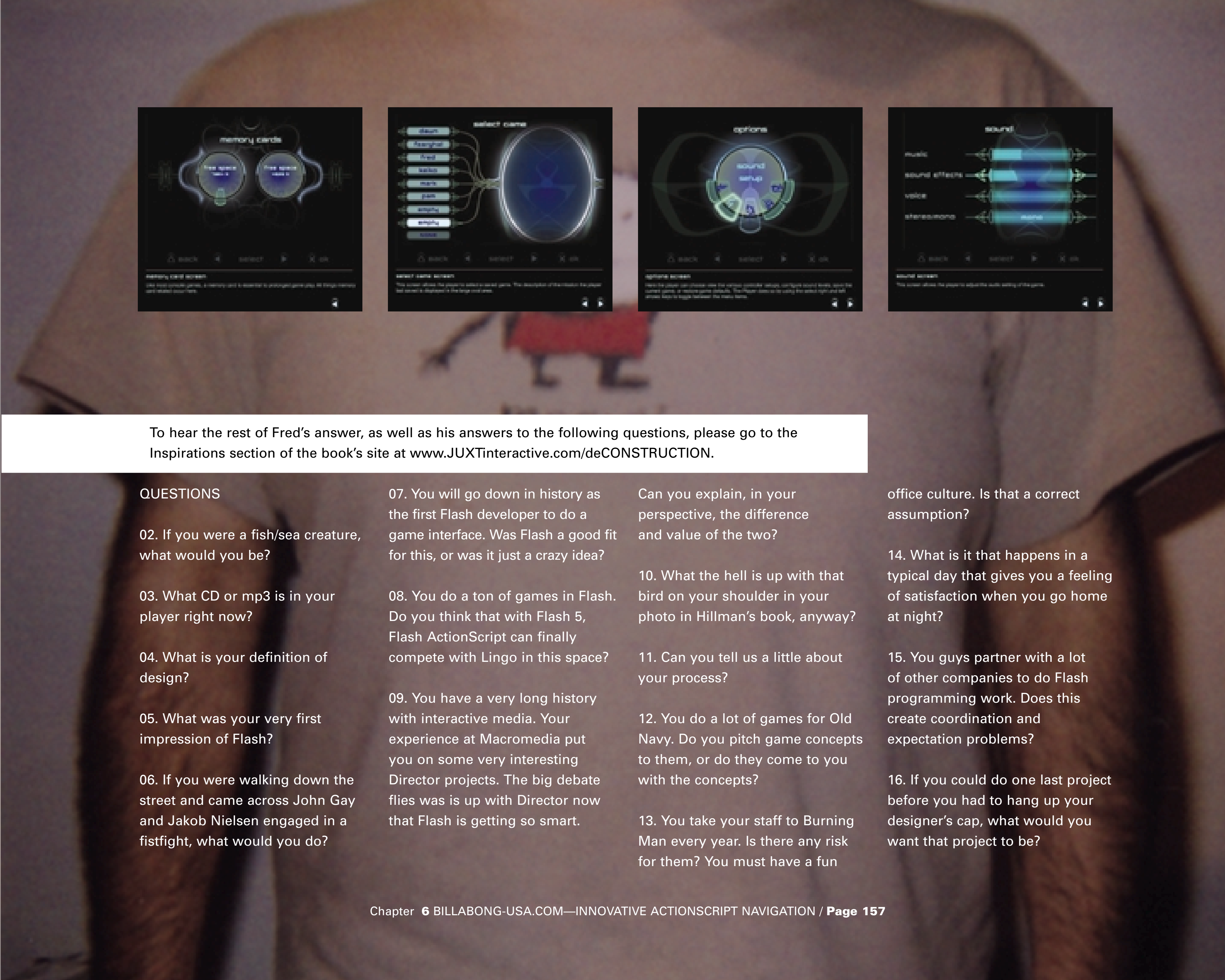
FS: Yeah, well…

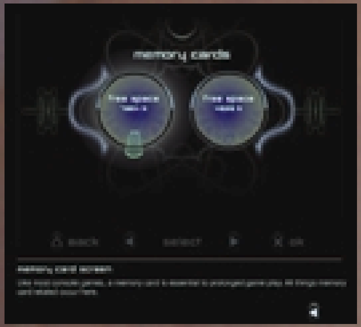
TP: Relatively.

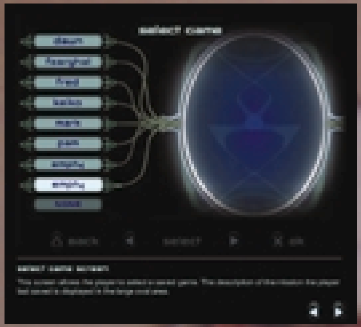
FS: Yeah, compared to most places, yeah definitely. So yeah, we all work really hard, obviously, but it's more of a social experiment.

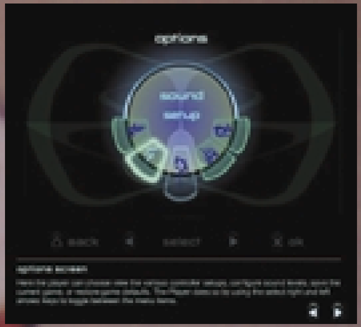
TP: Cool.

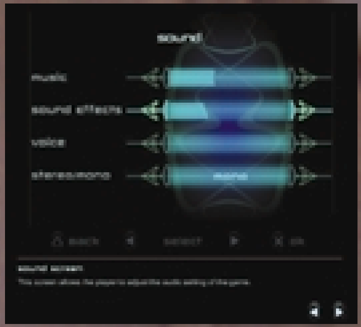
FS: That's all. It's turned out really well.











To hear the rest of Fred's answer, as well as his answers to the following questions, please go to the Inspirations section of the book's site at www.JUXTinteractive.com/deCONSTRUCTION.

QUESTIONS

02. If you were a fish/sea creature, what would you be?

03. What CD or mp3 is in your player right now?

04. What is your definition of design?

05. What was your very first impression of Flash?

06. If you were walking down the street and came across John Gay and Jakob Nielsen engaged in a fistfight, what would you do?

07. You will go down in history as the first Flash developer to do a game interface. Was Flash a good fit for this, or was it just a crazy idea?

08. You do a ton of games in Flash. Do you think that with Flash 5, Flash ActionScript can finally compete with Lingo in this space?

09. You have a very long history with interactive media. Your experience at Macromedia put you on some very interesting Director projects. The big debate flies was is up with Director now that Flash is getting so smart. Can you explain, in your perspective, the difference and value of the two?

10. What the hell is up with that bird on your shoulder in your photo in Hillman's book, anyway?

11. Can you tell us a little about your process?

12. You do a lot of games for Old Navy. Do you pitch game concepts to them, or do they come to you with the concepts?

13. You take your staff to Burning Man every year. Is there any risk for them? You must have a fun office culture. Is that a correct assumption?

14. What is it that happens in a typical day that gives you a feeling of satisfaction when you go home at night?

15. You guys partner with a lot of other companies to do Flash programming work. Does this create coordination and expectation problems?

16. If you could do one last project before you had to hang up your designer's cap, what would you want that project to be?