CHAPTER 06

Less Cluttered and More Usable

Thinking is more interesting than knowing, but less interesting than looking.

—Goethe

When creating a Flash interface, the designer must consider carefully all the options the potential user will need. Almost always, you must provide two sets of features:

- Those critical to user success in accomplishing their goals on the site
- Those that are not essential for the site goals to be met, but that add to the overall user experience

Often the most critical features are also global features, such as a help system. They should be accessible from anywhere and have prime real estate on the screen. The functions that enhance the user experience rather than enable it are generally not global and relate to a certain focused area of the site. To make the user experience as intuitive as possible, access to these features should be local to the area to which they relate. For example, information about a thumbnail image in the MODA site should be accessed via the thumbnail rather than through a global menu control.

To successfully design an interface that may have global and section-specific features, you should begin by organizing these different groups of features. Actions central to the user's main goals should have prime locations on the screen, whereas less critical actions can afford to be less in the limelight.

Nancy's main goals on the MODA site are as follows:

- To learn about the artist
- To view the paintings

The functionality to accomplish these goals should be given the most importance visually on the screen. Nancy's secondary goals on the site include the following:

- To find out more about each piece and how it was made
- To save a given piece to her desktop to use as wallpaper
- To print out the information she might come across on the site

To help Nancy achieve these secondary aims, you need to make sure these functions are still accessible but less dominant in the interface.

SOLUTIONS

Most of Nancy's interaction on the site will be with the scrolling list at the bottom of the screen, because that gives her access to the primary features. Some other features of the site may interest her as well but may not be part of the client's main focus or part of Nancy's goals on the site. Examples might be accessing additional information about the paintings, seeing the original sketches, printing out copies, and downloading desktop wallpaper. Because of Nancy's comfort level with computers and her obvious affinity for aesthetic value, it becomes clear that the best way to incorporate these secondary features is by storing them in the mouse under menu.

The trick when designing a user interface that incorporates a number of elements of differing importance is to keep the design simple. The main controls in the interface should be clearly distinguished from the ancillary features of the site. This is particularly important on the MODA site because its clean appearance is crucial to providing a gallery-like feel to the site. As in a gallery, here visitor attention should be drawn to the art they are seeing, not the gallery itself.

In exploring the possibilities available for providing the less critical functionality to Nancy without cluttering the interface and drawing attention away from the main controls, one mechanism offers a clear solution: the mouse under menu. This is a menu of items accessed by just clicking the target, much like the contextual (or right-click) menus in Windows, MacOS, and other operating systems. The immediate advantage that this control offers is that it is never in the way, yet always accessible. It is context-sensitive, so it will be accessible only when the item that Nancy wants to act upon is clicked.

For this control to work for Nancy, she needs to know that it is available. A straightforward way of doing this is to have a tool tip, caption, or custom mouse cursor that shows Nancy that the item under her mouse cursor is clickable.

For the MODA site, clicking a painting, for example, will bring up options to print the image, download a copy to her hard drive, or see alternative versions of the chosen image. Because Nancy must click the image to trigger the mouse under menu, she should intuitively understand the menu relates specifically to that image.

The mouse under menu adheres to Fitts's law pretty closely, too. By creating a menu system that appears at the mouse pointer, you are offering a menu that requires almost no mouse movement to access, greatly reducing the amount of time it takes the user to operate the menu.

To provide even greater usability, the mouse under menu will differ from a typical contextual menu in one key respect: It will be movable from its initial position. Contextual menus stay in the same position in which they were opened. This added control will enable Nancy to move the menu should it start to cover up a crucial part of the image or information.

Fitts's law states that one of the five most accessible areas of the screen is the point where the mouse is at any given time. The other four are the top, bottom, and right and left sides of the screen. If you want more detail on Fitts's law, check out Appendix B, "Usability Resources," for books and web sites devoted to specific usability guidelines.

Note

Seeing It in Action

Open **06_mouseMenu.fla** found in the **Projects** folder on the *Skip Intro* CD-ROM and export or test the movie. As you roll over the main picture and click, the mouse menu appears (see Figure 6.1). From here you can drag the menu wherever you want on the screen. Rolling off the menu will fade it out, so the site keeps its clean, uncluttered look.

Notice when you click that the menu appears wherever the mouse is. This is the most important characteristic of the mouse under menu. Close the test movie and open the Library panel for the sample movie. Edit the graphic symbol named Paintings and select the button on the top layer Menu button. If you look at the Actions panel, you will see how simple it is to activate the mouse under menu:

Figure 6.1
Clicking the big paintings should activate the mouse under menu.

```
on(press)
{
    mc_mouseMenu.showMenu();
}
```

The `showMenu()` function is all you have to call anywhere in the movie to show the menu. That's pretty simple.

Now click the Skip Intro Components folder in the library and open the MouseMenu folder that is inside the Skins folder. The three movie clips contained in that folder define what the mouse under menu looks like. These clips contain no code, just visual elements that can easily be changed to fit the mouse under menu to any project in which you may need it. Separating the visual parts from the code keeps other developers that may be using your components from having to dive into your code and potentially break it.

In the following section, you can take a look at how this menu was implemented to understand the power of this component.

IMPLEMENTATION

For the mouse under menu, you need to implement only a few features:

- **Mouse under.** The menu should appear directly under the mouse when Nancy clicks her mouse button.
- **Draggability.** Nancy should be able to drag the menu around the screen to avoid covering up parts of the image or text that she is viewing. The developer should, however, be able to turn off this feature.
- **Instant off.** Depending on the use of the menu, when the user clicks a selection the menu can either disappear, like contextual menus, or remain on. For Nancy and her use of the site, keeping the menu on is the better option.

Now that you have an idea of the tasks that the mouse under menu should accomplish, it's time to build it.

CONSTRUCTION

Good news! In terms of coding, the mouse under menu is not very complex, but takes advantage of some great Flash MX features that help to make this clip truly flexible.

The first step is to build the graphics for the mouse under menu. As always, even though your focus is on coding, the way you approach the visuals is just as important.

Setting Up the Movie

Make a new Flash movie and set its frame rate to 20. Create a new layer in your movie labeled **mouse menu**. Put the new layer at the very top of every layer with content in it. This will ensure the mouse under menu is on top of any other content in your movie.

On this layer, draw a rectangular shape on the Stage approximately 110 pixels wide by 30 pixels tall, and give it a black, hairline stroke and a light-gray fill. This will be the beginning of the skin movie clips for your mouse under menu.

Select the middle portion of the rectangle, 6 pixels from the top of the rectangle and about 18 pixels tall, and make it into a movie clip (F8) named **menu_middle** with its registration point to the upper-right corner.

Figure 6.2
*Select the middle portion of the rectangle and make it into a movie clip called **menu_middle**.*

Now give the top portion of the rectangle a small tab area like the one illustrated in Figure 6.2. Select this whole top portion and make it into a movie clip called **menu_top**, also setting its registration point to the upper right.

Finally, select the bottom portion of the rectangle and make it into a movie clip called **menu_bottom** and set its registration point to the upper right as well. You should now have three movie clips on Stage. Double-click the middle movie clip to place it in Edit mode.

Create a new layer called **item name** and create a dynamic text field on that layer. Set the *x* coordinate of the field to 3 and the *y* coordinate to 0. Double-click to edit the text of the field and set the text to read **item Name**. Then set the variable for the field to **itemName** and make sure the field is not selectable. Click the Character button in the Properties panel for the field and also ensure that you've embedded all the characters of the font. Make this clip 10 frames long. Make frame 5 a keyframe on both layers in this clip and give the frame a frame label of **over**. On frame 6, select the light-gray section of the menu item image and change its color to a light blue. This clip will be used for every menu item in your mouse under menu.

To make these accessible from within the ActionScript you'll write later, you will have to change the linkage properties for each of the three movie clips. Start by right-clicking (in Windows) or Control-clicking (on the Macintosh) on the **menu_top** movie clip in the Library panel and selecting Linkage from the context menu. Click the Export for ActionScript check box on the Linkage Properties dialog box. Flash will automatically fill in the Identifier field with **menu_top**. Click the OK button and repeat the process for the **menu_middle** and **menu_bottom** movie clips.

Now that you've created the skins that control how your menu *looks*, now it's time to dive into the guts of this component.

Creating the Component

Make a rectangle on the **mouse menu** layer. Select the rectangle and make it a movie clip named **MouseMenu**, and then give this clip and instance name of **mc_mouseMenu**. Double-click the new movie clip symbol to open it in Edit mode.

CODE The code for the mouse under menu will work like several of the other components in the book—it uses a frame loop. To begin, add two more frames to the movie clip and then add a new layer named **actions**. Select the three frames in the **actions** layer and convert them to keyframes (Modify>Frames>Convert to Keyframes).

Initialization

Click frame 1 of the **actions** layer, open the ActionScript editor (F2), and enter the following code:

```
var MENU_ITEMS = ["item 1", "item 2", "item 3"];
var HIDE_DELAY = 1000;
var HIDE_ON_CLICK = true;
```

These first few variables will be replaced later with component parameters. The ***MENU_ITEMS*** variable is an array of elements that represent your menu items. In this case, you have three menu items: item 1, item 2, and item 3.

The ***HIDE_DELAY*** variable holds the number of milliseconds in which the menu waits without user input before fading away.

HIDE_ON_CLICK will tell the component whether to fade away immediately after the user clicks an item in the menu.

Next you need to initialize some global variables. Add this code:

```
var gMenuShowing = false;
var gHideCalled = false;
var gTimerStarted = false;
var gStartTime = 0;

this._alpha = 50;
this.swapDepths(99999);
```

The ***gMenuShowing*** variable will be used later to tell different functions in the component whether the menu is currently being displayed.

The ***gHideCalled*** global variable is set to `true` when the `hideMenu()` function (which you will define later) gets called and is used in fading out the menu.

gTimerStarted holds the state of the timer which will hide the menu after a given number of milliseconds (defined earlier in ***HIDE_DELAY***) has elapsed.

The alpha of the component is set here to 50%, but this will be changed to 0 when the component is complete. This ensures the menu is hidden from view right when the mouse menu component is loaded.

The built-in Flash `swapDepths()` function ensures the menu displays over all other items on its layer.

Building the Menu

Now it's time to start putting those skin movie clips you created earlier to good use. Start by entering this code in the Actions panel:

```
this.attachMovie("menu_top", "mc_menu_top", 1);
```

This code attaches the **menu_top** movie clip to the current movie clip. This will be the top of your menu.

Now is a good time to save your movie and test it. You should notice that the **menu_top** movie clip appears 50% transparent in the same place as the **MouseMenu** clip you created. If this is the case, you're off to a good start.

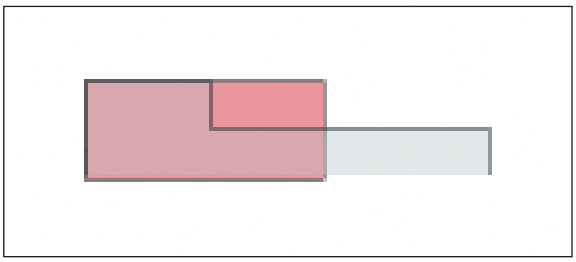

Figure 6.3
*The **menu_top** movie clip appears at the position of the **MouseMenu** movie clip you are currently defining.*

Close the test movie and return to the Actions panel for frame 1 of the **MouseMenu** movie clip. Add this code next:

```
var tempLevel = 2;
var tempOffset = this.mc_menu_top._height;
for (var i = 0; i < MENU_ITEMS.length; i++) {
    this.attachMovie("menu_middle", "mc_menu_item_" + i, tempLevel);
    this["mc_menu_item_" + i].itemName = MENU_ITEMS[i];
    this["mc_menu_item_" + i]._y = tempOffset;
    this["mc_menu_item_" + i].gotoAndStop(1);

    tempLevel++;
    tempOffset += this["mc_menu_item_" + i]._height;
}
```

This little loop is the heart of the under mouse menu component. It loops through the ***MENU_ITEMS*** array, attaches a new **menu_middle** movie clip for each element in the array, and then sets the itemLabel field inside that **menu_middle** movie clip to the text of that element of the array. This will enable you to add as many menu items as you need by just adding those items to the ***MENU_ITEMS*** array.

Now, before testing the loop, add the bottom part of the skin next by entering this code:

```
this.attachMovie("menu_bottom", "mc_menu_bottom", tempLevel);
this.mc_menu_bottom._y = tempOffset;
```

Now you're ready to test the movie again. You should notice that the whole menu appears as a whole piece. Great work so far!

As you can see, much of the real work is now done. The next several steps will deal with adding the finer details that help to make Nancy's experience the best it can be.

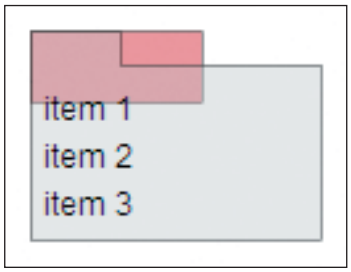

Figure 6.4
You should see something very similar to this now, with all the items in the **MENU_ITEMS** *array showing up as items in your menu.*

Mouse Handling

The next code you will add will handle the mouse events for the dragging of the menu. Add this code next to the Actions for frame 1:

```
function menuMouseDownFunc()
{
    if(this.hitTest(_root._xmouse, _root._ymouse, true)) {
            this.deltaX = _root._xmouse - this._parent._x;
            this.deltaY = _root._ymouse - this._parent._y;
            this.isDragging = true;
    }
}

function menuMouseUpFunc()
{
    this.isDragging = false;
}
```

The `menuMouseDownFunc()` function tests to determine whether the user's mouse is actually over the top of the menu. If it is, the difference between the clip's *x* and *y* coordinates and the mouse's are calculated and saved to the delta variables and the ***isDragging*** Boolean is set to `true`.

The `menuMouseUpFunc()` function sets the ***isDragging*** variable back to `false`. This will be used to tell the next function you will enter to stop moving the menu along with the mouse. Enter this code now:

```
function menuMouseMoveFunc()
{
    if(this.isDragging) {
            this._parent._x = _root._xmouse - this.deltaX;
            this._parent._y = _root._ymouse - this.deltaY;
            updateAfterEvent(mouseMove);
    }
}
```

`menuMouseMoveFunc()` checks to determine whether the ***isDragging*** variable is `true` and, if it is, updates the position of the menu and refreshes the drawing of the frame immediately to ensure the dragging is smooth.

How do these functions ever get called? These three functions will get attached to the movie clip at the top of the menu (**menu_top**), enabling the user to drag it around. Do that now. Right below the code where you attach the **menu_top** movie clip (before looping through the ***MENU_ITEMS*** array), add this ActionScript:

```
this.mc_menu_top.onMouseDown = menuMouseDownFunc;
this.mc_menu_top.onMouseUp = menuMouseUpFunc;
this.mc_menu_top.onMouseMove = menuMouseMoveFunc;
```

What this code does is attach each of the last three functions you defined to the three mouse events for the **menu_top** movie clip. It's that easy! Now before testing the movie again, go to keyframe 3 of the ***MouseMenu*** component and add this ActionScript:

```
41 this.mc_menu_top.onMouseDown = menuMouseDownFunc;
42 this.mc_menu_top.onMouseUp = menuMouseUpFunc;
43 this.mc_menu_top.onMouseMove = menuMouseMoveFunc;
```

Figure 6.5
*Attach the mouse functions to the **menu_top** movie clip to allow it to be dragged around with the mouse.*

```
gotoAndPlay(2);
```

This will ensure that the component loops over the last two frames and does not replay the frame of the clip, which would reinitialize the variables and cause the dragging to malfunction. Now save and test your movie.

You should be able to grab the top portion of the menu and easily drag it around the screen. Wow, that's smooth!

Now you will add two mouse functions to the menu items themselves. Start by going to frame 1 of the **MouseMenu** clip and adding this to its Actions:

```
function itemMouseUpFunc()
{
    if (this.hitTest( _root._xmouse, _root._ymouse, false)) {
            _root.underMenuDispatch(this.itemName);
            if (this._parent.HIDE_ON_CLICK) hideMenu();
    }
}

function itemMouseMoveFunc()
{
    if (this.hitTest( _root._xmouse, _root._ymouse, false)) {
            this.gotoAndStop("over");
    } else {
            this.gotoAndStop(1);
    }
}
```

The `itemMouseUpFunc()` function first checks to see that the mouse is released over the calling clip by using the Flash built-in function, `hitTest()`. Then it calls a dispatch function called `underMenuDispatch()` located in the root movie. A dispatch function takes in a parameter (in this case, `this.itemName`) and decides what to do with it, dispatching any other functions as a result. You will define this particular dispatch function near the end of this chapter. The `itemMouseMoveFunc()` function also checks to see whether the mouse is over the clip and, if it is, it tells the clip to go to the frame labeled **over**; otherwise the clip is told to display frame 1.

Now that you've defined these functions, it is time to attach them to the individual menu items. Inside the loop that creates each menu item, find the code that tells the newly created item to go to and stop on frame 1 (`this["mc_menu_item_" + i].gotoAndStop(1)`). Add this code immediately following:

```
this["mc_menu_item_" + i].onMouseUp = itemMouseUpFunc;
this["mc_menu_item_" + i].onMouseMove = itemMouseMoveFunc;
```

Now save your movie and test what you have so far. You should now be able to roll over each of the items in your mouse under menu and see the item highlighted. If it is not working properly, that's alright; just review the code and compare it to what's in this chapter to help you track down the problem.

You may have noticed one peculiar thing about your menu: When you drag it, the original rectangle you made at the very beginning of this chapter is still showing behind the menu and drags along with it. To fix this problem, select frame 2 of the layer that the rectangle is on and create a keyframe (F6). Now delete any items from this layer on frames 2 and 3. This will ensure that the rectangle shows up only in the first frame of the clip by removing any remnant of the rectangle from frames 2 and 3.

Open to the Public

You are now ready to add the public functions to this component. At the top of frame 1, right after you define the ***HIDE_ON_CLICK*** variable, enter this code in the Actions panel:

The notion of public and private functions and variables is a powerful notion in the world of the C programming language; because when you declare something as private, it *is* actually private, and trying to use it will shower you with compiler warnings. In ActionScript and JavaScript, however, the notion of public and private functions and variables is purely an organizational tool and in no way changes the way Flash or your web browser interprets or manages the code.

Note

```
function showMenu()
{
    if (gMenuShowing) {
            if (this.hitTest(_root._xmouse, _root._ymouse, true)) {
                    return;
            } else {
                    hideMenu();
                    return;
            }
    }
    gHideCalled = false;
    gTimerStarted = false;
    gMenuShowing = true;
    this._x = _root._xmouse;
    this._y = _root._ymouse;
    this._alpha = 100;
}
```

The `showMenu()` function typically gets called as a result of a mouse click and begins by checking to see whether the menu is already showing and whether the mouse is over the menu. If the menu is showing and the mouse is not over it, the menu is hidden by calling the `hideMenu()` function (which you will define in a moment). If the menu is not already showing (***gMenuShowing*** is `false`), the ***gHideCalled*** and ***gTimerStarted*** variables are set to `false`. These variables tell the clip that the `hideMenu()` function has been called, and that the delay timer (which hides the menu after a certain amount of inactivity) is set to `false`, respectively. The ***gMenuShowing*** variable is then set to `true` to tell the rest of the clip that the menu is showing. Finally, the menu is positioned at the current mouse pointer coordinates and the alpha of the clip is set to 100% opacity.

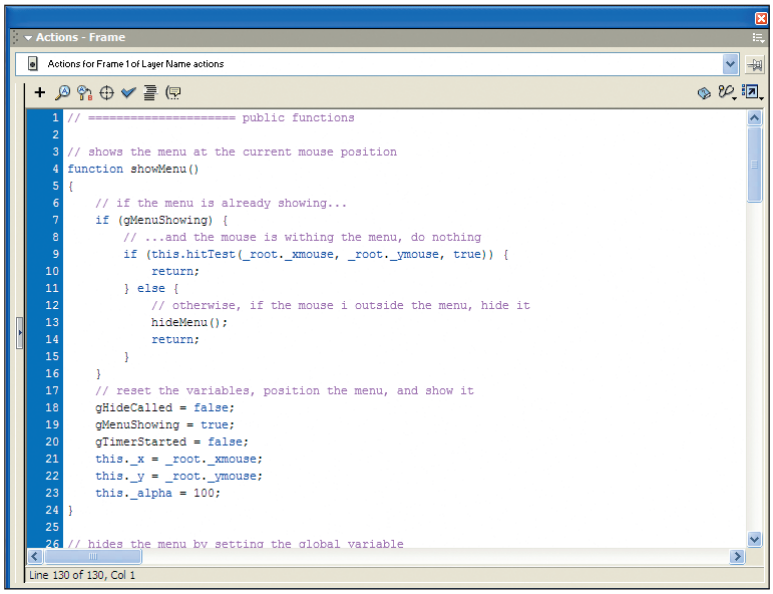

Figure 6.6
Define the clip's public functions at the top of the Actions panel. This makes it easy for other developers to spot if they open the clip and view the code.

Now it is time to define the `hideMenu()` function. Enter this code:

```
function hideMenu()
{
    gMenuShowing = false;
    gHideCalled = true;
}
```

This function doesn't seem like it does very much, but it is important. After telling the rest of the movie clip that the menu is no longer showing, it sets the ***gHideCalled*** variable to `true`. This variable will get checked in the code you will be entering next and used to fade the menu out.

To test that the `showMenu()` function works properly, you need to hide the menu completely when it initializes. Therefore, find the line of code where you set the alpha of the clip to 50% and set the alpha to 0. Below that line of code, add this:

```
this._x = -1000;
```

This will ensure not only that the clip is invisible on the Stage (at 0 percent alpha), but also that it can no longer receive any mouse event because it will be positioned way off Stage.

As mentioned earlier, the `showMenu()` function typically gets called from a button symbol, so go to scene 1 of your movie and create a square. Select the square and make it into a button symbol (F8). Add this code to the button:

```
on(press)
{
    mc_mouseMenu.showMenu();
}
```

This very simple code calls the `showMenu()` function you created only moments ago. Now save your work and test the movie again. You should notice that the movie now begins without the menu clip showing and that clicking the button you created shows the menu at the mouse coordinates. Great! Now add the fading-out code.

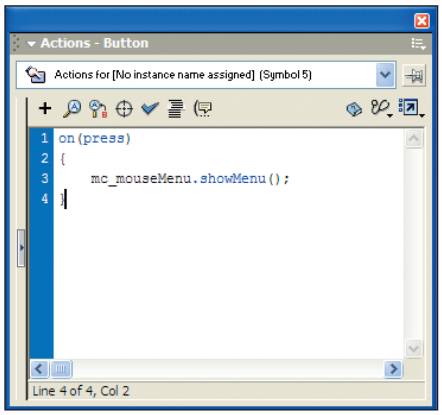

Figure 6.7
The button symbol that will activate the mouse under menu.

Fading

Close the test movie and edit the **MouseMenu** movie clip. Select frame 2 of the Actions layer in the timeline and enter this code:

```
if (gHideCalled) {
    if (this._alpha <= 10) {
            this._x = -1000;
            gHideCalled = false;
    } else {
            this._alpha -= 20;
    }
}
```

Because this clip loops over frames 2 and 3, this code gets checked throughout the lifetime of the movie. If ***gHideCalled*** is `true` (which would be the case if the `hideMenu()` function were called), the clip begins to grow more and more transparent until it is positioned off Stage.

Test your movie and click the button you created to activate the menu. Then click outside the menu area but still in the button. You should find that the menu gently fades away when you click outside the area of the menu.

Now it is time to add the timer code, which will fade the menu out if the user's mouse pointer is outside the menu for a given amount of time. Start by adding this code right after the code you added previously:

```
if (gMenuShowing && !gHideCalled && !this.hitTest(_root._xmouse,
➥_root._ymouse, true)) {
    if (!gTimerStarted) {
            gStartTime = getTimer();
            gTimerStarted = true;
    } else {
            if (getTimer() > (gStartTime + HIDE_DELAY)) {
                    hideMenu();
            }
    }
} else {
    gTimerStarted = false;
}
```

This code might look a bit daunting, but it is really pretty simple. The first `if` statement checks to see whether the menu is currently showing, that the `hideMenu()` function has not been called, and that the mouse pointer is not within the menu area. If all of those conditions are met, and if the timer has not yet been started, it is started; otherwise, the code checks to see whether the number of milliseconds defined in the ***HIDE_DELAY*** variable have elapsed and then calls `hideMenu()` automatically. As you recall, the ***HIDE_DELAY*** variable was initially set to 1000 milliseconds, or 1 second. Therefore after 1 second of the mouse pointer being outside the area of the menu, the menu should automatically fade. Save and test your movie to determine whether this works.

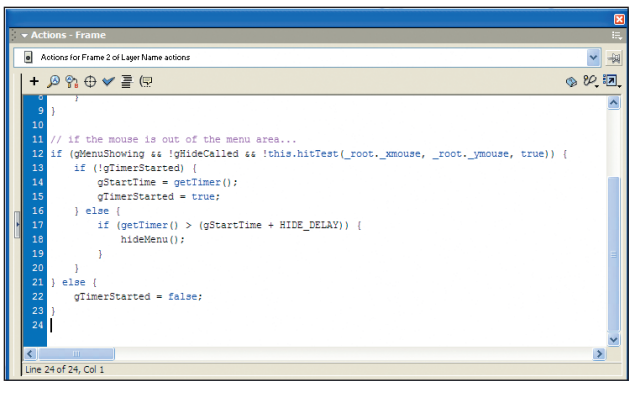

Figure 6.8
This timer code handles fading out the menu after a period of inactivity.

All that is left is to define the component parameters to make this a real, reusable component, and to add the dispatch function mentioned earlier to the root movie timeline.

From Movie Clip to Component

To begin, you will need to comment out the three variables you defined at the top of the Actions panel for frame 1 in your **MouseMenu** clip. Select frame 1 of the Actions layer in your **MouseMenu** clip, open the Actions panel for it (F2), and comment out the ***MENU_ITEMS***, ***HIDE_DELAY***, and ***HIDE_ON_CLICK*** variables.

```
1  /*
2  var MENU_ITEMS = ["item 1", "item 2", "item 3"];
3  var HIDE_DELAY = 1000;
4  var HIDE_ON_CLICK = true;
5  */
```

Figure 6.9
Don't forget to comment out the temporary variables you defined in the beginning.

Right-click (Windows) or Control-click (Mac) the **MouseMenu** clip in the Library panel and select Component Definition.

Start off by adding a ***MENU_ITEMS*** variable and making it an array. Then add a ***HIDE_DELAY*** variable of type Number with a value of 1000, and finally add a ***HIDE_ON_CLICK*** Boolean variable.

Click the OK button and return to your main movie timeline. Select frame 1 of the main timeline and open the Actions panel. Add the following code:

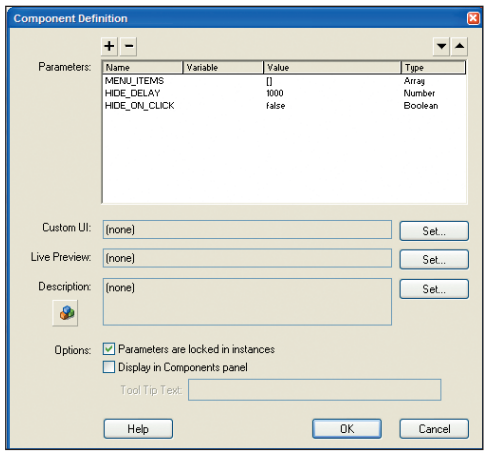

Figure 6.10
This component will have only three simple parameters.

```
function underMenuDispatch(clickedItem)
{
    switch (clickedItem) {
            case "item 1":
                    trace("1 was clicked");
                    break;
            case "item 2":
                    trace("now 2 was");
                    break;
            default:
                    trace(clickedItem);
    }
}
```

This function gets called every time a menu item is clicked. It gets sent the label of the menu item and checks it against a list of cases; if a match is found, the code within that case branch is executed. Using this method, you can have the menu execute any functions you desire. In this particular case, there are cases for handling menu items named item 1, item 2, and a default case that will handle any other menu items.

> The code in the sample movies included on the *Skip Intro* CD-ROM should be identical to the code discussed in the chapter except for the fact that the code on the CD-ROM is heavily commented to make it easier to go through. In some cases, however, items such as the ones listed in the menu and dispatch functions may differ from those in the sample files to make it less specific to the sample movie and more general.
>
> **Note**

```
function underMenuDispatch(clickedItem)
{
    switch (clickedItem) {
        case "item 1":
            trace("1 was clicked");
            break;
        case "item 2":
            trace("now 2 was");
            break;
        default:
            trace(clickedItem);
    }
}
```

Figure 6.11
The dispatch function handles the different menu item labels sent in and dispatches the appropriate functions for them.

To test this code, you need to adjust the parameters of your component. Select the ***MouseMenu*** component on the Stage and click the Parameters tab. Enter five items: **item 1**, **item 2**, **item 3**, **item 4**, and **item 5**. Notice that after adding the first one, every time you click on the plus sign (+) to add a new array element, Flash automatically adds a new item and increments the number appropriately. Now set the ***HIDE_ON_CLICK*** parameter to `true` and test your movie.

Upon clicking any of the menu items in the mouse under menu, you should notice that the menu disappears and that the Output window displays the appropriate message for each item. That's it! You now have a complete, functional, and reusable mouse under menu component.

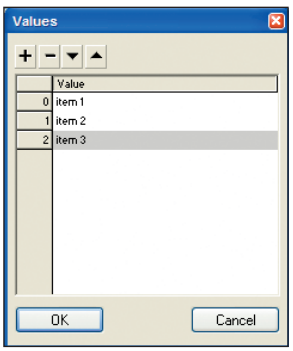

Figure 6.12
Flash automatically adds the next item in the array with an incremented number.

You need to perform one final trick for this component to be truly reusable: You need to ensure the **menu_top**, **menu_middle**, and **menu_bottom** clips tag along appropriately when you drag and drop this clip into other movies. For this to function properly, select the ***MouseMenu*** component in the library, Right-click (Windows) or Control-click (Mac) and select Edit. Create a new layer named **skins** and move it to the bottom of the list of layers. Now drag each of the menu skin clips (**menu_top**, **menu_middle**, and **menu_bottom**) onto the newly created layer and resize each of them so that they are about 10 pixels wide by 10 pixels tall. This effectively hides them from view. Now right-click (Windows) or Control-click (Mac) on the **skins** layer in the layer list and make it a guide layer. This further hides it completely on the Stage. Now you can drag the ***MouseMenu*** component to any other movie and the menu skin clips will automatically follow.

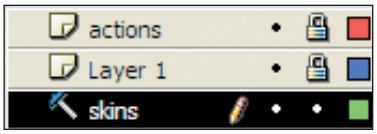

Figure 6.13
Making the layer a guide layer ensures that Flash will not display it.

CONCLUSION

You now have a nice little module that you can use, with minor graphic modifications, in any of your projects!

Key Points/Don't Forget

- **Mouse under menus are an excellent implementation of Fitts's Law.** There is no mouse movement on the user's part to open a mouse under menu. Just click and it opens.
- **Mouse under menus shouldn't be used as a primary navigation device.** Despite their efficiency in terms of Fitts's law, mouse under menus are hidden from the users' view until they are click. There will always be a moment's delay while the user identifies the new options presented. Ideally, mouse under menus should offer the user supplementary options. The core navigation elements of the site should be kept onscreen and easily identifiable at all times.
- **There should be some indication that mouse under menus exist.** This is extremely important. If the users don't know the menu is there, they won't use it. Informing the user—either through direct means such as instructions or through hints such as a cursor change—is essential in the use of mouse under menus.
- **Dragging has to be the single biggest UI crime in Flash design.** Yes, in some instances dragging and dropping is the perfect metaphor to use. Puzzle-type games, room planner devices, and so on all use Flash's dragging capability effectively. For the most part, however, dragging in Flash is just annoying for the user. Watch out! If you want to read more about why dragging can be a drag, check out the *Skip Intro* web site (`www.skip-intro.org`).