SLIDE

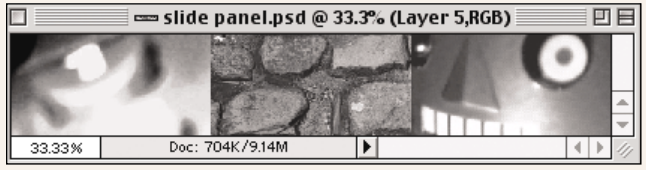
Knowing how to move an object around the screen can open up new possibilities of what you can do with your Flash designs. This chapter looks at the basic 'slide' algorithm, which can easily be built upon.

Creating Sliding Panels for Navigation and Transition Effects

The sliding panel navigation system has become quite prevalent in Flash-powered sites. Click a button and a panel slides into place to take you to a new area of the site. The goal of this chapter is not to show you techniques that have been done to death, but knowing how this effect works will give you some important principles about programmatic movement. In this chapter you're going to look at the basic principle of creating motion and then look at how you can expand on it to create some more interesting effects, and not just for navigation.

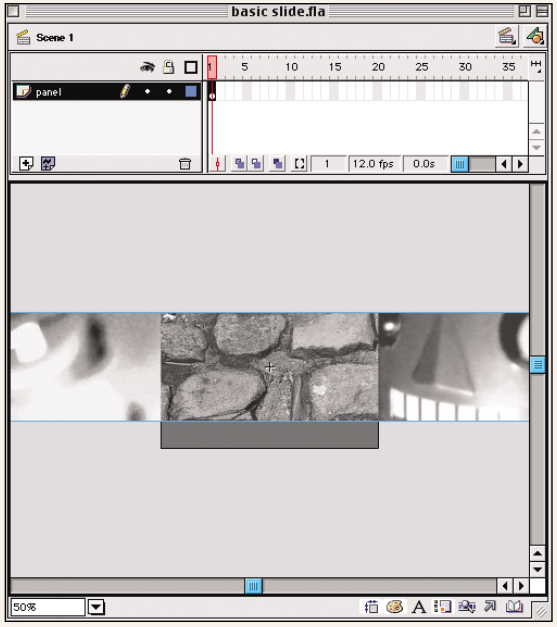
Basic Slide Algorithm

The sliding action works by manipulating the X value (or if you prefer, the Y value) of the targeted movie clip. When a button is pressed or rolled over, a variable is set which specifies the new X position of the panel movie clip. A script is then run, looping around and moving the panel a bit at a time to its new X position. And that's all there is to creating a basic slide action! But how does it seem to ease in and out during the slide, slowing down as it reaches its final X position? Well, that's really simple also, but I'll show you that later in the chapter.



3.1 Save the image as a high-quality Pict or Bitmap. You can compress it later in Flash.

1. First, you need to have something to slide. I created a 3-panel bitmap in Photoshop that consisted of three 400×250 pixel images laid out in a row 1200 pixels wide (see Figure 3.1). You can make your own or use this same one, which is available from the book's web site (`dragslidefade.com`).
2. Create a new Flash movie sized at 400×250 pixels.
3. In the first layer, import your panel image by pressing Ctrl+R (Cmd+R on Mac).
4. Select the image and then turn it into a movie clip by pressing F8. Name it **panel** and press OK.
5. With this clip selected, position it so that the middle picture in your panel strip is in the center and the upper edge is against the top of the main movie (see Figure 3.2).
6. Using the Instance panel, name it **panel**.
7. Above this layer, create a new layer and name it **script**. This is where you will put your script movie clip that will control the movement of the panel.

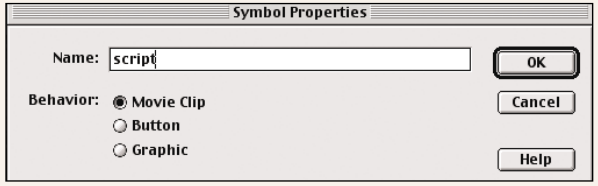


3.2 Use the Align panel to easily position the movie clip correctly.

Setting Up the Script Loop

The script loop is the main engine that will control the movement of the panel. It is a standard movie clip script loop with the main code being on one frame and a `gotoAndPlay` action on a subsequent frame to make it loop.

1. Make sure nothing is selected by pressing Ctrl+Alt+A (Cmd+Opt+A on Mac) and then press Ctrl+F8 (Cmd+F8 on Mac) to create a new movie clip. Name it **script** and press OK (see Figure 3.3).



3.3 When you create this new movie clip, you automatically go into the symbol editing mode for it.

2. Insert a keyframe (F6) at frame 5 and then one also at frame 6. You now have keyframes in your script movie clip at frames 1, 5, and 6.

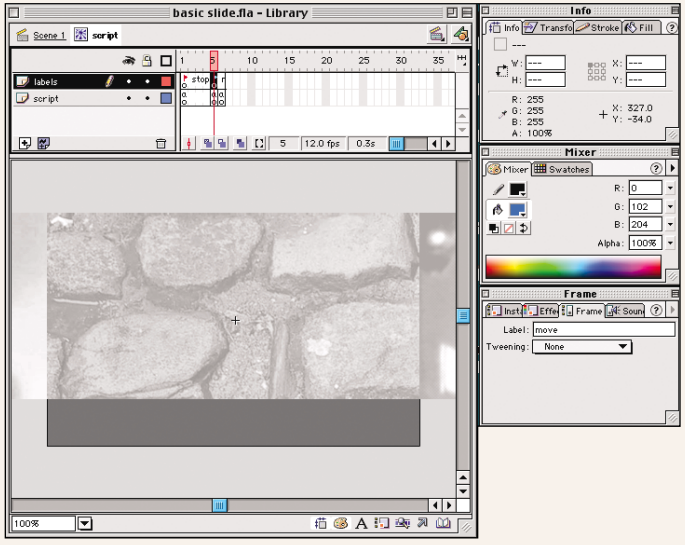
3. Name the first layer **script**. Insert a new layer above the first layer and name it **labels**.

4. In this labels layer, add a keyframe at frame 5.

5. Click frame 1 in the labels layer and use the Frame panel to give it the label of **stop**. For frame 5, give it a label called **move** (see Figure 3.4).

continues...

...continued



3.4 Adding labels makes it easier to keep track of the script setup.

These labels enable you to execute the code in the script without having to remember which frame you put your code on!

6 In the first layer (the script layer), double-click the first frame to display the Frame Actions panel for that frame.

7 This first frame does two jobs. First, put a `stop()` command in, so the movie clip doesn't automatically start when it first appears on the stage. Next, set a variable called `baseRate`. This dictates the speed of the panel slide, as you will see in a moment. So for this frame, add the following code to the Frame Actions window:

```
stop();
baseRate = 1.6;
```

Coding the Slide Algorithm

The code on frame 5 is the main piece of code that loops around, moving the panel to its target destination, either left or right, when a button is pressed. First, look at how you move the panel left if the target X position is less than the panel's current X position.

1 Double-click the keyframe at frame 5 in the script layer to display the Frame Actions panel.

2 Check to see if the panel's X position is greater than the value contained in the `targetx` variable, which is set when a button is clicked:

```
if (_root.panel._x > _root.targetx) {
```

If this is the case, then you need to move the panel to the left. To work out how much you move the panel with each pass of the loop, use a very simple algorithm.

First find out the difference in the two X positions and put that into a variable called `difference`. Say, for example, your current panel's X position is 500, and the target X position is 200. 500 minus 200 is 300. You can't simply move the panel 300 pixels to the left, as there would be no perceived movement—it would just *jump* to its new X position. So, what you need to do is divide the `difference` by a number. The number you use is the value of `baseRate`, which in this case is 1.6.

Thus, `difference` divided by `baseRate` equals `rate` in the current example:

300/1.6=187.5

This gives you the amount that you move your panel to the left. Now you're probably thinking, "Hang on, I can't simply just keep moving it 187.5 pixels to the left. It should slow down as it reaches its destination." Well the thing is, the difference is recalculated every time the script loops around. So in the given example, the next time it loops around the current X position of the panel, it will be 500 minus 187.5, which is 312.5. See how the new calculation works out as follows:

312.5–200=112.5 (`panel_x–targetx=difference`)

112.5/1.6=70.3125 (`difference/baseRate=rate`)

You can see that the amount you will move the panel is just over 70 pixels. Obviously the nearer the panel gets to its target X position, the slower the panel moves each time the loop circles around. Eventually it stops moving because the difference is zero! The effect of this is that the panel seems to ease into its final resting place. Of course you can alter the `baseRate` and try different numbers. The higher numbers move the panel more slowly, lower, and more quickly. Season to taste!

Following is the complete code for moving the panel left and right. Enter this into the current open Script window, which is frame 5 of the script layer (see Figure 3.5).

```
difference = _root.targetx - _root.panel._x;
rate = difference / baseRate
_root.panel._x += rate;
```

3.5 After putting this code into the open Script window, you are ready for the final looping command.

3 Double-click frame 6 in the script layer.

4 In this Frame Actions panel, add the following code that will keep the script looping:

```
gotoAndPlay("move");
```

That's it for the script! Now you need to add the buttons that will actually execute the code.

Adding the Buttons

The buttons actually execute the code you've just written by simply telling the script movie clip to `gotoAndPlay` the frame with the label move. You also set a few variables when a button is clicked so that the panel knows where to move.

1. Back on the main stage, drag the script movie clip into the script layer.

2. With the clip still selected, use the Instance panel to name it **script** (see Figure 3.6). This is to help reference the script from your buttons when they're clicked.

3. Create a new layer called **buttons** above the script layer.

4. Draw out a rectangle on the stage, select it, and then turn it into a button (F8). Name it something creative, such as **button**, and click OK (see Figure 3.7).

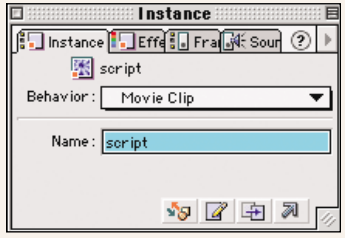
   When this button is clicked, you want it to do two things. First, it needs to set the value of the variable `targetx`. This is the new X position you want the panel to move to as follows:

   ```
   targetx = 600;
   ```

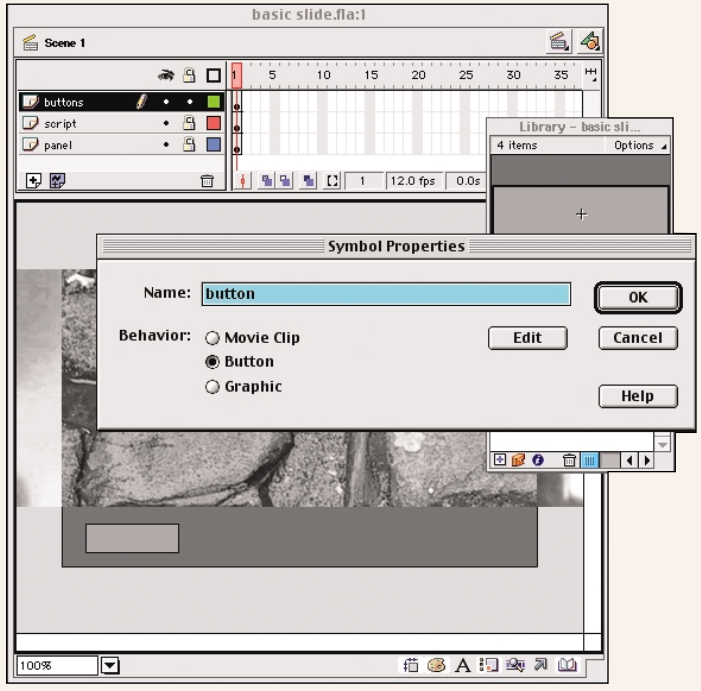
   Second, you want to actually call the script itself to move the panel as follows:

   ```
   script.gotoAndPlay("move");
   ```

5. To add this functionality to the button, select it and press Ctrl+Alt+A (Cmd+Opt+A on Mac) to display the Object Actions panel.



3.6 Always try to use descriptive instance names for movie clips.



3.7 You can add a great deal of functionality to the buttons you have created here.

6 Add an `on(release)` event handler to the Script window as follows:

```
on (release) {
}
```

This means that when the button is released (that is, clicked) the code between the curly brackets is executed.

7 Add the following code between the curly brackets to finish the button script:

```
targetx = 600;
script.gotoAndPlay("move");
```

8 Back on the main stage, copy the button and paste it twice so that you have three buttons on the stage (see Figure 3.8).

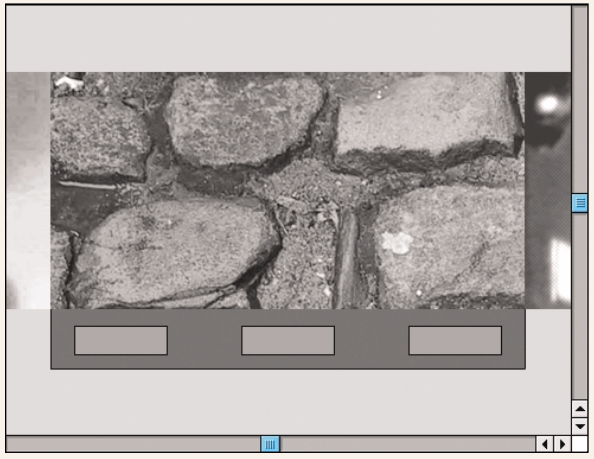
9 Select button 2, open the Object Actions panel for that button (Ctrl+Alt+A [Cmd+Opt+A on Mac]), and alter the `targetx` variable so that it now reads as follows:

```
targetx = 200;
```

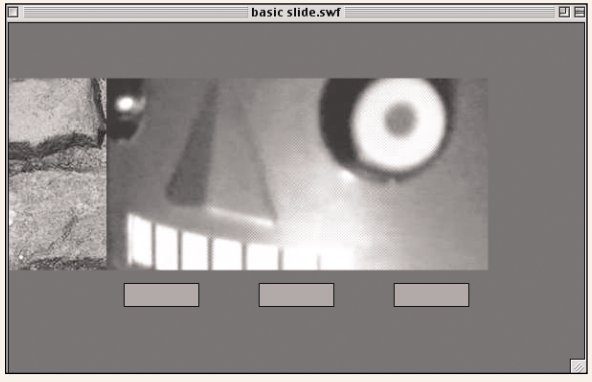
10 Do the same for the third button, but alter the `targetx` variable to **–200**. You now have three buttons that set the panel to different X positions.

11 Test your movie to see if it works. You should find that clicking the buttons sends the panels sliding left and right in a very fluid fashion (see Figure 3.9).

Try altering the `baseRate` variable to a lower or higher number to get different speeds.



3.8 The three buttons you have created will set the panels to different X positions.



3.9 The key test of successful button making is how smoothly the motions of the panels execute.

Other Uses of the Slide Algorithm

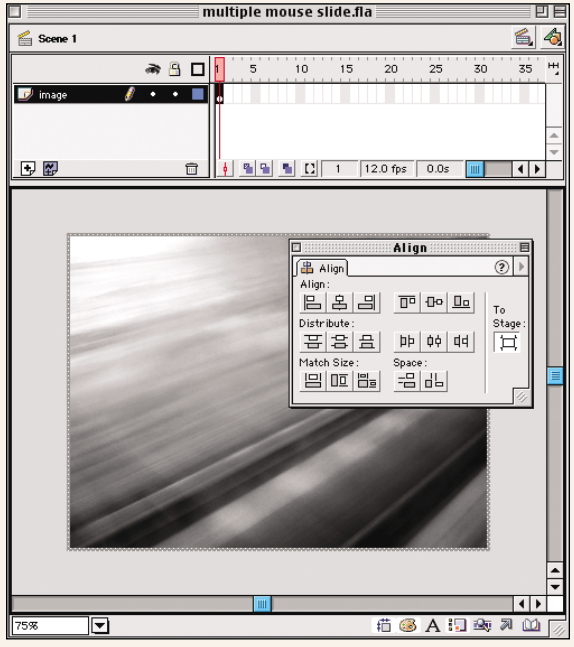
Now that you have a handle on the basic slide algorithm and how it works, you can start to use it in more inventive ways. (Versus doing a carbon copy of an effect that's probably already overused anyway.) That's what's so great about interactive design—there's always room to expand on a technique and take it to new places—it's not just limited to copying a good idea.

Multiple Mouse Slide

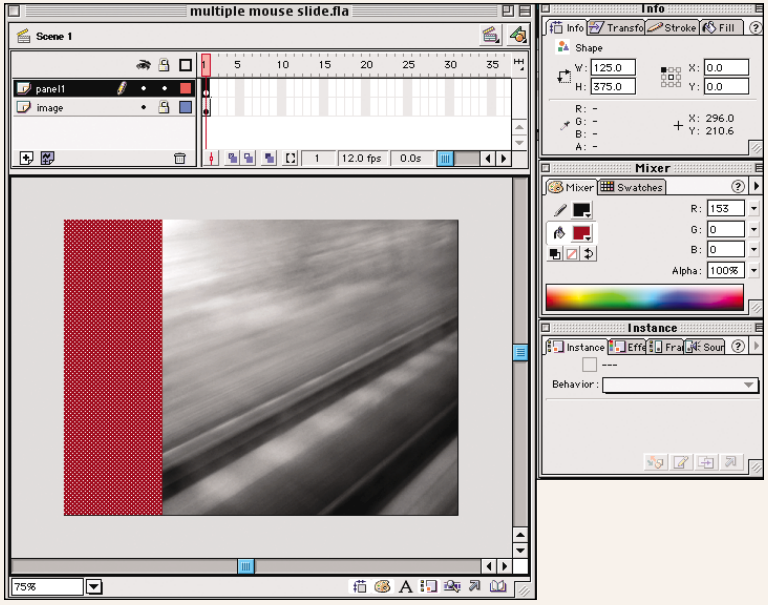
Okay, the title of this is a bit confusing, but it actually refers to moving several panels at different speeds to the mouse X position. As they reach their destination, the panels gradually fade out. Again, this looks great over the top of a bitmap. I've used a shot taken with my Canon Digital Ixus on the train to Manchester, one cold January morning. The picture is available from the book's web site (`dragslidefade.com`), but you can, of course, use any image you'd like.

Setting Up the Movie

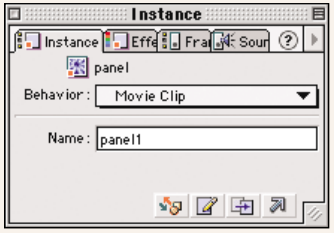
1. Create a new Flash movie sized at 500×375.
2. Rename the first layer **image** and then import your bitmap into this layer.
3. Use the Align panel to center align the image to the stage (see Figure 3.10).
4. Next you need something to slide. Create a new layer above the image layer and name it **panel1**.
5. In this layer draw a rectangular box 125×375. Use the Info panel to set the dimensions (see Figure 3.11).
6. Select the rectangle and turn it into a movie clip by pressing F8. Name it **panel** and press OK.
7. With this movie clip still selected, use the Instance panel to name it **panel1** (see Figure 3.12).
8. Create a new layer and name it **panel2**.



3.10 After the image is aligned to the stage, you can prepare for the coding that will get your movie moving.



3.11 Using the Info panel is a very easy way to alter dimensions to a certain size, rather than trying to do it by sight.



3.12 This is the first of two panels into which you will be coding movement.

9 Copy the movie clip from the panel1 layer and then paste it into the panel2 layer by choosing Edit/Paste In Place.

10 With this second panel selected, use the Instance panel to rename it **panel2**.

You now have two panels on separate layers above your bitmap image. Now it's time to do the coding to get them to move.

Coding the Multiple Movement Algorithm

The brute-force way to make this work is to simply take your previous slide code, copy and paste the same code underneath the original code, and then alter this second part so that it affects the movie clip called panel2. As a result, you would end up with four `if` statements. That's reasonable when you only have two things to slide, but what if you had 5 or 10? What I want to do is show you a much more efficient way of coding this using something you will fall in love with—functions.

Using Functions to Make Efficient Code

Functions are self-contained bits of code that are executed or "called" when you need to use them. The best way to describe it is to give you an example. Say, for example, you have a button that, when it's clicked, will increase the scale of `myClip` in the X and Y axes. You could write the following code:

```
on (release) {
myClip._xscale = myClip._xscale + 10;
myClip._yscale = myClip._yscale + 10;
}
```

This would, of course, work fine. But what if you needed to do this on several buttons? Writing the same two lines would be a pain. Instead, you can define a function in the first frame of the main timeline as follows:

```
function increaseScale() {
myClip._xscale = myClip._xscale + 10;
myClip._yscale = myClip._yscale + 10;
};
```

Now whenever you want to increase the scale of the movie clip with the button, you can simply write the following code:

```
on (release) {
_root.increaseScale()
}
```

This will have exactly the same effect, but it's much more efficient. It also means that if you wanted to amend the code—to decrease the alpha as well as the scale, for example—you could just add it to the function. Before I would have had to go into every single button and alter each piece of code.

Passing Parameters to Functions

The true power of functions is their capability for allowing you to pass parameters to them. For instance, in the previous example, the function always affects the myClip movie clip. By defining a parameter in the function, you can change which movie clip is affected as follows:

```
function increaseScale(whichClip) {
_root[whichClip]._xscale = _root[whichClip]._xscale + 10;
_root[whichClip]._yscale = _root[whichClip]._yscale + 10;
};
```

Define the parameter inside the brackets following the function name. I've called it `whichClip`. Inside the function itself, use the parameter the same as any other variable. Notice that the clip is referenced in a slightly different way as follows:

```
_root[whichClip]
```

Basically this is the same as writing `_root.myClip`, but you can't literally write `_root.whichClip` because it would try and find a movie clip with an instance name of whichClip. So, instead use a different way of referring to the clip. To normally access myClip written this way we would enter the following:

```
_root["myClip"]
```

Instead, replace it with a variable. In this case, it's `whichClip` (notice there are no inverted commas).

So, how do you set the `whichClip` parameter so that it contains the instance name myClip. Do this when you call the function from your button as follows:

```
on (release) {
_root.increaseScale("myClip")
}
```

If you want to affect a different movie clip with the same function, you would simply enter the following:

```
on (release) {
_root.increaseScale("anotherClip")
}
```

Those are the basics of using functions. We'll cover more ground with them later. Now move on to coding the multiple slide.

Defining the Slide Function

You can now add the main function to a frame on the main timeline, ready to be called when needed. I usually place all my functions in the same frame on a layer named "functions." This makes it easier to find code when I return to a project months later. Remember, even though your function will be set on the first frame of the movie, they will not execute until they are "called."

1. On the main stage create a new layer and name it **functions**. This layer will contain your function.

2. Double-click the first frame of this new layer to bring up the Frame Actions panel. In this window you will define the function.

   Functions are always defined in the same basic way. First, write the actual word **function**. This tells Flash that what follows is a function definition.

   Following the word "function" is the name of the function itself. This is the name you use to call or execute the function when you want to use it. Function names cannot include spaces, so if you need to, you can use an underscore instead. I have a basic syntax I always employ when writing function names. In this example you are going to create a function called `movepanel`. To make for easier reading, I always start the second word and any subsequent word with a capital letter (for example, `movePanel`). This has become a common standard amongst JavaScript and the like, and it's not a bad thing to employ the same sense here.

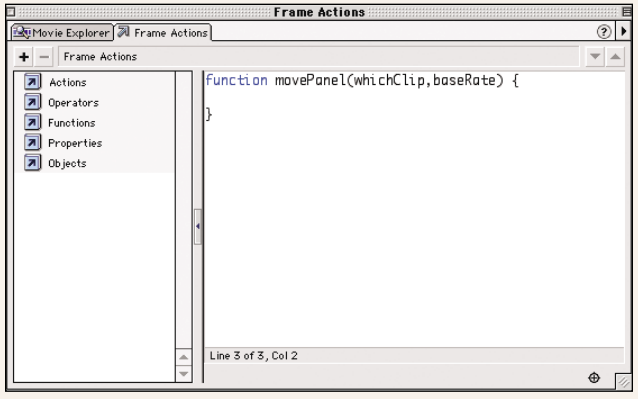
   After the function name come the parameters for the function, held within parentheses. You don't have to define parameters, but you always need the parentheses. You can have more than one parameter. In fact, in this example you're going to use two—one called `whichClip` and the other called `baseRate`. A comma separates these two parameters when you define them as follows:

```
function movePanel(whichClip,baseRate)
```

   The final part of the basic function definition is adding the curly brackets. These go at the beginning and end of your function and define any code between them as belonging to that function as follows:

```
function movePanel(whichClip,baseRate) {
code goes here
}
```

3. Add this function definition into the open Script window. Your script should look like Figure 3.13.



3.13 When your script looks like this, you then need to add some previous script, but with modifications that will determine the movie clip you call.

continues...

...continued

4 Add your basic slide script from the previous example into this function, but with three slight changes. Wherever there is mention of `_root.panel`, you need to change it to read `_root[whichClip]`. You do this so that you can dynamically change which movie clip is affected when you call the function (remember, `whichClip` is a parameter of this function). The second change is that your target destination is no longer the variable `targetx`. Instead, you want the target destination to be the X position of the mouse. To do this use the built-in property `_xmouse`. The third change is that a line of code has been added that sets the panel clip's alpha setting to the value of `rate`. Remember, `rate` is constantly decreasing as it reaches its destination. This will result in the panel fading to nothing as it comes to rest.

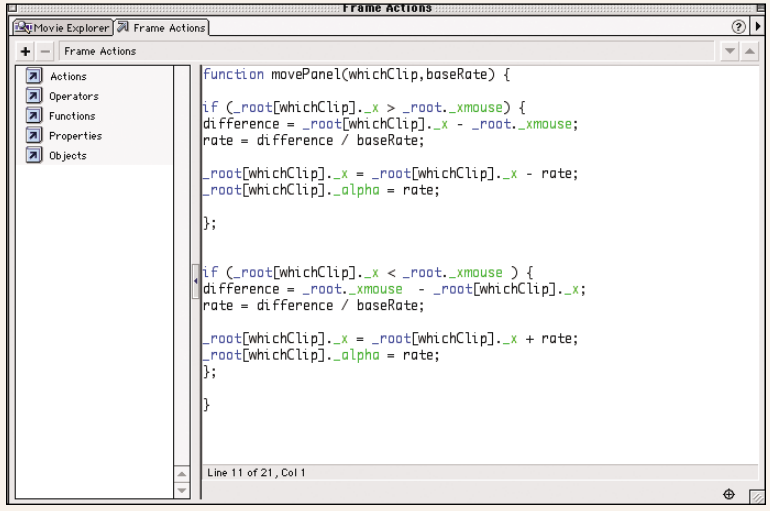
```
_root[whichClip]._alpha = rate;
```

The full code is as follows. Your script window should now look like Figure 3.14.

```
function movePanel(whichClip,baseRate) {

difference = root._xmouse - _root[whichClip]._x;
rate = difference / baseRate * 2.3

root[whichClip]._x += rate;
}
```



3.14 When your Script window looks like this, you're ready to script the looping.

Building a Script Loop

All that remains now is to build a simple script loop movie clip to trigger the function.

1. Create a new layer above the functions layer and name it **script loop**.
2. Make sure nothing is selected and press Ctrl+F8 (Cmd+F8 on Mac) to create a brand-new movie clip. Name it **script loop** and press OK.
3. When you create this new clip, it automatically opens, and is ready to be edited. Double-click the first frame in this movie clip to display the Frame Actions panel.
4. This first frame is where the function will be called as follows:

```
_root.movePanel("panel1",3);
```

When the script sees this line of code, it will execute the function called `movePanel`, which is on the main timeline (`_root`). When you call the function, you set panel1 as the movie clip you want to be affected. You also define the `baseRate` of the movement as **3**.

Now don't forget you have two panels to move. No problem! Just put another function call after the first, but this time specify the panel2 movie clip, and with a slightly higher `baseRate` of **4** (so this panel moves more slowly) as follows:

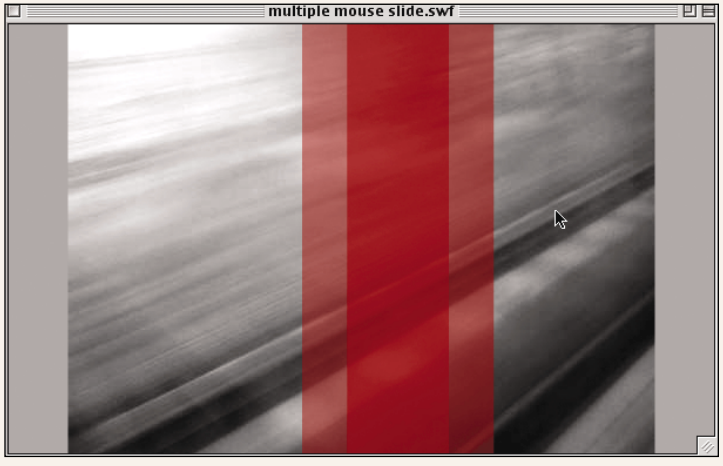
```
_root.movePanel("panel2",4);
```

This is the power of functions. The same code written once is doing two different things, per the parameter's instructions.

5. Add a keyframe (F6) on frame 2 and then double-click it. Add a `gotoAndPlay(1)` action as follows so that this script keeps looping:

```
gotoAndPlay(1)
```

6. Back on the main stage, drag the script loop clip you've just created into the script layer.
7. Test the movie. As you move your mouse across the movie, the panels slide across to the current X position of the mouse, gradually fading away. See Figure 3.15.

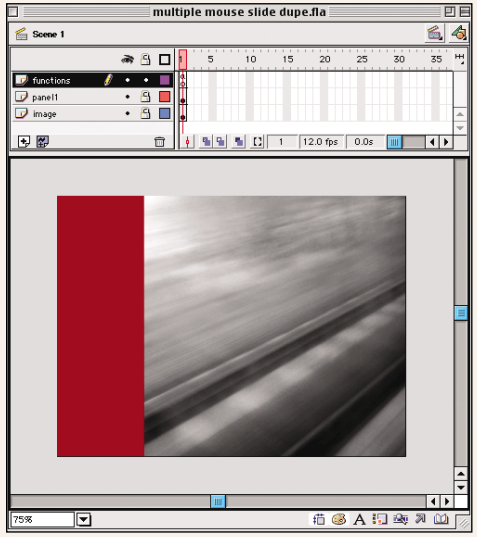


3.15 Try altering the speed of the panels by changing the numbers in the function calls. A higher number makes the panels move more slowly; a lower number makes them move faster.

Further Streamlining the Code

The code you've put together so far is pretty efficient. One engine—the function—is driving all the panel movement. Next I want to show you a way to automate the number of panels on the stage, each with its own speed. What I mean is, you only create one master panel on the stage. When the movie is run, the panel is duplicated as many times as you want. In this example you'll make 10 panels, each one progressively slower than the first. What I want to get across is how, after you've completed your code, there is nearly always a way to fine-tune it and make it more efficient.

1 You are not going to need your script loop. Instead, make use of a movie clip `onClipEvent` handler. In your previous movie, delete the script layer and the panel2 layer. You should be left with just three layers (see Figure 3.16).



3.16 Another efficient coding method is to just delete a script layer and utilize an `onClipEvent` handler.

To call the function and move the panel, you are going to put the function call inside an `onClipEvent(enterFrame)` event for the panel movie clip; so that when you come to duplicate the movie clip on-the-fly, the function call is self-contained inside each panel movie clip.

One difference with the function call is that you need it to dynamically know what its own instance name is. You can do this by using the movie clip property `_name`. When you call the function, instead of hard coding the instance name, you can enter the following:

```
_root.movePanel(this._name,3);
```

You see that the code includes `this._name`, whereas before you wrote panel1 or panel2. Because this code will be attached to each individual clip, use `this._name` to get the instance name of the movie clip that the code is attached to.

2 Select the movie clip on the panel1 layer and press Ctrl+Alt+A (Cmd+Opt+A on Mac) to display the Object Actions panel.

3 Add the `onClipEvent` handler with the function call as follows:

```
onClipEvent(enterFrame) {
_root.movePanel(this._name,speed);
}
```

If the movie clip doesn't have a stop action in it (and this one doesn't), the clip event will continually loop, executing the code in between the `onClipEvent` curly brackets. This has the same effect as your previous script loop movie clip.

Now you should notice that instead of using a number for the speed, as you did before, you've used a variable. This is so that each time this clip is dynamically duplicated, the speed for each clip will get steadily slower. You'll learn how exactly you do that in a moment.

Duplicating the Movie Clip On-the-Fly

To duplicate the movie clip many times, when the movie is run, you need to add some code to the main timeline. This code is run only once.

To duplicate a movie clip, use the built-in function `duplicateMovieClip`. This function has two parameters. The first is the new name of the duplicated movie clip. The second is the level number of this new clip. Note, however, that you can only have one movie clip per level. Any new clip put onto a level where there is an existing movie clip will delete the old clip. So, to duplicate the panel1 movie clip, for instance, you could enter the following code:

```
_root.panel1.duplicateMovieClip( "panel2",2)
```

This would duplicate the movie clip with the instance name of panel1 and give it a new instance name of panel2. This new clip is put onto level 2. When you duplicate a movie clip, you get an exact copy, complete with all the attached `onClipEvent` handlers, and in exactly the same place on the stage.

Using a Loop to Duplicate Movie Clips

In your movie, you want to duplicate the original movie clip (panel1) 10 times. To do this, use a `for` loop as follows:

```
for ( loopCount = 1; loopCount < 10; loopCount ++ ) {
}
```

`for` loops repeat code that is in between its own curly brackets *until* a certain condition is met. In the previous example, a variable called `loopCount` was set to 1 as follows:

```
loopCount = 1;
```

This is its starting value. The loop continues as long as `loopCount` is less than 10 (the `condition`).

```
loopCount < 10;
```

So, how do you increment the value of `loopCount`? This is done by the last part of the `for` loop as follows:

```
loopCount ++
```

This is exactly the same as writing `loopCount=loopCount+1`. It's just a shorthand way of writing it.

You can now add the code that you want repeated—inside the `for` loop's curly brackets as follows:

```
_root.panel1.duplicateMovieClip( "panel" + loopCount, loopCount)
```

continues…

...continued

Notice that the new instance name you give each duplicated clip is made from taking the word "panel" and then sticking the current value of `loopCount` on the end:

```
"panel" + loopCount
```

So if `loopCount` is 3, then the instance name for this new clip is panel3. Consequently, the next time the loop comes around, the new instance name for the next duplicated clip is panel4.

You also use the value of `loopCount` to set the level of each new duplicated clip. Because `loopCount` is increased with every pass of the loop, each new duplicated movie clip will be on its own level.

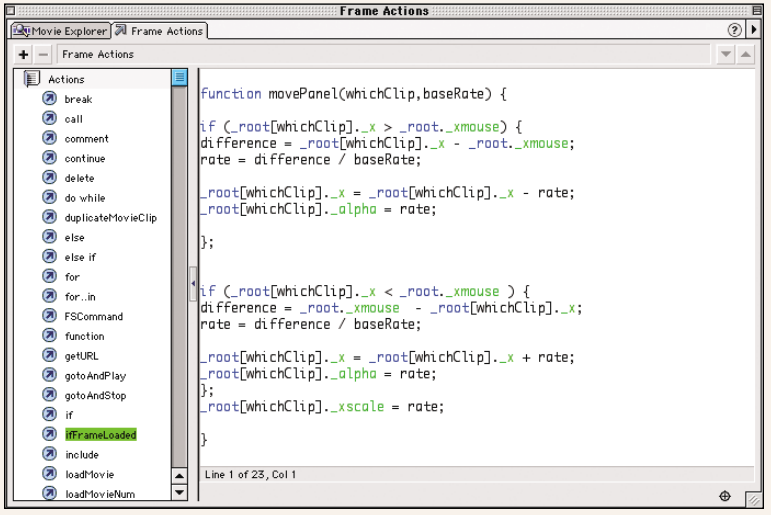
You need to add this code into your main timeline, and you might as well put it into the function layer. The following steps show you how to do this:

1. Double-click the first frame in the functions layer to display the Frame Actions panel. You should see the existing function (see Figure 3.17).

2. Add the following code into the Script window, making sure it's above the `movePanel` function:

```
for ( loopCount = 2; loopCount < 10; loopCount ++ ) {
_root.panel1.duplicateMovieClip( "panel" + loopCount,
➥loopCount)
}
```

The `loopCount` variable is set to 2 rather than 1, as this is the level at which you want to start duplicating the movie clips.



3.17 If this is what you see on your screen, adding a little more code will enable you to start duplicating movie clips.

Changing the Speed of Each Panel

Each successive panel needs to be slightly slower than the one before. If you remember from the earlier example, you can alter the speed of each panel by increasing the number in the `movePanel` function call (the higher the number, the more slowly the panel moves). Because you are duplicating panels on-the-fly, you want to automatically set each panel at a slower pace than the last as they are duplicated. To do this, make use of the `onClipEvent(load)` event handler, which you attach to your first panel.

1. Select the panel movie clip on the panel1 layer. Press Ctrl+Alt+A (Cmd+Opt+A on Mac) to display the object actions for this clip.

2. You should see the `enterFrame` clip event script in this window (see Figure 3.18).

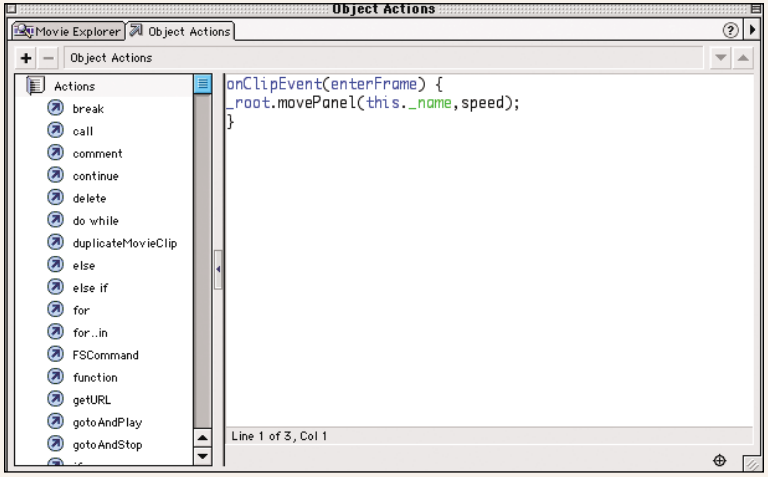
3. Add the following code to the Script window:

```
onClipEvent(load) {
        _root.counter = _root.counter + 1;
        speed = _root.counter
}
```

When the movie clip loads, it increases a variable called `counter` on the main timeline by an increment of one:

```
_root.counter = _root.counter + 1;
```

This helps you keep track of how many of these panels you have on the stage. Each time the panel is duplicated, this script runs and increases the value of `counter`. This variable will always match the number of panel movie clips on the stage.



3.18 You can duplicate panels and set their movement speeds on-the-fly with just a little scripting.

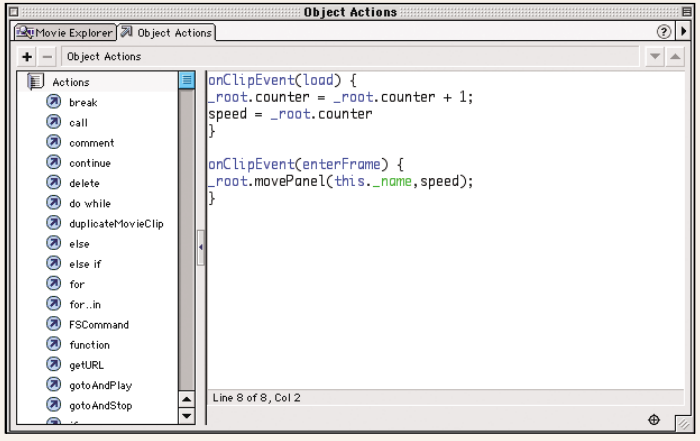
continues...

...continued

4 Use this information to set a variable called `speed` as follows:

```
speed = _root.counter
```

The `speed` variable is then used in the function call in the `enterFrame` event handler to set the speed of the panel. Your Script window should now look like Figure 3.19.



3.19 The order in which the `onClipEvents` appear in the Script window doesn't matter.

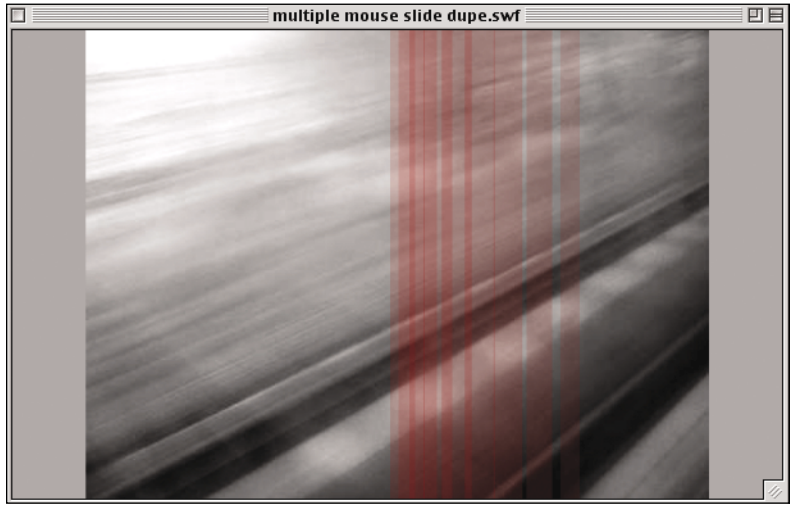
Adding Scale to the Effect

Before you test the movie, add the following piece of code into the `movePanel` function as the last line:

```
_root[whichClip]._xscale = rate;
```

This will have the effect of scaling down the `xscale` of each panel as it moves to its final resting place (remember the value of `rate` is constantly decreasing). This creates quite a good transition effect, but play with it to suit your taste.

Now test your movie. You should see 10 panels move across the stage to where the mouse rests, all with varying degrees of speed (see Figure 3.20).



3.20 This test gives you 10 panels in a parade on your screen, but it is easy to add more.

Do you want more than 10 panels? No problem. Simply increase the number in the `for` loop. For instance, if you want 20 panels, just change the condition in the `for` loop to the following:

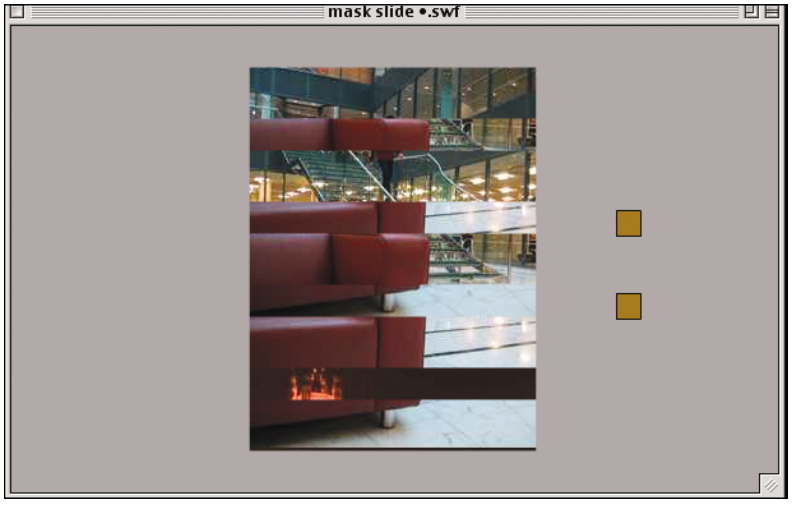
```
loopCount < 20;
```

This is the real beauty of designing with ActionScript. By changing just a few numbers, you can create different effects, quickly and easily. You can see how we have taken one basic slide algorithm, and used it not just for navigation, but also for visual effects. But there's still more you can do!

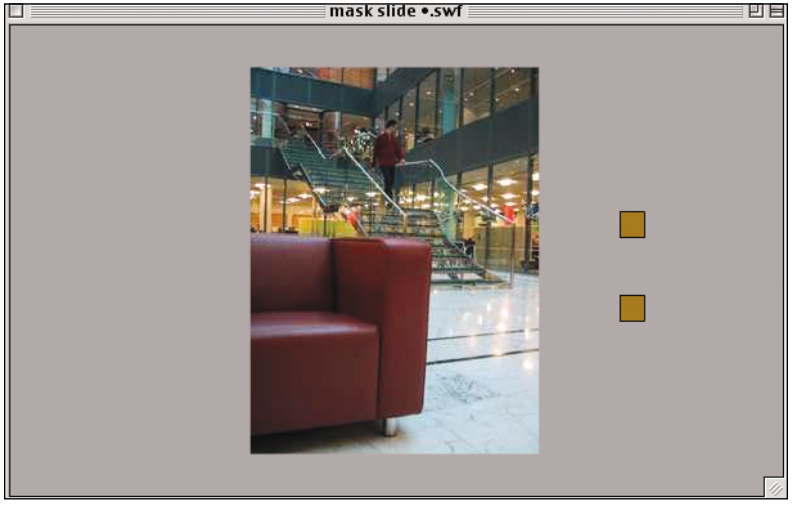
The Sliding Mask Transition Effect

By combining multiple panels with masks applied to them, you can create some interesting transition effects. This always looks great with photographs, and World Domination Design Group (WDDG) used it brilliantly on its John Mark Sorum web site.

The basic principle is this: The stage contains two identical movie clips that contain your pictures laid out vertically with one above the other. Clicking a button slides the panels to a new Y position, with the second panel moving slightly more slowly than the first. Each movie clip having its own mask applied to it achieves the transition effect. When both movie clips are in the same position, you can't visibly see the mask. But when you slide the clips, you see the masks because panel 2 is taking longer to reach its destination (see Figures 3.21 and 3.22).



3.21 The two movie clips move toward their target position…

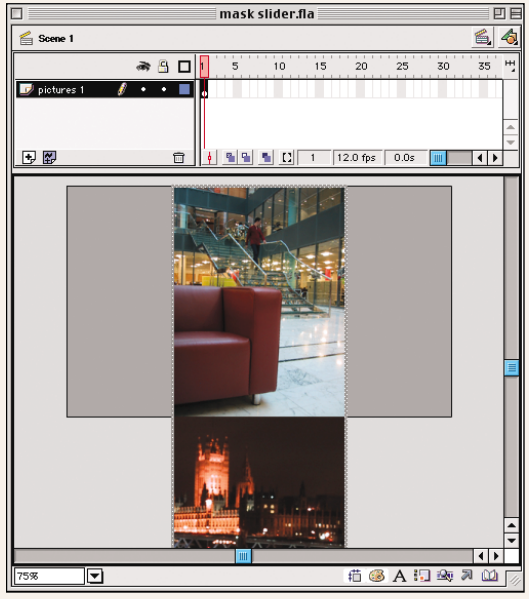


3.22 …and then finally come to rest, revealing the new picture.

Setting Up the Main Stage

I'm going to show you how to build a function that can achieve this effect, not just in the Y-axis but also in the X-axis. You'll also expand on the technique to create some other effects.

1. Create a new movie sized at 500×300.
2. Rename layer 1 **pictures1**, and then import a suitable strip of pictures. I've used pictures that are 300 pixels tall. You can download the two-picture strip I've used from `dragslidefade.com` (see Figure 3.23).
3. Select the picture strip and press F8 to turn it into a movie clip. Name it **pictures** and press OK.
4. Create a new layer above this one and name it **function**.
5. Double-click the first frame in this layer to show the Frame Actions panel. This is where you will create the main function to move the panels.



3.23 Go to `dragslidefade.com` to download this or other strips of pictures.

Creating the Function for X and Y Movement

This function is basically the same as your previous function, except you specify a target X position and a target Y position for the movement. This means you can move panels diagonally as well as up and down or left and right. The X and Y target positions are defined as new parameters in your function definition as follows:

```
function movePanel(x,y,whichClip,baseRate)
```

When you call the function, you set the X and Y positions to where you want to move the panel as follows:

```
function movePanel(100,300,this._name,1.6)
```

Before, the panel headed towards the X position of the mouse, this time it moves towards the value of X and Y.

The full function follows. The Y movement is the same as the X movement, only this time you alter the _y property. Add the following code to the Script window:

```
function movePanel(x,y,whichClip,baseRate) {
        if (_root[whichClip]._x > x) {
                difference = _root[whichClip]._x - x;
                rate = difference / baseRate;
                _root[whichClip]._x = _root[whichClip]._x - rate;
        }
        if (_root[whichClip]._x < x ) {
                difference = x  - _root[whichClip]._x;
                rate = difference / baseRate;
                _root[whichClip]._x = _root[whichClip]._x + rate;
        }
        if (_root[whichClip]._y > y) {
                difference = _root[whichClip]._y - y;
                rate = difference / baseRate;
                _root[whichClip]._y = _root[whichClip]._y - rate;
        }
        if (_root[whichClip]._y < y ) {
                difference = y  - _root[whichClip]._y;
                rate = difference / baseRate;
                _root[whichClip]._y = _root[whichClip]._y + rate;
        }
}
```

Calling the Function

You need to create a script loop that constantly calls the function and moves the panels into their target positions when a button is clicked. You could do this with a conventional movie clip script loop, but instead you'll do what you did in the previous example and use an `onClipEvent` handler on the pictures movie clip itself:

1. Select the pictures movie clip. Use the Instance panel to name it **pictures1**.

2. With this movie clip still selected, press Ctrl+Alt+A (Cmd+Opt+A on Mac) to display the Object Actions panel for this movie clip. You can add the `onClipEvent(enterFrame)` handler in this Script window.

   The main point of this script is to loop around and call the function each time so that it moves the panel to its target destination. But you only want it to do this when a button is clicked. Because of this you are going to use a variable of type boolean. This type of variable can only be set to `true` or `false`. When you click a button, you will set a variable called `pressed` to `true`. If this variable is `true`, you know you need to move the panel.

   ```
   if (_root.pressed == true) {
   ```

   Accordingly, call the function as follows:

   ```
   _root.movePanel(_root.targetx,_root.targety,this._name,1.5);
   ```

   The full script for this is as follows:

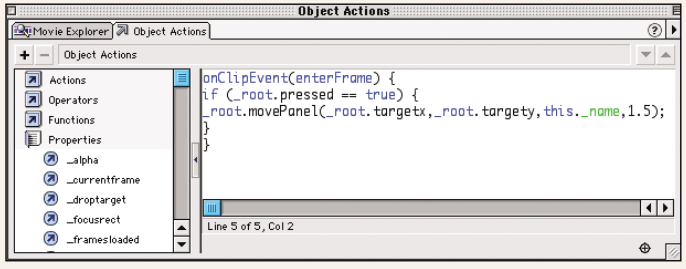
   ```
   onClipEvent(enterFrame) {
           if (_root.pressed == true) {
   ```

continues…

...continued

```
                _root.movePanel(_root.targetx,_root.targety,this
        ➥._name,1.5);
            }
    }
```

Your code should look like Figure 3.24.



3.24 This is the code you will need to begin to call the functions.

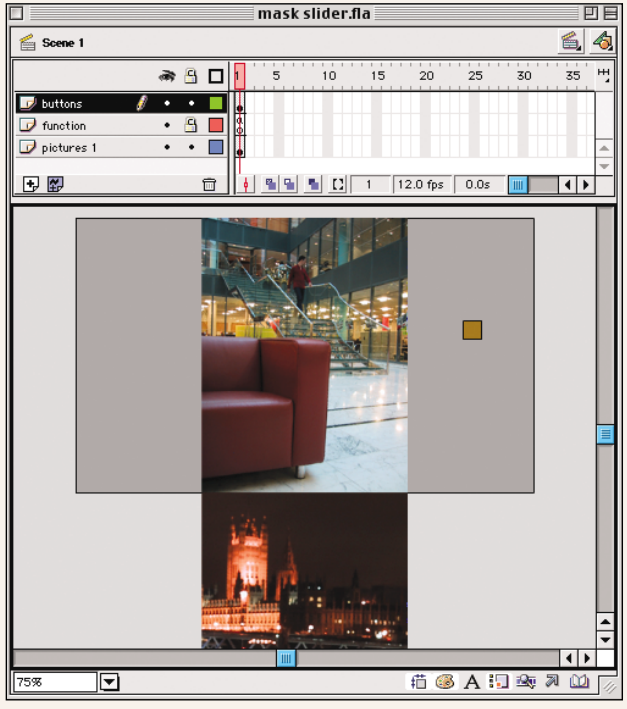
Notice that when you call the function, you don't actually specify the target X or Y position. You tell it to use the values of two variables—`targetX` and `targetY`. These variables are set on the main timeline when a button is clicked, so you can easily alter the destination of the panels with the ActionScript you attach to the buttons.

Of course, you need to set the value of `pressed` to `false` when the movie first starts; otherwise, the panels will start to move immediately, rather than waiting for a button click. To set it to `false`, open the script in the functions layer and add this line of code *above* the function:

```
_root.pressed = false;
```

Triggering the Function with a Button

1. Create a new layer and name it **buttons**.
2. Draw a small box on the stage in this layer (see Figure 3.25).
3. Select this box and turn it into a button by pressing F8 and choosing the Button radio button. Name it **button**. Click OK.



3.25 Buttons can be programmed to call a variety of functions, including multiple functions with a single click.

When this button is clicked, it needs to do three things. First, it must set the value of the `targetx` variable. If you don't actually want to move it across the X-axis, you can simply tell it to stay put by using the panel's current X position as follows:

```
_root.targetx = pictures1._x;
```

Next, set the value of the `targetY` position as follows:

```
_root.targety = 0;
```

Finally, set the value of `pressed` to `true` as follows, causing the function to kick in:

```
_root.pressed = true;
```

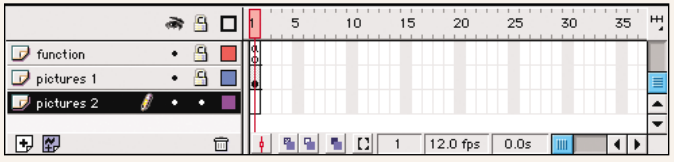
This all needs to go in an `on(release)` event handler. To do this, select the button and press Ctrl+Alt+A (Cmd+Opt+A on Mac) to display the Object Actions panel for this button. Add the following script:

```
on (release) {
        _root.targetx = pictures1._x;
        _root.targety = 0;
        _root.pressed = true;
}
```

Setting Up the Masks

So far you've created a movie clip with your pictures in it, created the main function, and made a button to move the panel. For your transition effect to work, you need to have two identical panels—the second moving slightly more slowly than the first.

1. Select the pictures1 movie clip and copy it (Ctrl+C [Cmd+C on Mac]).
2. Underneath the layer with the pictures1 movie clip on it, create a new layer and name it **pictures2**. (see Figure 3.26).



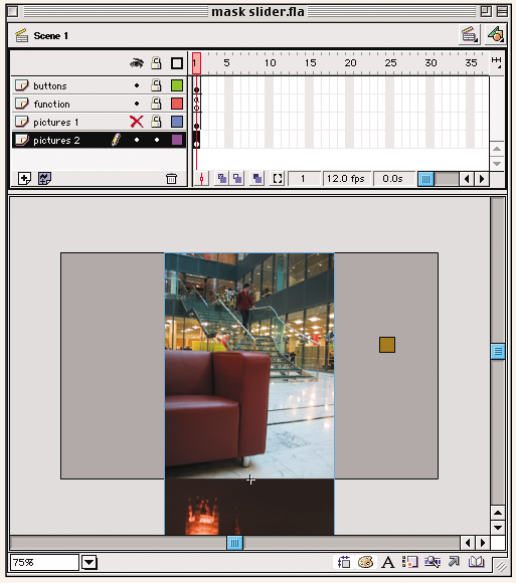
3.26 Make sure this layer is underneath the pictures1 layer.

continues…

...continued

3 Paste the copied movie clip into this layer (Ctrl+Shift+V [Cmd+Shift+V on Mac]). You now have the second copy of the pictures movie clip in place. You might want to turn off the visibility for the pictures1 layer for the time being so you can just see this new clip (see Figure 3.27).

4 With this new, copied panel selected, rename it in the Instance panel as **pictures2**. Each clip should have its own name so that it doesn't confuse the script.

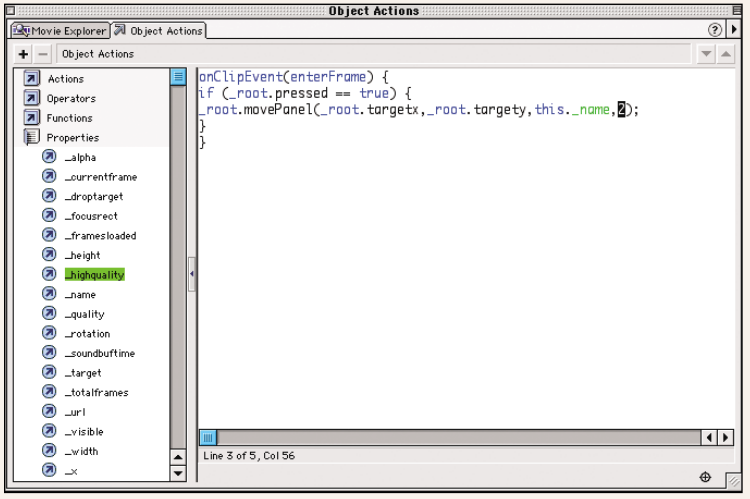


3.27 Turn off the visibility of the pictures1 layer to see the layer underneath, or alternatively turn that layer into an outline.

Now that you have the second panel in place, you need it to move more slowly than the first panel. All you have to do is alter the `movePanel` function call attached to this clip.

5 With the clip still selected, press Ctrl+Alt+A (Cmd+Opt+A on Mac) to display the Object Actions panel. You should see that it already has an `enterFrame` event handler attached to it. This is because this clip is a copy of the original.

6 To slow this clip down, increase the number at the end of the function call. Remember, the higher the number, the more slowly the panel moves. The original was set to 1.5, so set this to **2** (see Figure 3.28).

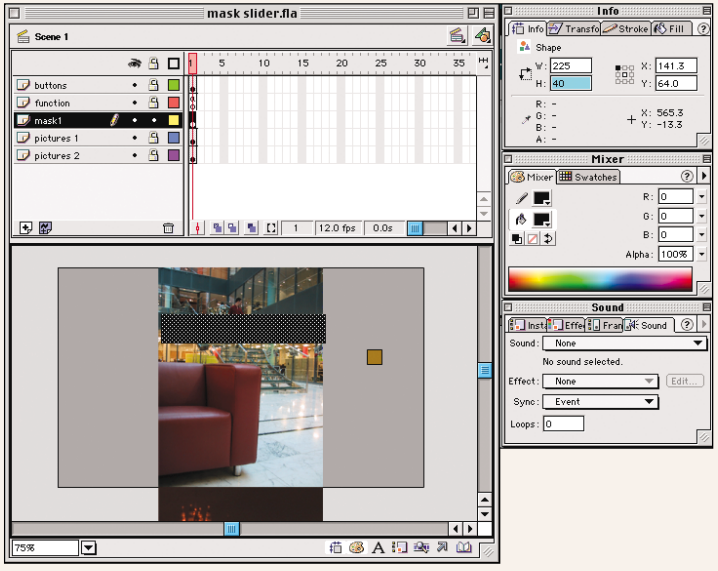


3.28 Play around with these numbers to alter the speed of the effect.

Creating the Mask Shapes

You next need to create the actual mask shapes for the two picture layers. Remember that the two masks must form a whole when shown together. Other than that, they can be any shape you desire.

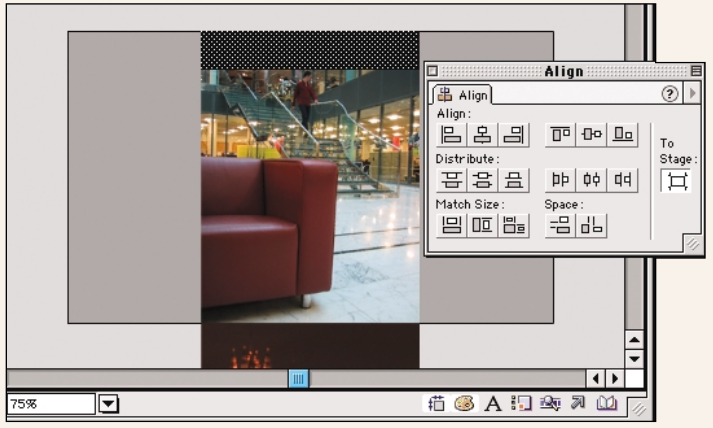
1. Create a new layer above the pictures1 layer and name it **mask1**. You will create the first of the two masks on this layer.
2. Draw a rectangle on this layer. Use the Info panel to make the rectangle 225×40 (see Figure 3.29).



3.29 The rectangle you create here begins the mask creation.

3. With this rectangle still selected, use the Align panel to align it to the upper edge and center it on the stage (see Figure 3.30).

   This shape needs to be copied four times, with each of the five blocks spaced evenly across the vertical space of the stage.
4. With the rectangle selected, copy it (Ctrl+C [Cmd+C on Mac]).
5. Paste it in place (Ctrl+Shift+V [Cmd+Shift+V on Mac]).



3.30 After the rectangle is aligned on the stage, you then begin the replication.

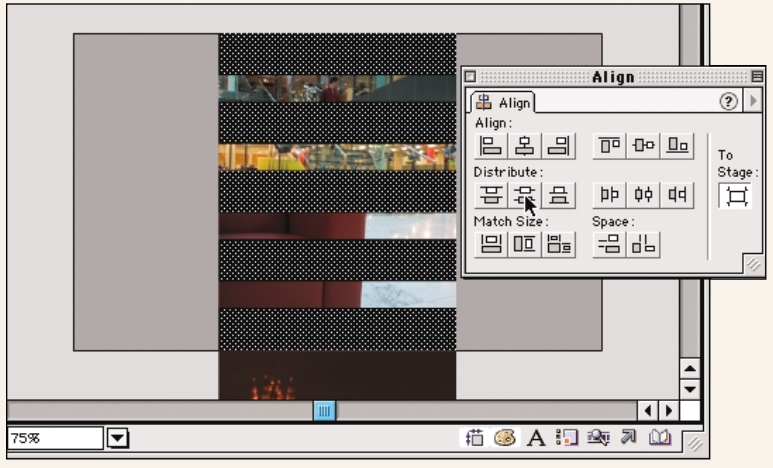
continues…

...continued

6 Without deselecting this new rectangle, use the arrow keys to move it underneath the original (see Figure 3.31).

Repeat this procedure for this new rectangle until you have five rectangles spread out across the stage from top to bottom. Align the lower-most rectangle so it's at the bottom of the picture. You'll want to space them evenly across the stage. To do this, select all the rectangles. The best way to do this is to lock all the other layers and press Ctrl+A (Cmd+A on Mac) to select all the objects in the mask1 layer. Now use the Align panel and make sure the To Stage option is depressed. Click the Distribute Vertical Center button, and the rectangles will be spaced evenly in the vertical direction (see Figure 3.32).

3.31 If your alignment matches this figure, you're ready to copy and space the new rectangles.



3.32 Make sure the To Stage button is depressed.

7 To turn this layer into a mask for the pictures1 layer underneath, first deselect all the panels on the mask layer by pressing Ctrl+Shift+A (Cmd+Shift+A on Mac).

8 Right-click (Ctrl+click on Mac) the mask1 layer name and choose Mask from the popup (see Figure 3.33).

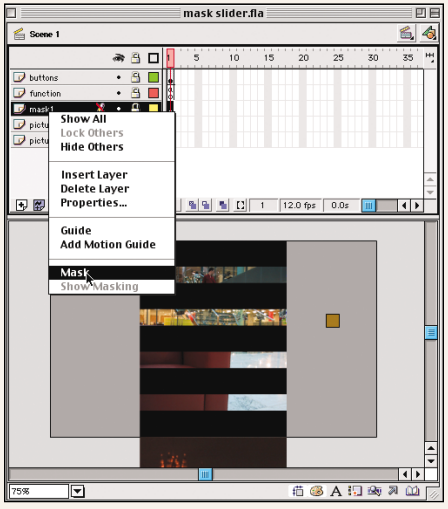
The picture on the pictures1 layer is now masked off, with the only areas showing through now being the areas defined by the rectangles. To see the mask effect, turn off the visibility of the pictures2 layer (see Figure 3.34).

You also need the pictures2 layer to be masked in exactly the same way, but its mask should be the opposite of the pictures1 mask. That is, to the point where you can't see the image in pictures1; you should be able to see the image in pictures2.

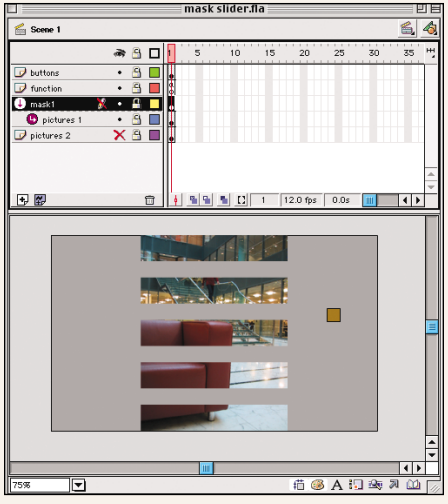
9 Unlock the mask1 layer, and select all five rectangles and copy them (Ctrl+C [Cmd+C on Mac]).

10 Lock that layer again to turn the mask back on. Above the pictures2 layer, add a new layer and name it **mask2**.

11 Paste the copy (Ctrl+Shift+V [Cmd+Shift+V on Mac]) of the five rectangles into this layer. You won't see them appear, however, because they will be covered by the pictures1 layer above them.



3.33 After you have masked off the picture...



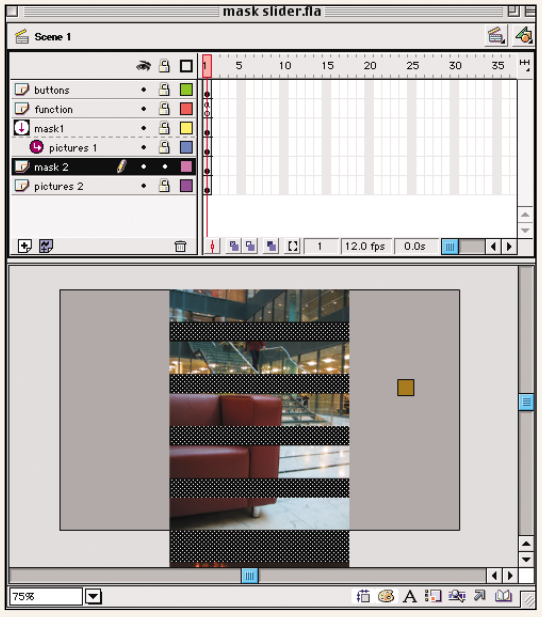
3.34 ...you can evaluate the mask effect.

continues...

...continued

12 Use the arrow keys to move all five rectangles down so you can see them (see Figure 3.35).

13 Finally, turn this layer into a mask for the pictures2 layer by repeating Steps 7 and 8.



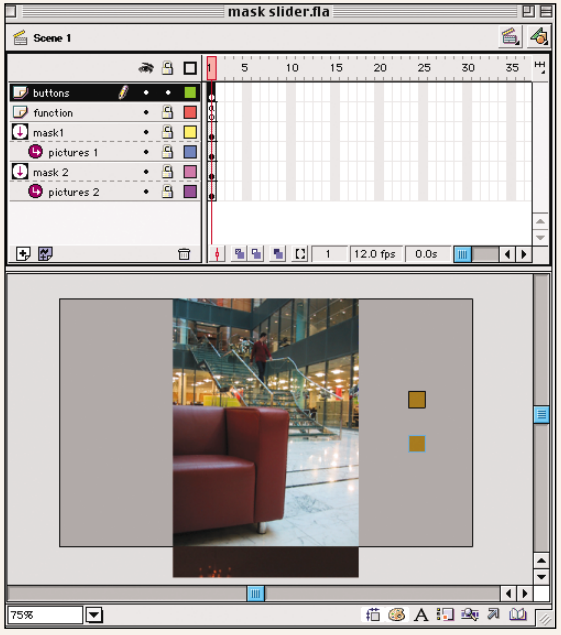
3.35 If you now are looking at all five rectangles, creating the mask is simple.

Adding the Final Button

Now you have your two movie clips masked off with alternate masks, and you have a button in place to move the strip upward. What you need now is a button to move the strip back down again.

1 In the button layer, select the small button, copy it (Ctrl+C [Cmd+C on Mac]), and then paste it in place (Ctrl+Shift+V [Cmd+Shift+V on Mac]).

2 Use the arrow keys to move it below the original button (see Figure 3.36).



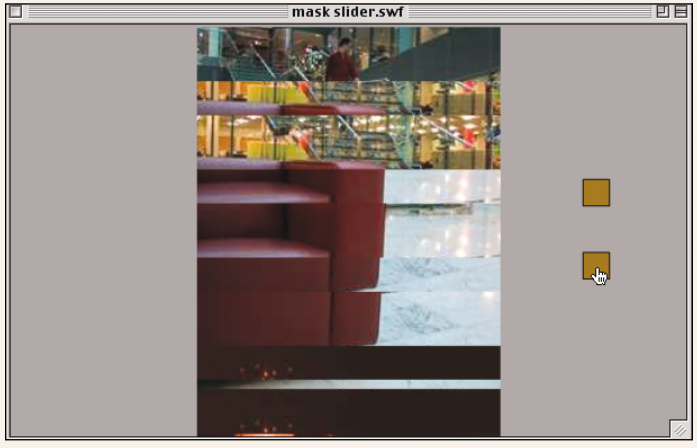
3.36 Building the down button is the final step.

3 With this new button still selected, press Ctrl+Alt+A (Cmd+Opt+A on Mac) to open the Object Actions panel.

4 In the `on(release)` script, change the `targetY` position value to **300**. This means that when this button is clicked, the panel is moved so that its Y position will equal 300. The full code for the button is as follows:

```
on (release) {
        _root.targetx = pictures1._x;
        _root.targety = 300;
        _root.pressed = true;
}
```

5 Now test the movie. Click on the top button and the panel will slide upwards, giving a venetian blind type transition effect. Click the bottom button and it slides back to its original position (see Figure 3.37).



3.37 If your code is in place, you should be able to move the panels up or down.

Adding to the Transition Effect

After you have the basic effect working, try altering the speed of the panels by changing the numbers in the function calls. Also try making some more interesting mask shapes, rather than simple boxes. Remember, you also can move the images left, right, and diagonally by specifying a `targetX` position. For example, the best way to utilize this is to have nine images laid out in a 3×3 grid. You can then set up nine buttons that move the images up, down, left, or right, all without any additional programming to the main function.

Adding Rotation

One addition you can make to the `movePanel` function is rotation. When the panel is moved, you can make the rotation equal to the `rate` variable. This makes the image spin into its resting place!

1 Double-click the first frame of the functions layer to display the `movePanel` function.

2 Inside this function, add the following piece of code to the last line of the function (before the last curly bracket):

```
_root[whichClip]._rotation = rate;
```

This makes the `_rotation` property of the targeted movie clip (`whichClip`) equal to the value of the `rate` variable. Remember `rate` starts off as a big number and gradually decreases down to 0.

Adding Scale

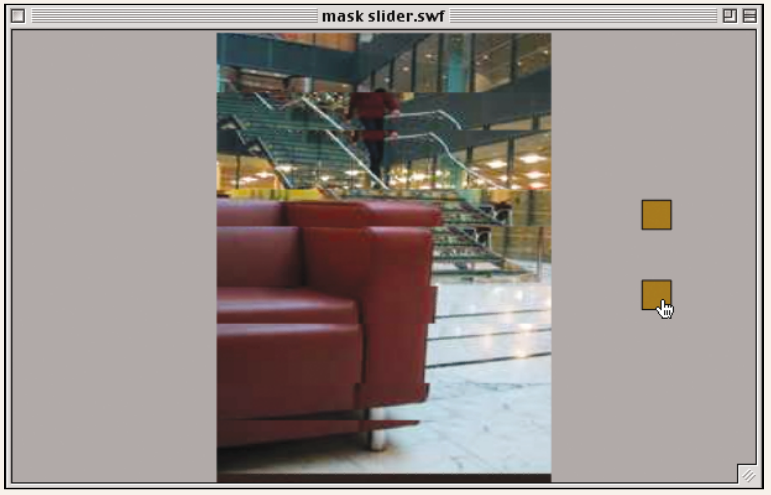
Another effect you can add is that of scale. When the panel is moved, you can make it seem like the picture is zooming in as it reaches its target destination.

1. Underneath the rotation line you've just added to the function, add another line of code:

```
_root[whichClip]._xscale = 100 + (rate *2);
```

   This sets the `_xscale` property of the targeted movie clip to a value of 100 (which is its normal state) plus the value of `rate` multiplied by two. It's been multiplied by two here so that it has a more dramatic effect.

2. Test the movie. You should find it spins and zooms into place when the panel moves. See Figure 3.38.



3.38 The spinning and zooming effects you see are just the first of the special effects you can add.

Play with the script to achieve different effects. Remove the rotation line to see what it looks like with just the scale effect, or add the scale effect to the `yscale` of the movie clip. Play around and experiment.

Conclusion

You've learned how to take a basic ActionScript algorithm and apply it in various ways. First, the slide algorithm was used for a navigation type system. Then you used it to create transition type effects. This shows how you can move clips around without the use of tweening.

You've also been introduced to functions. Functions are incredibly powerful and allow you to streamline your code, and also keep track more easily of where all your main script is located. I usually put all my functions in one place on the main timeline. This way I can easily find something if I come back to it a few months later.

As always, experiment with the code. You don't have to take it literally. You can use the basic code to move any movie clip, not just a panel! Try altering various parameters or adding lines of code to see if you can expand on them. Why not try adding the mask effect at the end of the last chapter, combining it with the slide algorithm?