{ Chapter 13 }

Smart Clips

Smart Clips are a sophisticated and convenient way to encapsulate code snippets in a form that can be shared and reused. A Movie Clip becomes a Smart Clip when you specify the parameters that you want the author to modify in each instance. Effectively, you're just extending the properties by which clip instances can vary from the built-in set of properties (such as `_alpha` and `_xscale`) to include anything you design.

Although you might have seen very advanced examples of Smart Clips (including those found in Flash's Common Libraries menu or that you've downloaded from the Macromedia Flash Exchange site), just because Smart Clips *can* be very advanced, they don't have to be. You can make a simple Smart Clip that serves to automate a small portion of just one project. The most important fact to realize is that everyone can make Smart Clips.

In this chapter, you will

- Learn all the steps involved to turn a Movie Clip into a Smart Clip
- Create adaptable parameters that the author can modify when using your Smart Clip
- Explore some practical uses for Smart Clips including improving productivity, assuring consistency, and centralizing code
- Build Custom User Interfaces that serve to replace the generic Clip Parameters panel with one you build in Flash

Before we begin this chapter, realize that Smart Clips are only used for authoring. After you build a Smart Clip, you can use it as many times as you want. You can even share it with others. It makes sense to do so because you might spend a lot of time making the Smart Clip really useful and adaptable to any situation. Because this means there are two authors, it makes sense to refer differently to the author who builds the original Smart Clip and the "using author" (that is, the person using a finished Smart Clip while building a Flash movie). After you build a Smart Clip, you could become the using author. In this chapter, I will refer to them differently—an author and a using author.

Standard Smart Clips

In Chapter 7, "The Movie Clip Object," you learned how to think of variables contained inside clip instances as homemade properties because you access and change them using the same syntax (`clip.property` or `clip.variable`). This concept will help when creating Smart Clips. The process involves specifying which clip variables (homemade properties) can be initialized through the Clip Parameters panel. Each Smart Clip instance can also have a unique starting value for any of these variables, just as each instance of a clip can have different starting values for any built-in property. Call them parameters, variables, or homemade properties—they're all the same, and with Smart Clips they're adjustable to the using author (see Figure 13.1).

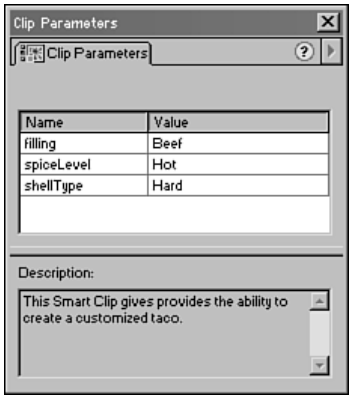

Figure 13.1 *The author can specify variables uniquely for every instance of a Smart Clip by using the Clip Parameters panel.*

The process of *using* a Smart Clip is simple: drag an instance onstage and set the initial values for the variables in the Clip Parameters panel. Then when the movie plays, it's as though each instance has a different `onClipEvent(load)` event to assign the values for each variable uniquely for each instance. What's the point? You could just write an `onClipEvent(load)` script for each instance, and you could avoid Smart Clips altogether. The problem, however, is that it's a lot more work and the process is less intuitive. (For example, you'd have to remember to include assignments for each variable.) Plus, a true Smart Clip lets you take advantage of the Clip Parameters panel and its description field (shown previously in Figure 13.1).

In addition to providing an error-resistant method for the using author to specify parameters, you can include the same base of code (through the master symbol in the Library) in Smart Clips. In this way, you can write one block of code that behaves differently depending on the values of the variables that have been initialized through the Clip Parameters panel. For example, the Smart Clip could contain Dynamic Text fields of a specific font and layout. If the values for the text field variables are specified in the Clip Parameters, each instance will display different text, but the font and layout will remain the same. Plus—just like any Movie Clip—if you make a change to the master in the Library (say that you change the font in the Dynamic Text field), you'll see that change in every instance. In this way, a Smart Clip can serve to establish consistent text styles. Another simple example is a clip with an animation of a ball bouncing. The ball will bounce as many times as the using author specifies for the `bounceCount` variable. They could have several instances of the same Smart Clip, but each would bounce a different number of times.

Let's first look at a couple basic Smart Clips and then we'll look at more advanced practical examples.

Making Smart Clips

Similar to a lot of programming, sometimes it's best to start by hard-wiring a prototype and then come back to clean up things, which makes the code more adaptable. Let's first go through some (non-Smart Clip) solutions to making individual clips behave differently.

Solutions That Don't Use Smart Clips

First, consider a Movie Clip with a 20-frame animation. In the last frame, place a script that reads

```
loopsRemaining--;
if (loopsRemaining) {
  gotoAndPlay (1);
} else {
  gotoAndStop (1);
}
```

(We'll keep this script for a few examples.) The first line of this decrements `loopsRemaining` by `1`. Assuming that the variable `loopsRemaining` is initialized with a value greater than `0`, this script will cause the clip to keep looping until `loopsRemaining` is reduced down to 0. Remember, if `loopsRemaining` is zero the condition is false, so it goes to the else part where `gotoAndStop(1)` executes. You can place two instances of this clip onstage and simply use the following script on each instance (not *in* the clip, but *on* the instance):

```
onClipEvent (load) {
  loopsRemaining=3;
}
```

Just change the value to which you're assigning `loopsRemaining` in each instance, and they'll repeat a different number of times.

So far, we don't have a Smart Clip and we can see it's slightly difficult to go through writing the `onClipEvent` script on each instance. Consider that you might not even need a Smart Clip but the preceding script is too difficult to trust all your using authors to execute—a Smart Clip would be more fool proof. Another (less than ideal) solution would be to first remove the entire `onClipEvent` (previous), name each instance (say `ball_1` and `ball_2`), and then from the first frame in the main movie, use a script such as this:

```
ball_1.loopsRemaining=3;
ball_2.loopsRemaining=5;
```

We still don't have a Smart Clip, and you can see this technique has its faults (namely, you have to name each instance and type the preceding script without error).

Finally, remove the script in the first frame so I can show you one other non-Smart Clip solution. Actually, this solution is not too bad although it's a lot of work. It also lets us explore a clip property that hasn't been mentioned previously (`_name`). In the first frame inside the master movie clip, we can write the following script:

```
loopsRemaining=_name;
```

Translated, this says "set loopsRemaining to the instance name of the clip I'm inside." To use this solution, you'll have to name the clips with names such as `"1"` or `"2"`. Plus, you'll have to change the script in the last frame to "go to" frame 2 (not 1). Otherwise, `loopsRemaining` will keep getting reassigned with the previous script. To get around the first problem, you could use a naming convention such as `"ball_1"` and use a String method such as `loopsRemaining=_name.subStr(5)` to extract just the portion of the name you need. Obviously, this is beginning to be a pain and has definite drawbacks like how we can't really use the first frame inside the clip and how we have to be careful what we name the instance.

Your First Smart Clip

Smart Clips offer the same basic features explored in all the preceding solutions—but we want the using author to specify the value for `loopsRemaining` in a very controlled and easy manner (that is, through the Clip Parameters panel). It's really quite simple to convert this clip into a Smart Clip. A Movie Clip becomes a Smart Clip when you Define Clip Parameters. If you first select our Movie Clip that uses the `loopsRemaining` variable and then choose Define Clip Parameters from the Library's Options menu, you'll be faced with the dialog shown in Figure 13.2.

From the Define Clip Parameters dialog, you can use the plus button to add variables that will be set-able by the using author. For this example, we'd simply press the plus button once and then double-click the `"varName"` that appears in the `Name` column and type **`loopsRemaining`**. Under the `Value` column, we'd double-click to replace `"defaultValue"` with `1` (meaning that if the using author never bothers to access the Clip Parameters panel, `1` will be used by default). Finally, leave the `Type` column in its default setting—meaning that the data type for this variable will be string or number. (We'll look at the other options in a minute.) That's all you need to do, but I want to mention a couple of other options in this dialog before we move on.

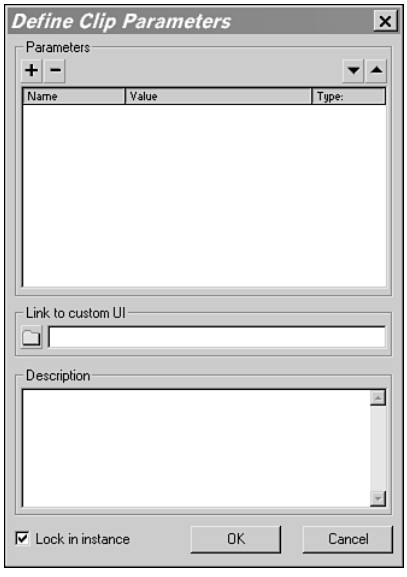

Figure 13.2 *The Library's Define Clip Parameters lets you specify which variables will be set-able in the Smart Clip.*

Normally, we want the using author to only change the values of variables, not variable names. The Lock in Instance option will prevent the using author from changing the name of the variables being edited through the Clip Parameters panel. Although I can't think of a practical reason for letting them change the name of a needed variable, unchecking Lock in Instance will also allow them to *add* variables through the Clip Parameters panel that provides an effective alternative to the `onClipEvent(load)` option I showed earlier.

Another option worth checking out is the Description field. Into this field you can write up a concise explanation of how to use the Smart Clip. It's a good idea to include information as to how to set each variable. For example, you could say something like "Use the `loopsRemaining` variable to specify the number of times you want the animation to loop."

Finally, we'll return to the Link to Custom UI feature later in this chapter when we use a Flash movie to replace the Clip Parameters panel.

By simply adding at least one variable through the Define Clip Parameters dialog, our Movie Clip turns into a Smart Clip evidenced by the new icon in the Library and the fact that the Clip Parameters panel is usable (see Figure 13.3).

If you drag three instances of this Smart Clip onstage, you can then set `loopsRemaining` for each one individually very quickly and easily through the Clips Parameters panel. (Be sure that your edits are accepted by clicking the stage after entering a value in to the Clip Parameters panel; otherwise if you test movie, the last saved value might be used instead.)

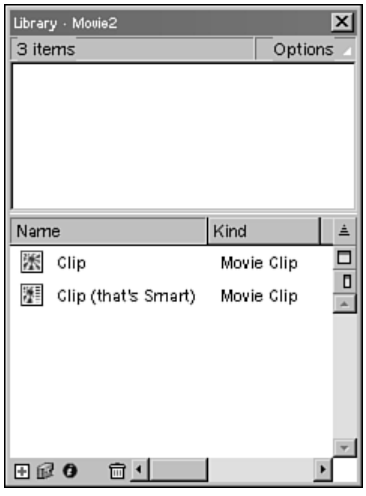

Figure 13.3 *When a Movie Clip turns into a Smart Clip, its icon changes in the Library (as shown in the second one listed).*

A Practical Example

Let's quickly build another simple Smart Clip as a quick review and so that you can see a more practical application: a template. Layout two Dynamic Text fields with placeholder text: one for a title and one for a subtitle. Make sure that the margins are wide enough to accommodate any likely content and pick a nice type face. Make sure that the title is associated with a variable `title` and the subtitle with a variable `subTitle` (Figure 13.4).

Select both blocks of text and convert them a Movie Clip symbol (F8). Confirm that the layout is satisfactory and use the Info panel to notate the x and y coordinates (making sure to use the center point indicated by a black box in the Info panel). Go inside the clip you just created and, in the first keyframe, use the following script to specify the initial location for the clip:

```
_x=150;
_y=229;
```

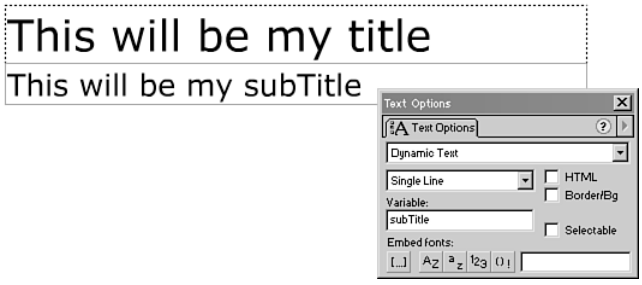

Figure 13.4 *The first Smart Clip we build will include Dynamic Text fields that we associate to variables.*

(But use whatever values you found through the Info panel.) Finally, we can make this a Smart Clip by Defining Clip Parameters. Just add `title` and `subTitle` to the name column for parameters. That's it! Anyone can now drag an instance onstage and define the content through the Clip Parameters panel, and the layout will be perfectly consistent. You can even make a global edit to the layout or font style by editing the master symbol. All the instances in use will retain their values for `title` and `subTitle`. (By the way, you'll need to test movie in order to see anything except your placeholder text in the two fields.)

Other Data Types

Before we move on to some advanced examples of Smart Clips, let's quickly explore the alternatives to the Default data type found when Defining Clip Parameters as shown in Figure 13.2, earlier in the chapter. The other data types are Array, List, and Object.

When you want the using author to populate the clip's variables with strings or numbers, just leave the option set to Default. However, as you know by now, other data types can come in handy. For example, you might want to let the using author set a single variable to an array. When you Define Clip Parameters, you simply specify that a particular variable shall be an Array. At that point, you can populate the array with default values by double-clicking on the field in the Value column. What's interesting about the Array type is that when the using author is populating the variable with values, he not only can change the default values you provided, but also he can add items to the array. None of the other data types (Default, Object, and List) let the using author add. A great example in which Array is a good choice is the Menu Smart Clip that ships with Flash (found under the menu Window, Common Libraries, Smart Clips). The using author is

allowed to add as many items to the menu (which is to say, he's allowed to add items to the array).

When you want to give the using author a choice of several discrete options, select List. That is, Default lets him type anything he wants; Array not only lets him type anything, it lets him add items; but List only lets him select from a predefined list. For example, if you want to provide a choice of music genres but don't want the using author to type in just anything, you can instead offer a list of just Classical, Country, Rock, Jazz. In the Value column, you simply fill in what the options should be, and the using author can only select from a drop-down list. If you want something other than the first item—the zero spot—to be selected by default, provide an "offset" (see Figure 13.5).

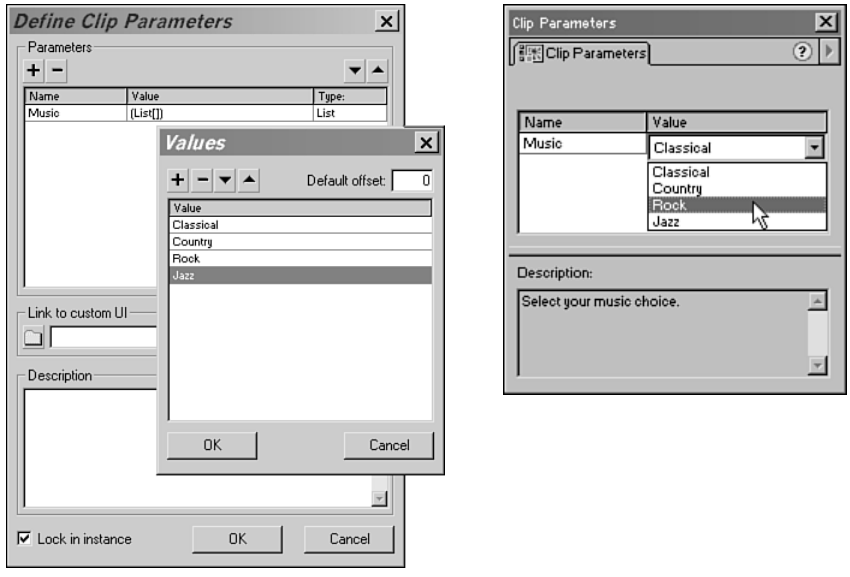

Figure 13.5 *Creating a "list" variable when defining Clip Parameters (left) gives the using author a limited choice (in the drop-down list on the right) when using the Clip Parameters panel.*

Finally, the Object type is convenient when you know that the variable used in your clip needs to be in the form of a generic object. Remember, generic objects are unique and hard-wired. We built generic objects to be passed as parameters in the `setTransform()` methods for the Sound and Color Objects in Chapter 11, "Objects," and they're the same as Associative Arrays (discussed in Chapter 10, "Arrays"). It's simply a data type that has multiple named properties (accessible

either by `variableName["propertyName"]` or `variableName.propertyName`). Unlike the Array type, the Object type requires that you define the properties so that the using author simply defines the values for each property. In fact, you could just provide the using author several separate variables (all of the Default type), and then once inside the clip (say, in the first keyframe) write a script that builds an object by assigning properties based on the values provided. Selecting Object simply prevents the need to translate individual variables and instead creates a single variable with multiple properties (that is, an object).

As a quick review, we've seen that a Smart Clip is simply a Movie Clip for which the Define Clip Parameters option was used to specify clip variables that the using author will be able to manipulate. One would suspect that you'll be *using* those variables somehow inside the clip—such as within a Dynamic Text Field or in a script that utilizes the value of the variable. Each instance is unique in all the ways the different clip instances can be unique; but, in addition, the using author can change the value of all variables listed in the Clip Parameters panel. In this way, the using author makes each instance of a Smart Clip behave differently because each instance will start off with different values for all its variables.

Advanced Applications for Standard Smart Clips

Although the earlier example of using a Smart Clip like a template to impose a consistent font and layout for text was indeed practical—it was quite simple. Other practical examples aren't quite as simple. I'm going to call the following examples "standard" Smart Clips (not simple) because they don't involve the more advanced feature called custom UIs. A *custom UI* (which stands for User Interface) replaces the Clip Parameters dialog (and its rigid looking name and value columns) with a Flash movie that you have to build. It's the job of this other Flash movie (the UI) to set the necessary variables, but it can do so in a very graphic way. We'll make some in the next section, but I want to first show some advanced examples of the standard form. You don't need to try to follow along as we explore some of the possibilities.

In the Horizontal Slider workshop you'll build a slider that lets the user (not just the using author) interact by dragging it from 1 to 100 (as shown in Figure 13.6). After we build a somewhat hard-wired version, we turn it into a Smart Clip to allow the using author to specify four properties: the location of 1 (that's the

minimum location for the slider), the location of 100, the initial location (so that the slider can default to a point other than 1), and the name for a function that the slider will continually call as the user slides the slider. Using a function makes it possible for the slider's value to be used to modify anything onstage—from another clip's alpha level to the overall sound level. This kind of function that a Smart Clip calls (back in the main movie) can be referred to as a *call back function*.

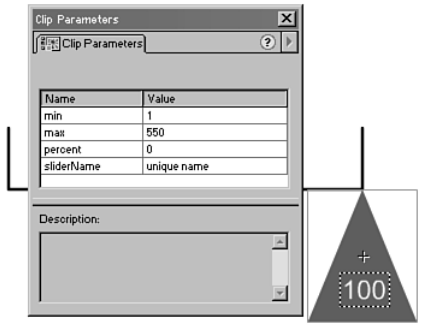

Figure 13.6 *In a workshop later, we'll turn a slider into a Smart Clip so that it can be reused.*

In the Tool Tip workshop, we build a Smart Clip where the using author can specify the exact string of text that should appear (when the cursor rolls over another object). The using author can drag as many instances of this Tool Tip Smart Clip as he wants.

Finally, here's an example of a Smart Clip I built for use in a real project. I used Flash in a presentation, but wanted another version (that the audience could download) that included speaker notes. The content for the notes would be loaded in from an external file (as you'll learn in Chapter 14, "Interfacing with External Data") because it wouldn't be written until later—plus I wanted to be able to modify it any time. Anyway, I made a Smart Clip not unlike the tool tip described previously, but instead of making the using author (me) specify *all* the text, I simply specified the section and subsection where it was being used. The loaded data included details as to which section and subsection it applied to, so my Smart Clip simply displayed (similar to a ToolTip) the data appropriate for that section. The result was that users can view the presentation and optionally click a Speaker Notes button to see additional information (see Figure 13.7).

Figure 13.7 *In an actual project, I made a Smart Clip to display the appropriate speaker notes.*

Although these examples are not the only things possible with standard Smart Clips, I just wanted to make a point that you often don't *need* to build custom UIs.

Replacing the Clip Parameters Panel with Custom UIs

One of the most intriguing features of Smart Clips is the fact that you can assign an interactive Flash movie to play inside, and effectively replace, the Clip Parameters panel. The process involves first building a Flash movie, exporting it as a `.swf`, and finally pointing to the `.swf` through the Define Clip Parameters dialog. You'll have two files: an `.fla` file with the master Smart Clip in a Library and a `.swf` that plays inside the Clip Parameters panel. The `.swf` is called a custom UI and its filename is specified in the Link to Custom UI field shown in Figure 13.8.

By replacing the Clip Parameters panel, we have the opportunity to make something more usable. But most Smart Clips are perfectly suitable without a custom UI. So instead of leaping straight into building custom UIs, we'll first look at how to design them so that we're sure to use them appropriately. You'll see that custom UIs can be difficult to build—so it makes sense to make sure that they're necessary.

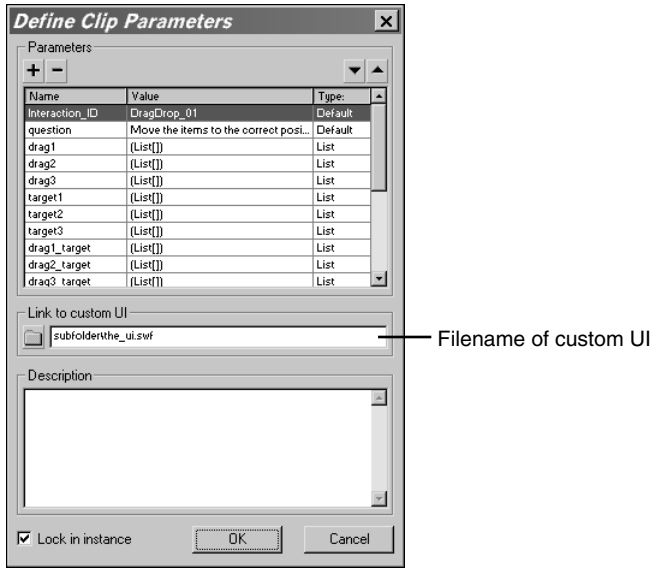

Figure 13.8 *The Link to Custom UI field points to an external file that will be used in place of the Clip Parameters.*

Designing Custom UIs

Naturally, the process of creating a custom UI is purely technical. To make a *good* custom UI is another matter. I think it's fair to say that the only time to create a custom UI is when the built-in Clip Parameters panel is inadequate. There are many situations in which this could occur. For example, making the using author set several variables through the standard Clip Parameters panel could be unreasonably tedious. In this case, an easier solution might be a graphic selection device such as a slider (see Figure 13.9). Or maybe you want to give the using author a taste of the selections he's making. If he's picking several colors, it might be nice to give him a preview so that he can visualize the results. Or, if he's selecting sounds, you could include a short audio sample. Finally, a perfect situation for a custom UI is when there is a series of complex selections the using author must make. For this case you could build a custom UI that served as a wizard—walking the using author through all the steps involved and even providing online help where appropriate. A good example of this approach can be found in the Smart Clips included with Flash's Learning Interactions (under Window, Common Libraries).

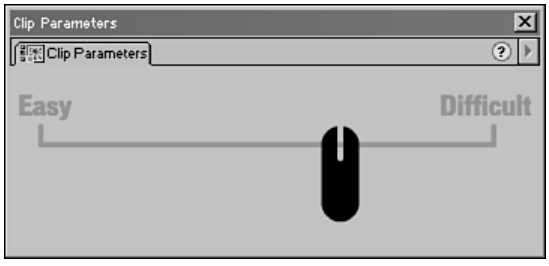

Figure 13.9 *A slider is just one graphic interface element possible to include in a custom UI.*

After you've determined that a custom UI is appropriate, you can take steps to design how it will function. The most important consideration is usability. Because the purpose of the custom UI is to provide some benefit not found in the standard Clip Parameters panel, you should make sure to make it easy for the using author. I suppose if you're building a Smart Clip for your own use, you can invest less time designing (at the expense of usability).

As it turns out, one of the most critical features that will make your custom UI more usable happens to be one of the most difficult to program. It's important for the custom UI to always indicate the current settings. For example, every button should include a highlight to indicate selected. This highlight should not only provide a clear indication at the time a selection is made, but also the using author should be able to return to the Clip Parameters and easily ascertain the current setting. After all, they might have several instances of the Smart Clip and want to check each one's settings. This round-trip feature (being able to leave and come back to a Smart Clip) is the difficult programming task. You'll see how to program it in the next section, but realize that it's also a matter of design how you choose to treat the graphic solution. Ultimately, making an intuitive custom UI takes more skill and creativity than simply programming it.

Building Custom UIs

Assuming that you've determined a custom UI is really necessary and you have a decent design, you can move on to really building it! First let's make a very simple one and then add some features. The concentration of this example is on making the custom UI, but we'll need a Smart Clip for whom the custom UI sets properties. You can use the simple bouncing ball Movie Clip used earlier. You'll want to create a file (maybe called `host.fla`), make sure it's saved, and then place this script in the last frame of the Movie Clip:

```
loopsRemaining--;
if (loopsRemaining) {
  gotoAndPlay (1);
} else {
  gotoAndStop (1);
}
```

The job of our custom UI will be to set the `loopsRemaining` variable. (Arguably, this Smart Clip doesn't really *need* a custom UI, but we're just doing it for practice.) Make sure that your Movie Clip is a Smart Clip by using Define Clip Parameters to specify that `loopsRemaining` is set-able by the using author. Finally, type **`myUI.swf`** in the Link to Custom UI field. You can also click the folder button and point to a file. However, besides the fact that we haven't made the UI yet, this feature always produces an explicit path where—for most situations—the relative path we typed in is more desirable. (By the way, in a workgroup situation, you could keep the custom UI in an explicit path on a server for everyone to share.)

Create a new file and save it as **`myUI.fla`** in the same folder as the host file. Finally, we can program it. Because we're basically replacing the standard Clip Parameters panel, we need to do what it was doing: setting variables. The only catch is that the variables that get exchanged with the host movie's Smart Clip need to reside in a movie clip that has an instance name of `xch`. You can have other variables in the UI file, but only the ones in the clip `xch` will become part of the Smart Clip in the main movie. This clip instance doesn't need anything graphic; it serves only to hold the variables that get used in the main movie. Although you can think of the `xch` clip as a surrogate of the actual Smart Clip, it doesn't need to have any correlation (in looks or function) to the Smart Clip—only that it contains the necessary variables.

We can make the fastest custom UI in history by creating an Input Text field associated with the variable `loopsRemaining` and then selecting the text block and converting to the Movie Clip symbol. Finally, just make the movie clip's instance name `xch` and export the movie as `myUI.swf` in the correct folder. Go back to the host movie and test it out by dragging two instances of the Smart Clip. Through the Clip Parameters panel you should see the Input Text field where you can specify a number of loops. Also, notice that you can keep the Clip Parameters panel open when you alternatively select the two Smart Clip instances onstage. Each should retain its `loopsRemaining` value. Pretty easy really.

Even if we try to spice it up with gratuitous effects (such as maybe text color), this custom UI is pretty simple. Let's change it so that we can encounter something more challenging. You can keep the `xch` clip—but change the text field to Dynamic Text (so that the user can't edit it). Make a button and create four instances in the main timeline lined up vertically. Place the following script in each button (changing the `1` to `2`, `3`, `4` for each button, respectively):

```
on (release) {
  pickLoop(1);
}
```

Now in the first keyframe of the main timeline, type this function:

```
function pickLoop(whatNum){
  _root.xch.loopsRemaining=whatNum
}
```

This achieves the task of changing `loopsRemaining` to whatever value is passed from the buttons that call `pickLoop()`. You can export the `.swf`, and it should work. However, when you test this from the host movie (the place where you should be testing this), there are two significant problems. Upon making a selection, the user is not given any graphic feedback as to which button was pressed. The other problem is when a user returns to view the current setting in a Smart Clip—he has no clue what the value is for `loopsRemaining`. We can produce a highlight on the currently selected button by creating another movie clip in the main timeline and calling the instance `arrow`. Then we just need a script in the `pickLoop` function to change the `_y` property of `arrow`. If the spacing is consistent, you could use a formula such as

```
arrow._y=46+75*(whatNum-1);
```

where 46 was the location for the top button and each button was 75 pixels apart. Or—in conjunction with an array full of discrete y locations—you could use an expression such as

```
var locs=[46,121,196,271];
arrow._y=locs[whatNum-1];
```

Each solution moves a clip instance (`arrow`) to point to the last button clicked—and the time to do this is inside the `pickLoop` function.

At this point, the custom UI should function as far as indicating a selection *after* you make it, but it still fails to appear with `arrow` in place upon returning to a previously edited Smart Clip instance. All we need to do is place the following script in an appropriate keyframe so that it executes every time a using author returns to edit the Clip Parameters:

```
pickLoop(_root.xch.loopsRemaining);
```

Basically, this script sends the current value of `loopsRemaining` (which is in the `xch` clip) to the `pickLoop` function. The problem with our custom UI is not that `loopsRemaining` is being lost (if you test it, you'll find that it is still there); but rather that the value of `loopsRemaining` is unknown when returning to the clip. However, if you place the preceding script in the first frame, it won't work! The issue is that Flash needs some time to send the variables from a Smart Clip instance to the Clip Parameters panel and then to your custom UI. The solution is to move everything in your custom UI out past frame 10 or so, and then invoke a function call (similar to the previous). One little catch is that the `xch` clip must be present in the first frame! The way I remember this rule is to imagine the Clip Parameters panel is attempting to set the variables that are part of the `xch` clip. Just like how any script can only set variables for clips that are currently present, the `xch` clip must be present at the very start so that the Clip Parameters panel can set the variables. I'm not sure if that's really the reason, but I simply place the `xch` clip in its own layer and make sure that it starts on frame 1. Actually, this is the best reason to consider *not* putting any graphics or buttons in the `xch` clip itself as you don't want the user interacting until everything is reinitialized.

The arrangement I'd recommend is as follows (and this will work for the previous exercise):

1. Place your dummy `xch` clip in frame 1 on its own layer.
2. In frame 1 of your `xch` clip's layer in the main timeline, place all the functions (such as the `pickLoop` function).
3. Make a new layer just for Actions, and in frame 10 make a keyframe that contains the reinitializing script (`pickLoop(_root.xch.loopsRemaining)`).
4. On frame 11, place a `stop()` script.
5. Place all your interactive elements (buttons for example) on frame 11.
6. Use the first 10 frames to display a "loading" message or an animation that serves to placate the using author who must wait for the reinitializing to occur.

Notice that you want to make sure that the user cannot interact while the custom UI is initializing, and that's why no buttons appear until frame 11. (Your loading message occurs before that.)

Although this example is admittedly simple, a more complex custom UI will have the same elements. You always need a clip named `xch` that holds the variables to be set in your Clip Parameters panel. And, unless your only means for the user to see his current settings is a text field, you'll need to initialize such highlights somewhere other than the first frame (I think frame 10 is a safe bet). After you understand these minimum features, you can move on to making more complex Smart Clips and custom UIs.

Keep in mind that the goal for any good Smart Clip (and custom UI, if needed) is to be something useful: for instance, a code snippet that can be used over and over. Although something that's particularly useful is worth investing time and effort to make it right, every Smart Clip is not necessarily hard to create. In practice, I've found that the majority of Smart Clips are built on a per-project basis. That is, you might need a special Smart Clip that's used throughout one Flash movie. As you saw early in this chapter, Smart Clips can be used as simple templates or style guides. A large number of the workshops later in this book involve creating Smart Clips because I want the code to be more usable. It turns out the Smart Clips we build won't have universal appeal: only that making it a Smart Clip means that the code is just a bit more adaptable.

If you do build a Smart Clip with a wide general appeal, you can share it with the Flash community. In Appendix B, "Making Flash Extensions for the Macromedia Exchange Web Site," you'll see how easy it is to turn a Smart Clip into a Flash Extension that can be downloaded from the Macromedia Exchange Web site (`http://www.macromedia.com/exchange/flash/`).

Summary

This chapter turned out to be more like a workshop and less like many of the earlier foundation chapters. That's because Smart Clips are more a feature of Flash than a language element of ActionScript. The only new concept was the "list" data type—which isn't really a data type but just another feature of the Define Clip Parameters dialog.

The concepts you learned in this chapter included how Smart Clips allow you to give the using author the ability to set initial values for any property, variable, or homemade property you specify. When you Define Clip Parameters, the Movie Clip magically turns into a Smart Clip. From that point forward, each instance onstage maintains its own values for the designated properties. The using author can change individual instance properties through the Clip Parameters panel (not to be confused with the Define Clip Parameters dialog in which you establish the properties that are set-able). Finally, if you go through the work to build a custom UI, you can use this Flash movie to effectively replace the Clip Parameters panel.

If you're left with any confusion as to the value of Smart Clips, don't worry. In the workshop portion of this book, you'll make plenty of them. It's not so much that you sit down and decide, "Today I'm going to make a Smart Clip." Rather, you can build some code and then say, "Hey, this would be way better as a Smart Clip because it would be adaptable for multiple instances."