

Smart Clips: Re-Usable Elements for User Interfaces, Applications, and More

Phillip Kerman

Annotated presentation, downloads, and sample chapters from my two books (*Sams Teach Yourself Flash 5 in 24 Hours* and *ActionScripting in Flash*) available at: www.phillipkerman.com/ucon2001/

__Clips that are smart and Smart Clips

Every instance of a symbol on stage maintains its own properties. For example, you can set the position, scale, rotation, and effects (like tint and alpha) differently for each instance you drag onto the stage. Using ActionScript you can refer to properties of individual instances during runtime using the form: object-dot-property--such as `clip1._rotation` where "clip1" is the instance name of a particular clip. To change the `_rotation` property of `clip1` you can simply assign it a new value as in `clip1._rotation=90`.

Similar to how clip instances maintain their own set of properties, custom variables become part of individual instances. You can refer to and change such variables using the *same* object-dot-property form. That is, `clip1.age=35` assigns the `age` variable in `clip1` to `35`. It's often easiest to simply think of variables in clips as a "homemade properties". Call them "variables" or "properties" (or even "parameters" if you want). The fact is, both built-in properties and custom variables are independent for each instance and use the same syntax. The fact that many custom variables have no visual representation onscreen is immaterial--you can always monitor variables with the Debugger or use the variables in other parts of your code. For example, when tracking the user's score you may not want to let them see the current value.

One difference between built-in properties and variables in clips is that built-in properties are set during authoring. Variables in clips are set during runtime with ActionScript. Setting variables is not difficult, but if you want the variables in each instance of the same symbol to start with different values you must set the variables from outside the clip. That is, initializing clip variables individually is achieved with a frame script *outside* the instance or in a **load** clip event script attached individually to each clip instead. (Actually, it's possible to write a dynamic expression and place it inside the master symbol--perhaps in its first keyframe--that initializes variables uniquely based on a clip property such as `_name` but you'll see Smart Clips make this unnecessary.) The practical problem initializing variables outside the master symbol is that you either need to name each instance (so you can refer to the variable contained) or you need to place a separate (**load** clip event) on every instance. For every new instance you need to remember to either name it and set its variables or to attach a script to the individual instance. Therefore, your code becomes spread out and repeated in many places.

Smart Clips solve this issue by allowing you (the author) the ability to specify which variables (call them properties or parameters) can be set via the Clip Parameters panel by the author using the Smart Clip. In this way, you now have homemade properties that are changed via a panel.

A Smart Clip is born

Converting a Movie Clip into a Smart Clip is easy. However, you should first determine which variables the using author needs access to. Then you simply use the Define Clip Parameters dialog to specify which variables will be accessible to the using author. That's it. Smart Clips can be dragged onstage and, through the Clip Parameters panel, have their respective variables changed. Although the Clip Parameters panel and Define Clip Parameters dialog sound like the same thing, they're not. You specify variables through the "Define Clip Parameters" dialog (in the Library) and the using author populates individual instances with the "Clip Parameters panel".

Examples of Simple Smart Clips

Style guide or template. Inside a Movie Clip, lay out a Dynamic Text field associated with the variable **title**. Use the Define Clip Parameters to allow the using author to specify the value for **title** (that is, the text that will appear). This Smart Clip can be used throughout a movie. The best part is that you can change the style or formatting of the text, and--since it's stored only once as a symbol--you'll see the changes reflected everywhere. Finally, a script placed in the first keyframe of the Smart Clip such as **_x=320; _y=240;** will make every instance appear in the same location regardless of where the sloppy author placed it.

Arrow button. Instead of having a separate button for "next" and another for "previous" (each with a slightly different version of the same basic code) place the button inside a Movie Clip. Turn it into a Smart Clip by allowing the using author to specify **direction** (either **1** or **-1**). Then the script you place on the button should use the value of **direction** to decide what script to execute. For example:

```
on (release) {  
    _root.gotoAndStop(_root._currentFrame+direction);  
}
```

Speaker Notes. The online version of this presentation loads variables containing about a paragraph of text for each slide. The user can rollover a button to reveal the appropriate text. I made a Smart Clip that let me (the using author) specify which variable would be associated with each instance I dragged onstage. I just dragged the Smart Clip into content areas (of my Flash movie) for which I had written speaker notes.

__Details of Define Clip Parameters

For every variable that you specify through the Define Clip Parameters dialog you need to provide a default value (in case the using author never fills it in). In addition, you can select from four "types" which are comparable to data types. Basically, the type you choose affects how the using author will populate the variables. Here are the four types:

Default is for individual numbers or strings. Using authors can basically set values to anything they want.

List lets the author select from a drop-down list. This way you limit the values from which the using author may select. In the end, your variables still contain just one string or one number value.

Array is quite different than **List**. Not only can the using author add and remove items (where in the **List** type is a predefined list) but in the end, the variable becomes an array data type with individual values in each index.

Object lets the using author set values for multiple properties of a single variable. The generic object data type (also known as "associative arrays") is like regular arrays with their multiple values, however objects have a name for each value. It's really like having a set of name/values within a single variable. This option is best when you need or want simple objects in your clip. It takes more effort to populate this type of variable but you can restrict the using author to a predefined list of properties like the **List** type.

You can allow the using author to rename, add, or remove the variables you defined by un-checking "Lock in Instance". You can also provide a description that serves to explain how to use the Smart Clips.

Realize that the using author sets variables that are part of the main Smart Clip. You can use as many variables both contained in the Smart Clip and in nested clips--but through the Clip Parameters panel the using author simply sets variables of the Smart Clip. Only make the using author populate the variables necessary to make the Smart Clip behave uniquely.

__Custom User Interfaces (UIs)

You can make a separate .swf that plays inside the Clip Parameters panel (effectively replacing the name/value table). You should first justify why you'd want to do this. To simply make something more entertaining has questionable merit. There are two times that I see a need to create a Custom UI. First, if a better interface device can remove a tedious process of populating data--use it. For example, selecting colors by clicking a swatch or color selector is much easier than, say, typing the HEX values. The second reason to consider a Custom UI is when the using author needs to be guided through the populating process. Not so much to keep them from breaking things, but sometimes one variable's value will eliminate the need to gather another. In those cases it's nice to make a Custom UI that walks the user through the various settings like a "wizard". For example, the learning interactions Smart Clips that ship with Flash use tabs for this purpose.

Enough of the reasons *not* to make a Custom UI--here's how it's done. Identify the variables the using author will be populating. These must be stored in a clip in the main timeline (of the source UI .fla) with the instance name **xch**. You can think of the **xch** instance as a surrogate for the actual Smart Clip--but just to hold the necessary variables. I find it easiest to just use an invisible clip instance for this purpose. This instance needs to be present in the very first frame of the ui.fla--put the **xch** clip in its own layer for convenience. Then you just build your UI movie to allow a user to set the key variables inside **_root.xch**. Building the script that lets the user interact is easy enough... even displaying highlights as they make selections is straightforward. However, realize that unlike most movies that always initialize all variables the same way, the custom UI should visually reflect the values of variables set previously. That is, the using author may pull up the Clip Parameters panel to inspect clips they've already populated. If you keep the **xch** clip in the first frame, the variables are initialized properly. However, you need to build some "restoration" script that re-displays highlights to represent the current values of the variables. The killer is that such a restoration script must be placed *after* the first frame... at frame 10 to be safe. Think of it this way: First, the using author clicks on a Smart Clip instance in the host movie. Then the Clip Parameter panel determines the variables of that instance and launches the Custom UI movie. A second later the Clip Parameters panel reinitializes variables in the Custom UI .swf's **xch** clip. Then... the Custom UI is free to use the new values of variables contained in its **xch** clip. Whether or not that makes sense (and regardless of the fact you may not like it) you must wait to restore visual highlights. By the way, Dynamic or Input text fields contained in an **xch** clips will automatically restore--but if you're just using text why even bother with a custom UI?

The Custom cursor example demonstrated is in my new book *ActionScripting in Flash*.

__Additional Information

--Tip 1: Buttons can't be placed on top of buttons. While it would be very convenient to use the **on (rollOver)** event that a button offers, if you plan to use your Smart Clip on top of buttons inside a movie you'll find the two buttons conflict. For example, if you make a custom cursor Smart Clip that lets the using author place it on top of any other button, you cannot use an invisible button inside the Smart Clip (to figure out when it's time to show the cursor). Instead use either the **getBounds()** or **hitTest()** methods.

Consider this code snippet:

```
onClipEvent (load) {
    rect=this.getBounds(_parent)
    _visible=0;
}
onClipEvent (mouseMove){
    if(_xmouse>rect.xMin&&_xmouse<rect.xMax&&_ymouse>rect.yMin&&_ymouse<rect.yMax){
        active=true;
    } else {
        active=false;
    }
}
```

Using `hitTest()` may look cleaner, but notice you can't make the clip itself invisible:

```
onClipEvent (load) {
    // _visible = 0;
    _alpha = 0;
}
onClipEvent (mouseMove) {
    if (hitTest(_root._xmouse, _root._ymouse, 1)) {
        active = true;
    } else {
        active = false;
    }
}
```

--Tip 2: If you plan to use `attachMovie()` to dynamically create clip instances inside your Smart Clip, be sure to include a copy of each symbol (being attached this way) in a guide layer. That way, when you copy the Smart Clip to another file, all the associated symbols are copied with it. Naturally, the guide layer isn't needed during runtime--it just assures all the symbols get copied.

--Tip 3: You can override built-in properties by including them when you "Define Clip Parameters". For example, specify that `_name` will be set through the Clip Parameters panel. If you provide a default this makes it possible to automatically name every instance that's dragged on stage. By the way, this *will* override any settings you make through other panels such as the Instance panel.

--Tip 4: When you add parameters (through Define Clip Parameters), instances on stage may need to be replaced to reflect the new options.

--Tip 5: Similar to how you must remember to click off the Instance panel after renaming an instance, you'll want to make sure changes you make (as a using author) in the Clip Parameters panel are saved by clicking the stage after each edit. (Don't immediately, Test Movie... don't click tab.)

__Bonus topic: The Cookie Object

Here's how you set preferences for a Custom UI through the undocumented (and unsupported) Cookie Object. Any .swf that is played through the authoring player (that is, through Test Movie or as a Custom UI) can read and write XML structured text files in the "Mmfdata" folder (adjacent to your installed version of Flash). When you write a file, the filename is generated automatically so that it's associated with the .swf that created it. Therefore, when you decide to load data from file you don't need to specify the filename--you automatically read from the file this .swf created last time. If you snoop around the "Mmfdata" folder you'll see that the Dashboard takes advantage of the Cookie Object in order to store information such the last date you ran the Update feature and your preference for auto-update.

Here's the syntax:

```
Cookie.setCookie(structuredString)
```

Where "structuredString" is your XML structured data string that you want saved.

```
returnedXMLObject=Cookie.getCookie()
```

Where "returnedXMLObject" is the value returned. The **getCookie()** method can be used just like the XML Object's **load()** method (but without the URL parameter). Similarly, you'll want to wait for the data to fully load. Therefore, be sure to use the **onLoad()** method like this:

```
xmlObj = new XML();  
xmlObj=Cookie.getCookie();  
xmlObj.onLoad=triggerFunction;
```

Where "xmlObj" is the variable that will contain the XML object and "triggerFunction" is the name of the function you want called when the data has fully loaded (perhaps a parsing routine).

Here's a contrived example that shows how to use the Cookie Object:

Consider a Smart Clip that provides the using author a choice of several colors. Each color is associated with a frame number (that's colored accordingly) in a clip **highlight**. The variable **colorNum** is contained in the **xch** instance so that it becomes part of the Smart Clip.

In frame 10 of the source Custom UI file, the following script either restores the last **colorNum** value (by jumping to the correct frame in the **highlight** clip), or (if no value has been set and the value is **undefined**) executes the **getCookie()** method to read in the most recent value of **colorNum** (selected by anyone using this Smart Clip).

```
message="";  
stop();  
if(xch.colorNum<>undefined){  
    highlight.gotoAndStop(xch.colorNum);  
}else{  
    xmlObj = new XML();  
    xmlObj=Cookie.getCookie();  
    xmlObj.onLoad=parse;  
}
```

Notice that if `xch.colorNum` is undefined that when the `getCookie()` method is finished loading into the `xmlObj` variable (that is `onLoad`), the `parse()` function (below) is called. The necessary portion of the whole `xmlObj` variable is extracted and placed in a variable `frameNum`, then the highlight clip jumps to that frame, `xch.colorNum` is assigned (in case the using author leaves without making a selection), and finally, the value for a variable `message` (contained in a Dynamic Text field) is assigned.

```
function parse(){
    var frameNum=Number(xmlObj.firstChild.childNodes[0].nodeValue);
    highlight.gotoAndStop(frameNum);
    xch.colorNum=frameNum;
    message="most recent";
}
```

Finally, this is the function that is called every time the using author makes a selection. That is, `setOne(whichWay)` is called where "whichWay" is the value of the `colorNum` they are selecting (also the frame number in the `highlight` clip to where they'll jump). After the `xch.colorNum` is assigned, they jump to the correct frame in the highlight instance. Then, a very simple XML structured string is built and the `setCookie()` method writes the information to file where it can be read in next time.

```
function setOne(whichWay){
    xch.colorNum=whichWay;
    highlight.gotoAndStop(whichWay);
    form="<COLORNUM>"+xch.colorNum+"</COLORNUM>"
    Cookie.setCookie(form);
}
```