

# HOUR 19



## Designing Your Web Site to Be Modular

It's possible to create a huge Web site entirely with one giant Flash file. However, separating the site into modular segments has distinct advantages. Just to name a few, you can load portions of the site as needed (instead of making every visitor download everything), several team members can be working on the same site simultaneously, you can update portions of the site as they change instead of having to reedit one master file, and you can create different versions of the site for different languages by just swapping out portions with language-specific content. There are other reasons why modularity is good, but it comes down to efficiency, your productivity, and the user experience.

This hour covers several ways a Flash site can be modularized as well as some of the issues you'll need to consider in deciding when and where to modularize.

In this hour you will

- Learn how the loadMovie Action lets you play one movie inside another or display an external .jpg image
- Learn how the loadSound Action lets you play external .mp3 sounds
- Learn the benefits of and how to use Shared Libraries
- See how scripts can be stored outside the Flash file using the Action #include

Although the technical issues covered this hour are not particularly difficult, the Flash features discussed are strict and unforgiving. After you get the features covered this hour to work, it's fine. The difficulty comes in deciding the appropriate use of such features. That is, it's easy to learn *how* these features work, but it's more difficult to decide *when* to use them and *where* to use them. For each feature, you'll first learn how it works and then look at practical uses.

## Loading Movies or JPGs

loadMovie is an Action that lets one Flash movie play another. Effectively, the second movie plays on top of the first. It's easier to understand, though, if you think of one movie as the *host* and the other movie (the one that's loaded on top of the host) as the *submovie*. Think of a big entertainment system—a wall of stereo and TV equipment. The movie you put into your VCR can only play on the TV screen. If you think of the TV as the host Flash movie, then video you put into your VCR is loaded on top of it.

One reason to do this is because you may have several submovies that only play one at a time. You may want to give the user the choice as to which submovie to play. If you use loadMovie, when the user clicks a button an Action tells Flash to load this movie now. It's sort of like a jukebox, where each record is a separate submovie. The reason loadMovie is beneficial is that only the submovies the user requests have to download.

It's time to look at some of the technical issues with loadMovie; then you can try it out. First, only .swfs can execute the loadMovieAction. Therefore, not only will you have to “Test Movie” to see the results, but the movie (or movies) that are loaded must have already been exported as .swfs. Second, when you use loadMovie, one of the parameters is Location (sometimes called Object), where you specify into which location you want to load the movie. Movies are loaded into one of two basic locations: a clip instance or a level number. If you load a movie into a clip, the clip is replaced with the movie that's loaded. If you load a movie into a level, anything that happens to be in that level currently is replaced by the movie.

**NEW TERM**

*Levels* are the numbering system Flash uses to describe the stacking order of loaded movies. Your host movie is always in level 0 (referred to as `_level0`). If you load one movie into `_level1` and another into `_level2`, the `_level2` movie will appear on top of everything.

Finally, remember when I said that when you load a movie, it gets loaded “on top”? That’s not entirely true. When you load a movie into a clip, the loaded movie is in the same level as the clip. That is, if the clip was in front of something or behind something, the loaded movie will be, too. In the case of loading a movie into a level, the loaded movie will be in front of everything else that happens to be assigned lower-level numbers and behind items assigned higher-level numbers. The `_root` Stage is always `_level0`. Therefore, if you load a movie into `_level1`, it will be on top. However, if you load another movie into `_level2`, it will be on top of everything—the first loaded movie (`_level1`) will be sandwiched between the Stage and the new loaded movie.

## Task: Use Load Movie

In this task you use the Load Movie Action to selectively let the user download just the segments he or she is interested in. Here are the steps to follow:

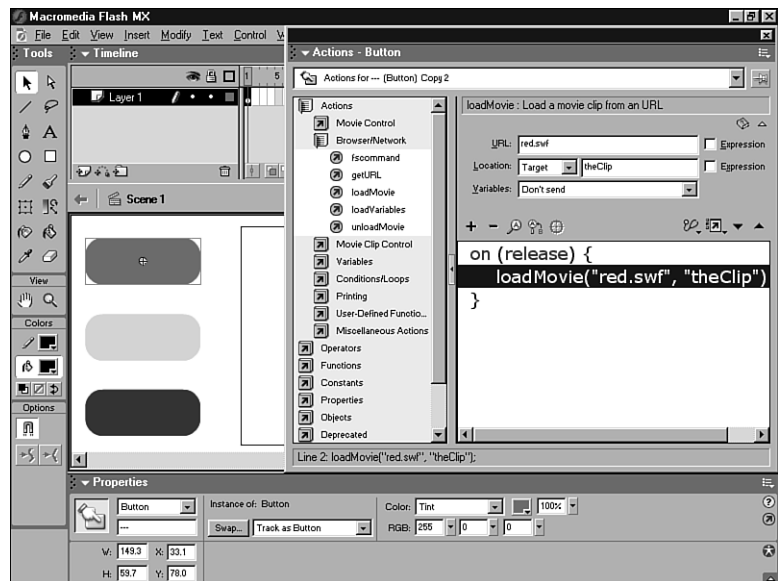
1. The name of the game for this task is organization. Remember, haste makes waste. Start with a new empty folder into which you’ll save and export all your movies.
2. Create a new file and set the movie’s width and height both to 300. Use Modify, Document (or Ctrl+J) to do this and make sure that Ruler Units is set to Pixels.
3. Create a simple tween of your choice, but make the tweening object entirely red. Save the movie as `red.fla` in your folder created in step 1. Do a Test Movie, which will export a movie called `red.swf` in the same folder as your `red.fla` file.
4. Do a Save As, name the file `green.fla`, and change the color of the tweening object to green. (You may need to change the color in each keyframe.) Remember to save and then Test Movie to create the `.swf`.
5. Repeat step 4 but create a file with everything blue.
6. You should have three `.flas` and three `.swfs` (red, green, and blue for both). Close all the Flash files. Then create a new Flash file and save it as `main.fla` in the same folder. Set this movie’s size to 500×500.
7. This “main” file will load movies into a clip. Draw a square exactly 300×300 (draw any rectangle and then use the Info panel to change its dimensions to 300×300). Make sure there’s a line around the box and then delete the fill. Select the entire box and convert to it a symbol (make it a Movie Clip and name it

“box”). Name the instance on the Stage “theClip” in the Properties panel while the box is selected.

8. Now draw a rectangle in the main scene (to become a button). Convert it to a symbol (button) and then access the Actions panel for this button instance. Insert a loadMovie Action (from the Actions section of the Toolbox list). Notice that, by default, your script will read loadMovieNum. Simply change the location drop-down (in the parameters area of the Actions panel) to Target (which really should be thought of as “clip”). For the URL parameter, just type red.swf, and then in the field to the right of Target, type theClip, because this is the name of the clip into which you want to load the movie (see Figure 19.1).

**FIGURE 19.1**

*Load Movie requires that you specify what movie (URL) you want to load and whether you want to load it into a target (a clip) or a level number. Here, a clip instance name already on the Stage is specified.*

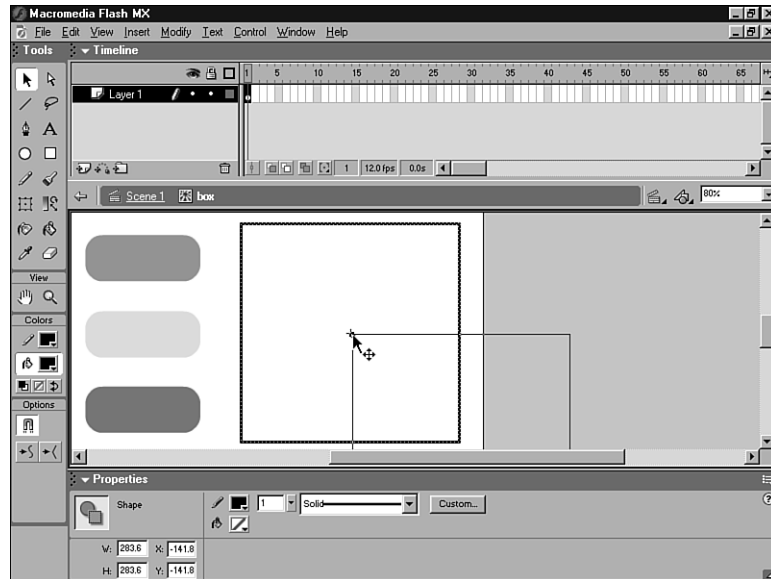


9. Test the movie. It should work, except the movie loads into its box too far down and to the right. Although the movie that gets loaded is the same size as the box, it gets positioned with its top-left corner registered to the center of the clip into which it is loading.
10. The way to fix this is to go inside the master version of the “box” clip and move the contents (the box you drew) so that the top-left corner lines up with the “center” plus sign inside the Movie Clip (see Figure 19.2). You can use the Info panel while inside the Movie Clip to set the top-left corner of the selected shape to position 0,0. When you return to the scene, your clip appears to have moved, but really you’ve just repositioned where the contents load. Move the box to a location toward the center of the screen (basically, wherever you want it).

11. You can now make two more buttons and use the same basic Load Movie Action, except change the URL to point to green.swf and blue.swf.

**FIGURE 19.2**

*Editing the relative center point of the master version of the clip in which you're loading movies will affect the positioning of the loaded movie. That is, the loaded movie's top-left corner corresponds to the clip's center point.*



Flash MX added a simple but powerful enhancement to the loadMovie Action. As an alternative to specifying a .swf to load, you can now specify the filename of an external .jpg. This means you could have a tiny Flash movie that the user visits, but have countless native .jpg images that are loaded as the user requests. You could have an entire portfolio of photographs accessible through Flash. It's as simple as just specifying a .jpg file instead of a .swf in loadMovie's URL parameter.

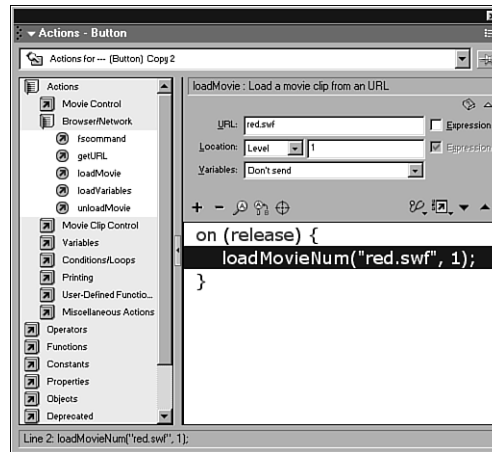
## Targeting Levels or Clips

In addition to learning how to use loadMovie, one lesson you learned in this task was that the top-left corner of loaded movies lines up with the center of the clip into which they are loaded. In the preceding task, you loaded movies into a clip. However, you can also load them into levels. Simply change the Target parameter to Level and type a level number into the field on the right (see Figure 19.3). Personally, I don't prefer loading into levels for two reasons. First, the positioning is *always* based on putting the top-left corner of the loaded movie in the top-left corner of the main movie, which makes registration difficult. Second, you have to keep track of level numbers (as opposed to

clip instance names). It's not as though there's never a need to load movies into level numbers, but for the registration issues mentioned, I don't think doing so is as easy.

**FIGURE 19.3**

*loadMovie can also load a movie (.swf) or image (.jpg) into a level number. You just have to specify "Level" from the Location parameter, and you must specify a number in the field on the right.*



One of the cool things you can do with movies after they're loaded is address them the same way you can address clip instances. More specifically, you can set any property of a loaded movie just like you set any property of a clip. For example, you could stop the loaded movie by adding a button. In this new button, you would insert an evaluate Action and type `theClip.stop()`. (If you loaded the movie into level 1, you would just type `_level1.stop()`.) Remember, there are a bunch of generic properties (such as `_x`, `_y`, `_alpha`, and so on), plus any homemade properties (that is, any variables you happen to be using inside the clip). For a practical example, consider the M3 snowboard site (it's located at [www.m-three.com](http://www.m-three.com) and pictured in Hour 3). Here, we loaded movies of various tricks. From the main movie, users can start, stop, and even jump to any frame in the loaded movie as they scrub.

## Determining When a Movie Is Fully Loaded and How to Unload It

Now that you know how to load a movie, it's a good idea to learn how to unload it. But first, you should learn how to determine whether a movie that's loading has completed loading! This will be important if the movie that's loading is large. It's nice to let the user see that a movie is indeed downloading. You might actually want to *make* the user wait for it to fully load. All these things require you to determine whether a loading movie has downloaded.

To determine whether a movie has loaded, you can use either the two built-in properties `_framesloaded` and `_totalframes` or the two clip methods `getBytesLoaded()` and `getBytesTotal()`. You can use either pair (`_framesloaded` and `_totalframes` or `getBytesLoaded()` and `getBytesTotal()`) and if their values are the same, then you know the movie has downloaded. For example, if you're loading into a clip and you want to know how many frames have loaded, you can use `theClip._framesloaded` (where "*theClip*" is the instance name of the clip). Of course, if you write this script *inside* or *attached* to the clip itself, you don't need to precede the property with the instance name (Flash knows you must be inquiring about the `_framesloaded` property of that clip). Finally, `_framesloaded` and `_totalframes` only make sense when the loaded movie has several frames.

## Task: Determine Whether a Movie Is Fully Loaded

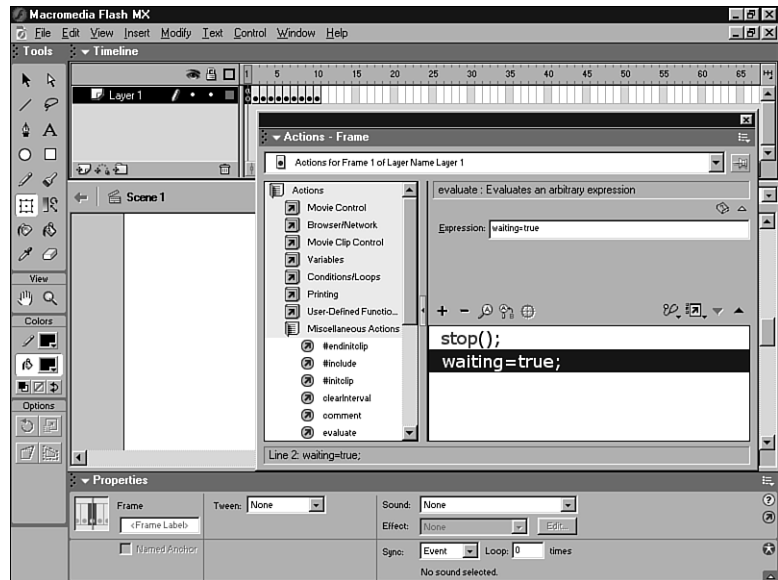
In this task you'll check the `_framesloaded` property against the `_totalframes` property to determine when a loaded movie is totally loaded. Here are the steps to follow:

1. First, create the submovie that will be loaded. Set the document properties (Ctrl+J) to 300×300. Then create a linear animation that starts in frame 2. That is, click frame 2, select Insert Keyframe (or press F6), and then build the animation that plays through many more frames (maybe out to frame 60). If you have a series of bitmaps (like still frames of a video), this would be ideal because you want something that takes a little while to download.
2. In the first frame of this submovie, put a stop Action (Esc, S, T) followed by an evaluate Action (Esc, E, V). Type in the Expression field `waiting=true`, as shown in Figure 19.4. (You'll use this in the main movie to make the host movie wait for the submovie to load.)
3. In frame 1 on the Stage, put the Static Text "Loading..." and then create a Dynamic Text block with "0" in it, but be sure to associate a variable called "percent" with the Dynamic Text (see Figure 19.5). Also set the margins on the text wide enough to accommodate "100". This field will show the user the status of the download.
4. In the last frame of this movie, place a GoTo Action and set the frame parameter to 2. This will cause the animation to loop, but not all the way back to frame 1 (where the loading screen appears).
5. In a new folder, save this movie as `submovie.fla` and then do a Test Movie (Ctrl+Enter) to create an `.swf` called `submovie.swf`. (When you Test Movie, the

movie just sits on frame 1—you'll control this issue from the movie into which it loads.)

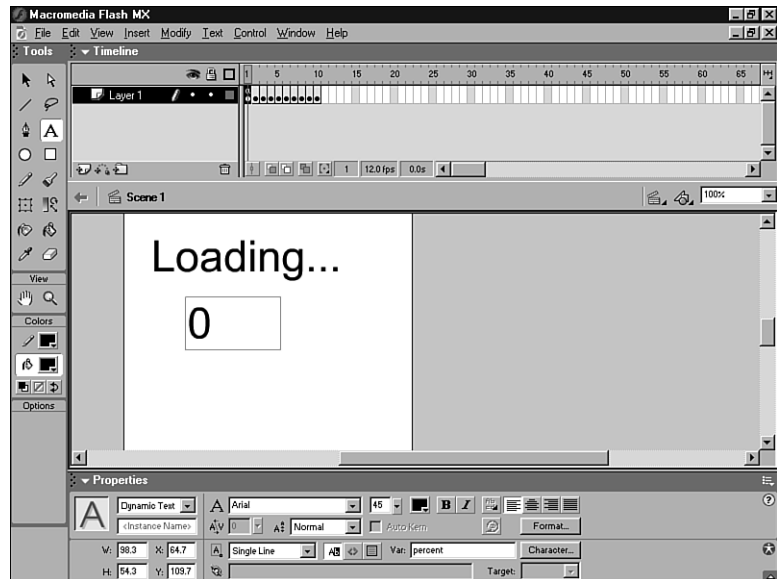
**FIGURE 19.4**

*The variable “waiting” is set to “true” because the host movie is going to wait until this movie is entirely downloaded.*



**FIGURE 19.5**

*Even though the movie's stuck on frame 1, the user will see a message that includes a Dynamic Text block containing a variable (which you'll be changing) called “percent.”*



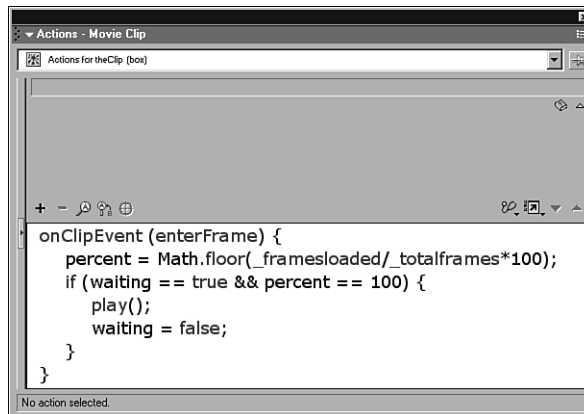


6. Now in a new movie, draw a box that's exactly 300×300 (remove the fill), select Convert to Symbol (or press F8), call it "Holder," and make sure it's a Movie Clip. Be sure to name the instance now on the Stage "theClip" (from the Properties panel). Now, edit the master version of this clip (just double-click it). Select View, Grid, Snap to Grid. Now select the entire square and then grab the top-left corner and snap it to the center of the clip (the little plus sign in the center of the screen).
7. Back in the main scene of this movie, create a button and make a total of three instances of it. In the first instance, insert a loadMovie Action with URL specified as submovie.swf, Location set to "Target," and theClip specified in the field next to "Target."
8. The other two buttons should each use a simple evaluate function using `theClip.play()` for one and `theClip.stop()` for the other.
9. Now it's time to create the script attached to the instance of the Holder symbol (instance name, "theClip"). Select theClip and open the Actions panel. First, insert an "onClipEvent" found under the plus button, Actions, Movie. Change the onClipEvent parameter from "load" to "EnterFrame" (the first line of the script). Then insert an evaluate Action (Esc, E, V) and type the following in the Expression parameter: `percent=_framesloaded/_totalframes` . This tells the Dynamic Text onscreen to display the results of dividing the number of frames completely loaded by the number of total frames (in this case, the number of frames in your sub-movie). This isn't complete, but let's think about it. The variable you're calling "percent" will be .5 when half the total frames load. When they're all loaded, it will equal 1. If you simply multiply this number by 100, you'll get the kind of percentages you expect. Therefore, using `percent=_framesloaded/_totalframes*100` is better. Finally, just so you don't have to come back and fix this later, you can eliminate decimal values if you change the expression to an integer (a number with no decimals). The final version should look like this:  
`percent=Math.floor(_framesloaded/_totalframes*100)`. Notice that the "floor" is calculated (that is, rounding down) by taking everything inside the parentheses to the right.
10. You're not out of the woods yet. This only tells you what percent has loaded. You want to do two things when the movie is totally loaded: tell the loaded movie to start playing and to stop checking whether it's totally loaded. You know you should still be checking when "waiting" is true, and you know the movie will be finished loading when "percent" is 100. Therefore, insert an "if" Action (either type "esc, i, f" or find if under the plus button Actions, Conditions/Loops). Into the Condition field type `waiting==true && percent==100`. This code looks for the condition where "waiting" is true and "percent" happens to equal 100. Notice the

double equal sign. This is different from the normal equal sign, which performs an assignment (that is, `percent=100` will make “percent” equal 100). The double equal sign is a comparison—more like “Does percent happen to equal 100?” Finally, add a simple play Action (that is, if the condition is true, it’s time to play the movie). Also, you should turn “waiting” to false so that checking can stop. To do this, add a simple evaluate Action with `waiting=false`. Figure 19.6 shows the finished version of this code.

**FIGURE 19.6**

*The finished code attached to the clip into which the movie loads.*



11. Test it out. The movie should work great. If you really want to test it, put it up on a Web server and try it from a slow connection. Note that if you test it twice, it might appear to download instantly the second time. That’s because the loaded movie has already downloaded to your browser’s cache. Basically, you either have to clear the cache or delete the .swfs from the folder full of downloaded files (for example, Internet Explorer uses `C:\WINDOWS\Temporary Internet Files`).

Now that you know how to load movies and check to see whether they’ve totally loaded, it’s important to learn how to unload them. There are two basic methods. One, you simply load something else into the movie’s place (effectively removing one and replacing it with another). It may not seem very clean to blast one loaded movie away by loading another in its place, but it’s good to know you can and that there’s no problem doing so. Otherwise, you might find yourself unloading one movie, only to immediately load another movie right away. Two, you can use the Action `unloadMovie`. The only trick with the `unloadMovie` Action is that you have to remember the location where you loaded the movie (that is, a target clip or a level number) so that you can unload it from the same location. Other than that, unloading is quite simple.

## Shared Library Items

The whole idea of items stored in your Library is that they're only stored once regardless of how many times they're used in a movie. However, an item in one movie's Library isn't automatically recycled if you copy it into another movie. Flash has two (confusingly similar) features to let you share Library items among your movies. There are two ways to do it: runtime sharing and authortime sharing. Runtime sharing involves storing Library items in a single source movie (that gets exported as a .swf). Then, one or more user movies can access the source .swf's items. This way if the source .swf ever gets updated the changes will be reflected in all user files. This is just like making an edit to a master symbol except this time multiple movies will reflect the change instead of just multiple instances within one movie.

Authortime sharing is slightly less complex. You still have one source movie containing Library items. But with authortime sharing, the shared items get recopied into each user file every time you do a test movie. That is, the user files copy the Shared Library item at the time of export. So, if a change occurs in the source item, all user files have to be reopened and exported to reflect the change. Plus, each copy of the source adds to the file size of the user files. It's true that authortime sharing is not as modular—but it's appropriate for larger common elements that you want copied into each user file. One limit of runtime sharing is that the entire Shared Library has to fully download before a user file can begin playing. In this way, runtime sharing isn't good for many or large items.

You're about to learn the steps to create and use shared items in the runtime mode. In Hour 22, "Working on Large Projects and in Team Environments," you'll walk through a task involving authortime sharing.

19

## How to Share Library Items at Runtime

Sharing items at runtime involves two basic steps. First, you need to create the Library containing items to be shared. Then, in each file that will use the shared items, you need to establish a link to the source.

### Task: Prepare Items to Share at Runtime

In this task you'll create a Library containing items that can be shared. Follow these steps:

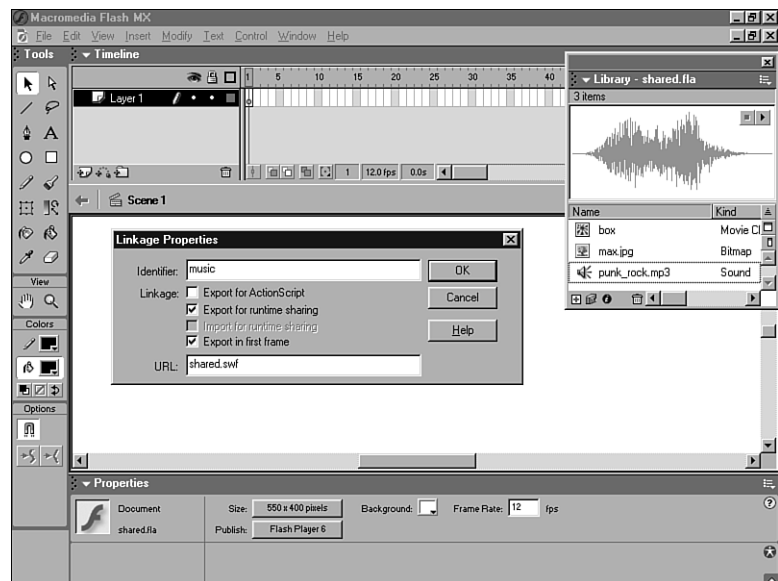
1. Create a new file and immediately save it as `shared.fla` in a folder called `RuntimeSharing`. Import one bitmapped image (.jpg, .bmp, or .pct, for example).

Delete the bitmap onstage (don't worry, it's still safe in the Library). Also, import one sound file. Finally, create a rectangle and convert it to a symbol (press F8). Then make it a Movie Clip, name it "box," and click OK.

2. You should now have a Library with three items. Rename each item to "image," "sound," and "symbol," respectively. Generally, the next step is to simply export a .swf containing these three Library items. However, normally when you export a .swf, all unused Library items are excluded; therefore, they don't contribute to the file size. Normally, this is a good thing. The problem is that this time you want all the items in the Library to export with the file! Sure, this will add to file size, but you want these assets to be available to many user files. To ensure that each item is included in the .swf export process, you'll define the linkage properties for each item.
3. To define the linkage for the sound you imported, find it in the Library, select it and then select Linkage... from the options menu (or right-click). Select the check box next to "Export for runtime sharing." This will expose two other required fields: Identifier and URL. We'll discuss the identifiers later, but generally I recommend trying to use the item's name for an identifier when you can (as you'll see by default) or something generic like "music". Into the URL field type shared.swf because that's the name for this movie once exported. Figure 19.7 shows the linkage settings you should use.

**FIGURE 19.7**

*Library items' linkage is set to Export for runtime sharing. This way the item will definitely export even if it's not being used in this file.*



4. Go ahead and define the linkage for the bitmap and box movie clip the same way (click Export for runtime sharing and type `shared.swf` into the URL field).
5. Now, save (Ctrl+S) and then export this file. You can just do a “Test Movie” (Ctrl+Enter) because it’s already saved in the right folder.
6. Your source items are now ready to be shared. You can think of this as your Shared Library.

Now you can use the shared items in any other movie. Inside the other movies, the items simply need to be set to “Import for runtime sharing.” In the following task, you’ll start *using* the items from the Shared Library, and then it should start to make sense.

## Task: Start Using a Shared Item

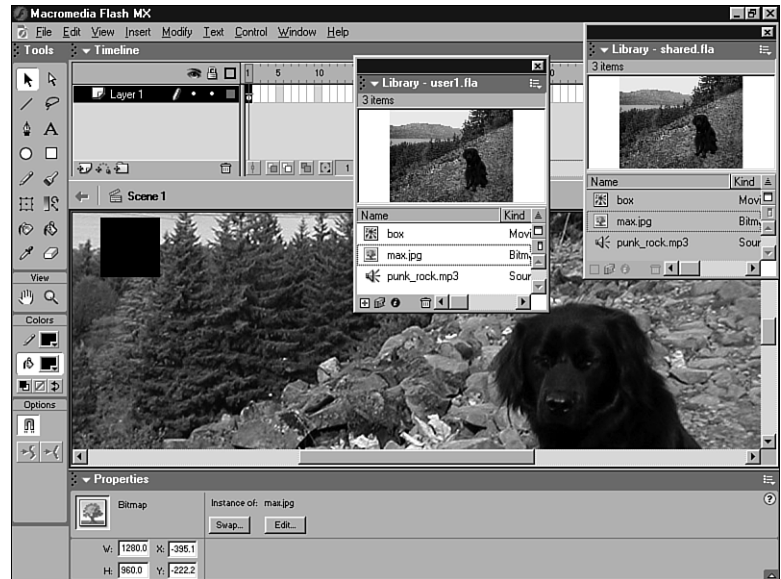
In this task you’ll create a user file that can access the contents of the Shared Library you created in the previous task. Here are the steps:

1. Create a new file (Ctrl+N) and select Save As.... Name the file `user1.fla` and save it in the RuntimeSharing folder (from the previous task). Now you have to get those shared items into the new movie. You *could* copy instances from the `shared.fla`—but this would involve first creating instances (by dragging from the Library) and then pasting into `user1.fla`. I think an easier way is to open `shared.fla` as a Library. To do that, make sure you’re editing `user1.fla` and select File, Open as Library and then select `shared.fla`. You’ll see the Library from `shared.fla`, but notice that it’s darkened a bit to represent that it’s from another file. You can now use instances from this movie without adding to the file size of the `user1.fla` file.
2. Drag items from `shared.fla`’s Library to the Stage of `user1.fla`. Drag the bitmap onto the Stage. Then open the Library for `user1.fla` (Ctrl+L) and notice that the bitmap looks like it was copied into the Library. Now, try dragging the sound from the `shared.fla`’s Library to `user1`’s Library. Finally, drag the “box” onto the Stage. File `user1.fla` now seems to have three items added to its Library (see Figure 19.8). However, these are really just instances pointing to the master version in the file `shared.fla`.
3. Go ahead and double-click the “box” clip in your `user1.fla` movie. You’ll see a dialog box like the one shown in Figure 19.9. Click No. The point is that you can only make edits to the master when you open `shared.fla`. By the way, you can drag instances from `user1`’s Library onto the Stage, and you’re still using instances of the masters saved in `shared.fla`.
4. Do a Test Movie and then look at the file sizes. The `shared.swf` file should be rather large (it has a bitmap, sound, and Movie Clip). The `user1.swf` file should be

tiny, because it only points to the shared.swf file. By the way, if you move or change the name of shared.swf, the file user1.swf won't work!

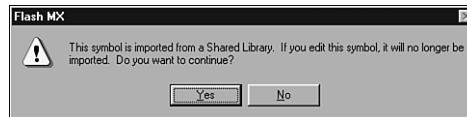
**FIGURE 19.8**

*After you bring assets from a Shared Library into a user file, they appear to be included in the user file's Library.*



**FIGURE 19.9**

*Trying to edit a clip set up for runtime sharing will cause this dialog box to appear. It basically says that if you want to edit the clip here, it won't be linked any more. Editing the original is done through the source movie.*



5. While you're inside user1.fla take a look at the linkage settings under each item's linkage properties. You should see "Import for runtime sharing" selected as well as the identifier and URL fields filled in. These settings mean that, when exported, this movie will look in the file shared.swf for items with the identifier listed.

What's the big deal? You have two files, and one's bigger than the other. So what? First of all, you can repeat the last task, but this time you can name the file *user2.fla*. You can

use instances of the same symbols (maybe in different ways), and user2.swf will be tiny, too! Visitors to your site will download shared.swf only once, regardless of whether they view user1.swf or user2.swf, or both. Additionally, you can change the contents of shared fla—maybe change the bitmap or the “box” symbol. Making a change and exporting shared.swf again will enable all your user files (user1.swf, for example) to reflect the change. This is just like how you can change a master symbol and every instance reflects that change. You’ll now give it a try. (If you want to first repeat the preceding task, but name your file user2 fla, you may.)

## Task: Update Shared Items in a Library

In this task, you’ll learn how to update the contents of a Shared Library. Follow these steps:

1. Open the source of your Shared Library (shared fla). Take note of what the identifier is for the bitmap (select it and select Linkage ... from the Library’s Options menu). Delete this bitmap from the Library and import another bitmap that will serve as a replacement.
2. Access the newly imported bitmap’s Symbol Linkage Properties via Linkage... on the Options menu, set it to Export for runtime sharing, and specify the same identifier as the old bitmap (probably image).
3. Save the movie and export an .swf (Ctrl+Enter).
4. Now double-click any user files you’ve made (user1.swf, for example). They should now reflect the change.

That was almost too easy. You simply created a new Library item in the master Shared Library file (shared fla), made sure it would export, and gave it an identifier name that matched the old one. Then, as soon as you exported the .swf, it worked. Basically, each user file doesn’t care what’s inside the master Shared Library; it simply looks for items that match identifier names. For example, if you have a Movie Clip containing English language text and swap it with one containing Spanish, as long as the identifier is the same, it will work fine (it would just appear in a different language).

Although you can probably start to see how Shared Library items can enhance productivity, remember that their other benefit is file size savings. The shared.swf file only downloads once. This might make you want to include larger items (like sounds and raster graphics). But you should know that when sharing items the source (Shared Library) must download entirely before the user files begin to play. For this reason you really should use loadMovie for larger elements so that you can control and monitor their downloading (like you did in an earlier task). The most appropriate time to use runtime

sharing is when you have lots of small items that you expect might need to change periodically—for example, a seasonal icon used in several different pages of your Web site. The icon (saved as Export for runtime sharing) could change from themes for Halloween to Thanksgiving to New Year’s.

Fonts are another Library type that can be shared. If you use the same font throughout several movies, sharing that font during runtime makes sense. You’ll do just that in the next task.

## Task: Share a Font During Runtime

In this task you’ll set up a font as a shared item so that it can be used by as many user files as you want. Here are the steps:

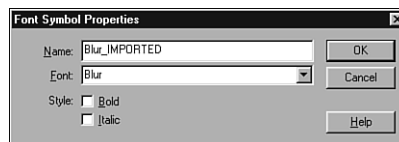
1. In a new file (or your shared.fla source), open the Library window. From the Options menu, select New Font....
2. You’ll see the Font Symbol Properties dialog box, which requires you to specify both a new name for your font and which font you want to copy. Suppose you own a font called *Blur* that you want to share. You need to find and select *Blur* from the Font drop-down list and then give it a new name. Any name will work (except a name used already). However, let me suggest that you use the original font name plus “\_IMPORTED.” This way, when you’re in the user file, you can find the font alphabetically, but there will be no question which one is imported. Notice in Figure 19.10 that I’ve given the *Blur* font the name *Blur\_IMPORTED*.

**FIGURE 19.10**

*When creating a new font, you must specify the font and the name you want to give it.*

*Adding*

*“\_IMPORTED” to the end of the original name makes it clear which font is the “imported” one.*



3. Now you need to make sure Flash exports the font when you export the movie. So, after you’re done making a new font, you should notice the “font” item in your Library. Select it and access its Linkage properties (right-click or select the Options menu). Choose Export for runtime sharing and give it the unique identifier



- “myBlur” (it just has to be a unique name). Also type `shared.swf` into the URL field.
4. Save this movie as `shared.fla`, export it (Ctrl+Enter), and leave it open. Create a new file and save it as `user_of_fonts.fla` in the same location as `shared.fla`.
  5. In the last task we used the Open as Library feature. This time, just expand the Library for `shared.fla` (it should be in gray) and the Library for the new `user_of_fonts.fla` (Press Ctrl+L if necessary). Drag the font `Blur_IMPORTED` (or whatever you named the font) from the `shared.fla`’s Library to your open file’s Library. Now, finally, we can use this font. Simply create some text and use the Properties panel to select `Blur_IMPORTED` from the list of fonts! (See how the name you chose made it easy? Flash also puts an asterisk after any imported font names.)

One last note on this subject: Sharing font items isn’t just for file size savings. You can use them like styles in a word processor. For example, you can go back to the `shared.fla` file and edit the font (just double-click the font item in the Library and choose a different font from the Font Symbol Properties). If you then re-export (to replace) `shared.swf`, you’ll see the new font being used in each user file. Notice that in this case, because you’re just editing a symbol (the font) you don’t have to worry about the linkage settings. If, on the other hand, you actually replaced an entire Library item, you would need to set the linkage and provide the same unique identifier used earlier.

Unfortunately when you change the shared items (in `shared.fla`) you won’t see the change in the user `.fla` files (just their `.swfs`). You can get around this fact (as you’ll learn in Hour 22, “Working on Large Projects and in Team Environments”) by using a feature called “Always update before publishing”—however, this only works with shared symbols (not sounds, bitmaps, and fonts).

Basically, all the upfront work in runtime sharing can be very useful, provided you use the same styles and assets throughout many files. It also means that everyone on the team could be given a copy of the latest version of `shared.swf` to use on their computers. In the end, when everything’s published, the latest version of `shared.swf` will be used by everyone. All it takes is a little planning.

## Linked Scripts

I want to mention the topic of linked scripts here because it fits very well with the other topics mentioned this hour. Remember all the ActionScripting you did in the Actions panel? Anything you can type in that panel can be stored as text in a text file. Let me just show you how it works.

The options arrow in the Actions panel allows you to “Export as File...” any code you’ve created in Flash. You can also copy and paste from Flash’s Actions panel into a text editor. Flash happens to be a convenient place to create ActionScript—it colors your text and has easy-to-use parameter drop-down menus, for example. But because Flash is almost identical to JavaScript, you may likely find that a third-party text editor will do almost as well. You can create a text file full of ActionScript by simply typing by hand into a text editor. It isn’t terribly important how you get your ActionScript into a text file, but that’s the first step to using linked scripts.

After you’ve created your text file full of ActionScript, you can return to Flash. Use the Actions panel to select the Action “include” (under the plus button, Actions, Miscellaneous Actions) and specify, as its parameter, a named text file. For example, if in a keyframe you open the Actions panel and place the script `#include "scripts.txt"`, Flash will use the contents of the file `scripts.txt` as if it were inside this keyframe. Presumably, the content of `scripts.txt` is a legitimate ActionScript. This is great because you can change the script without ever opening Flash. Just remember if you change the script file, you’ll need to export the `.swf` again (as this is when the code is copied into Flash). This is a great feature.

One note: Macromedia suggests that you use “.as” as a file extension (instead of “.txt”) so that everyone knows the file contains ActionScript. (Get it? The `.as` extension stands for *ActionScript*.)

Just like `loadMovie` and Shared Library items, external scripts let you modularize your projects for easy updates. The key is that you must plan for likely updates.

## Summary

This hour you saw several ways to cut up your movies into little pieces. Such modularization can have performance advantages as well as productivity value. The perceived download time is reduced if users only need to download segments as needed. You and your team’s productivity is increased by breaking a site into smaller files—not only can several people work simultaneously but you can impose consistent styles with Shared Library items. You also learned the steps necessary to modularize. However, the challenging and creative part comes as you decide when and where to modularize.

The first feature discussed was the `loadMovie` Action. This allows you to (at runtime) play another `.swf` on a level number or in a clip instance. You also learned about sharing Library items at runtime, where an `.swf` can act as a Library shared between several files

or among several team members. A lot of upfront work is involved to get such a Shared Library built, but this time investment will be paid back in the form of greater productivity. You can share any kind of Library item, including a font. Finally, I pointed out that there's a way to store your scripts outside the Flash file (as a text file) so that you can share one script and make edits to it without constantly returning to Flash.

## Q&A

**Q I've created the submovies that I intend to load into my main movie via loadMovie. I've saved them all in the correct folder, but when I test the main movie I get the error Error opening URL "c:\windows\desktop\somefolder\submovie.swf" in the output window (and my movies don't load). What's going on?**

**A** Most likely, when you made your various submovies, you saved them correctly, but you need to take the extra step of exporting them as an .swf (simply do a Test Movie). Only .swfs or .jpgs can be loaded using loadMovie.

**Q My Web site is getting pretty messy with all the little submovies and .jpgs in the main folder. Is there a way I can keep the movies that load in a separate folder to keep everything straight?**

**A** Yes. When you specify (in the URL field of loadMovie's parameters) the movie you want to load, if you simply type mymovie.swf, Flash will look for mymovie.swf in the same folder. If you want to store mymovie.swf in a subfolder called "movies," you would change the URL field to read movies/mymovie.swf. (By the way, you can use all the standard HTML relative references as well, such as "../" to mean "up one folder.")

**Q If I know I'm going to use a Shared Library item in a project, but I don't actually have the master media elements (such as the bitmaps I want to share). Is there any way I can start now?**

**A** Sure. Bitmap items properties dialogs have two buttons—"update" and "import"—which let you re-import (and replace) a previously imported bitmap. The particular scenario (of wanting to work without the final artwork) is explored in a task coming up in Hour 22 involving so-called "authortime sharing" (compared to this hour's runtime sharing examples). You may want to explore that task because it should help you fully understand the difference between authortime and runtime sharing.

## Workshop

The Workshop consists of quiz questions and answers to help you solidify your understanding of the material covered. Try to answer the questions before checking the answers.

### Quiz

1. How many movies can you load into `_level13`?
  - A. None, this level is reserved for sounds.
  - B. As many as you want.
  - C. One.
2. If you want to rotate (that is, set the `_rotation` property of) a movie that gets loaded, into what location must you load it?
  - A. A Target clip.
  - B. A `_level` number.
  - C. Either Target or Level.
3. How many people can simultaneously edit the master version of a shared item?
  - A. One.
  - B. As many people as you want.
  - C. A number equal to how many user files you have.

### Quiz Answers

1. C. Into any level number (or into any clip instance, for that matter) you can only load one movie or .jpg at a time. If you try loading another movie into an occupied level or clip, the new movie just replaces the current occupant.
2. C. Although changing the properties (including `_rotation`) of a clip instance (because you just have to use `clipInstanceName._rotation=180`) may seem easiest, you can also set a property of any level number—for example, `_level11._rotation=180`. I think it makes more sense to load movies into named clips, but you don't have to.
3. A. Only one person can actually edit the master version of the shared items. After it's turned into an .swf, you can share it throughout a network—or just redistribute copies to everyone involved.