"He who dies with

the most toys, is,

nonetheless, still dead."

—ANONYMOUS

PRODUCT CATALOG

Building a scalable application for product demonstrations

One of the most popular uses of multimedia on the Web is for product demonstrations prior to sale. By taking advantage of Flash's modular downloading and scripting features, you can create a scalable, robust Product Catalog application. The Product Catalog incorporates a wide range of techniques, including scrolling text fields and reusable interface elements.

PROJECT 11:
PRODUCT CATALOG

Layers

- ActionScript Library
- View Port
- Text
- Scrollbars
- Buttons
- Background

GETTING STARTED

If you want your results to match those of the finished project, follow the steps carefully. If you want to customize the project and are familiar with Flash 4, feel free to change the settings to those that meet your needs.

1 Copy all the .swf files from the ProductCatalog\ProductFiles folder into the folder on your hard disk in which you'll be working.

Flash loads these .swf files into the catalog; they are essential for testing purposes.

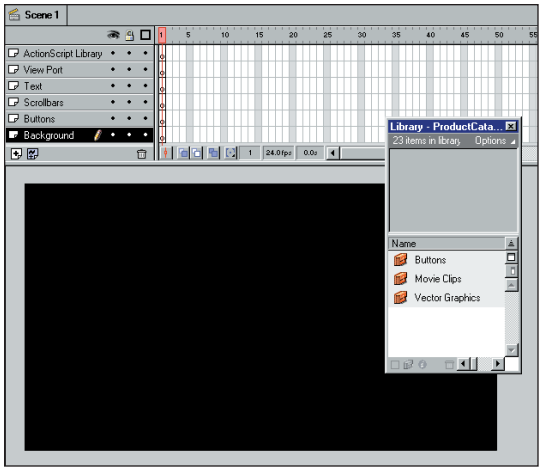
Copy all the .swf files from the ProductCatalog\ProductFiles folder into a folder on your hard disk.

2 Begin with a new Flash Movie, using these settings:

Frame rate:	**24 fps**
Stage dimensions:	**550 × 310**
Background color:	**Black**

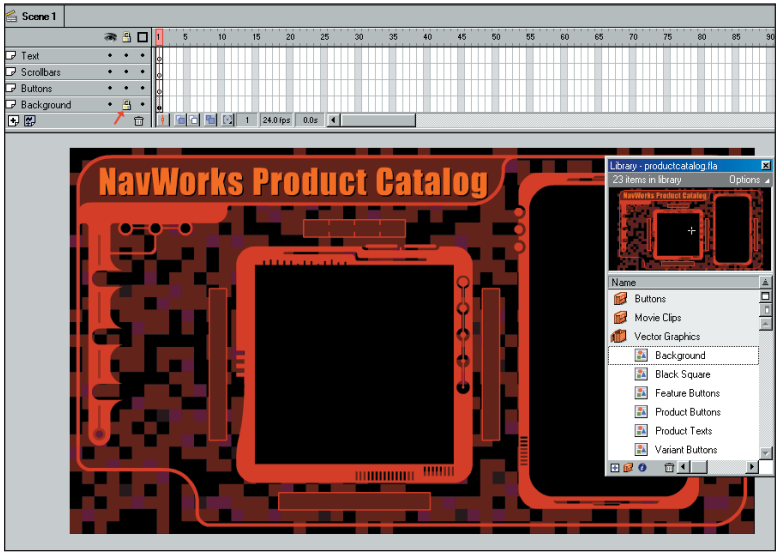
3 Set up six layers, naming them (from top to bottom) **ActionScript Library**, **View Port**, **Text**, **Scrollbars**, **Buttons**, and **Background**.

4 From the Flash 4 Magic CD, load ProductCatalog.fla as a Library.



Load the ProductCatalog.fla file as a Library.

5 Select the Background layer, drag an instance of the Background graphic symbol from the Vector Graphics folder onto the Stage, center the instance on the Stage. If you want, you can lock the Background layer. You won't be dealing with it again in this project.



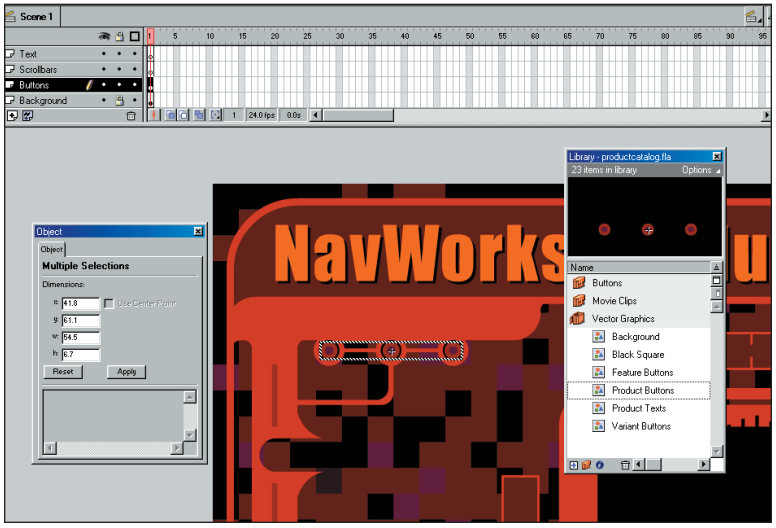
Center the Background graphic symbol on the Background layer.

Laying Out the Product Buttons

While Product Buttons is a graphic symbol, it is in fact a group of three buttons. Each of the three buttons represents a product that visitors can view when they click the appropriate button in the catalog.

1. Select the Buttons layer and drag an instance of the Product Buttons symbol in the Vector Graphics folder onto the Stage.

2. Select your instance of Product Buttons and use the Object Inspector to set the coordinates to x: **41.8**, y: **61.1**.



Use the Object Inspector to position the Product Buttons instance.

3. Enter the symbol-editing mode for the Product Buttons graphic symbol instance, select the far left button (FlashPack 1 Button), double-click to open its Instance Properties dialog box, and, on the Actions tab, assign the ActionScript.

Note: To enter the symbol-editing mode, right-click (Windows) or Control-click (Macintosh) on the instance, and choose Edit from the pop-up menu.

```
On (Release)
    Set Variable: "Product" = "flashpack1"
    Set Variable: "Text" = FlashPack1Text
    Set Variable: "Text.scroll" = 1
    Set Property ("/TextScroll/ScrollHandle", X Position) = 0
    Call ("/Library/:LoadView")
End On
```

The script sets the current product to FlashPack 1, loads the corresponding product text into the text area, and scrolls it to the top. The **.scroll** property of the text field named Text enables your text scroller, but more on that later. The script also sets the x-position of the text scroll handle to **0**. This is

necessary because the product text might have changed and therefore is scrolled to the top for convenience. Finally, the button calls the **LoadView** function, which loads the current product's picture into the main viewer.

4 Double-click the FlashPack 2 Button to open the Instance Properties dialog box and, on the Actions tab, assign the ActionScript.

```
On (Release)
    Set Variable: "Product" = "flashpack2"
    Set Variable: "Text" = FlashPack2Text
    Set Variable: "Text.scroll" = 1
    Set Property ("/TextScroll/ScrollHandle", X Position) = 0
    Call ("/Library/:LoadView")
End On
```

5 Double-click the FlashPack 3 Button symbol to open the Properties dialog box and, on the Actions tab, assign the ActionScript.

You'll notice that each script is identical except for the **Product** and **Text** values.

```
On (Release)
    Set Variable: "Product" = "flashpack3"
    Set Variable: "Text" = FlashPack3Text
    Set Variable: "Text.scroll" = 1
    Set Property ("/TextScroll/ScrollHandle", X Position) = 0
    Call ("/Library/:LoadView")
End On
```

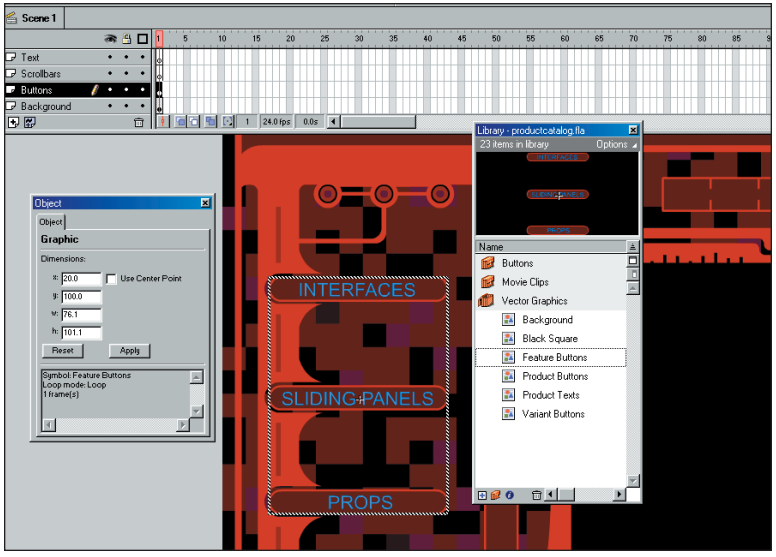
6 Exit symbol-editing mode and return to Scene 1.

Setting Up the Feature Buttons

Each product has three different features: interfaces, sliding panels, and props. In turn, each feature has three variations to give the user a good idea of what each product includes.

As with the Product Buttons graphic symbol, the Feature Buttons graphic symbol contains nested buttons to make positioning the buttons easier. This time around, you set the value of the currently selected Feature and then call the **LoadView** function to display the appropriate image.

1 Select the Buttons layer, drag an instance of the Feature Buttons graphic symbol from the Vector Graphics folder onto the Stage, and use the Object Inspector to position the instance at x: **20.0**, y: **100.0**.



Use the Object Inspector to position the Feature Buttons graphic symbol.

2 Open the Instance Properties dialog box of the Interfaces Button and, on the Actions tab, assign the ActionScript.

```
On (Release)
    Set Variable: "Feature" = "interface"
    Call ("/Library/:LoadView")
End On
```

3 Open the Instance Properties dialog box of the Sliding Panels Button symbol, and, on the Actions tab, assign the ActionScript.

```
On (Release)
    Set Variable: "Feature" = "slidingpanel"
    Call ("/Library/:LoadView")
End On
```

4 Open the Instance Properties dialog boy of the Sliding Props Button symbol, and, on the Actions tab, assign the ActionScript.

```
On (Release)
    Set Variable: "Feature" = "props"
    Call ("/Library/:LoadView")
End On
```

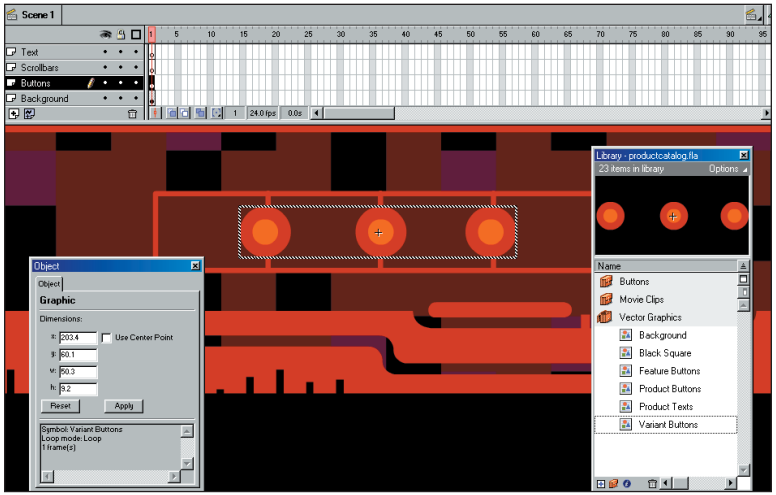
5 Exit symbol-editing mode and return to Scene 1.

SETTING UP THE VARIANT BUTTONS

The final set of buttons controls the current feature variation, or Variant.

1 On the Buttons layer, drag an instance of the Variant Buttons symbol from the Vector Graphics folder onto the Stage, and use the Object Inspector to position the symbol instance at x: **203.4**, y: **60.1**.

This time each button sets the value of **Variant** to either **1**, **2**, or **3** and then calls **LoadView** again to refresh the currently displayed picture.



Use the Object Inspector to position the Variant Buttons symbol.

2 Open the Instance Properties dialog box of the Left Button symbol, and, on the Actions tab, assign the ActionScript.

```
On (Release)
    Set Variable: "Variant" = "1"
    Call ("/Library/:LoadView")
End On
```

3 Open the Instance Properties dialog box of the Middle Button symbol, and, on the Actions tab, assign the ActionScript.

```
On (Release)
    Set Variable: "Variant" = "2"
    Call ("/Library/:LoadView")
End On
```

4 Open the Instance Properties dialog box of the Right Button symbol, and, on the Actions tab, assign the ActionScript.

```
On (Release)
    Set Variable: "Variant" = "3"
    Call ("/Library/:LoadView")
End On
```

5 Exit symbol-editing mode and return to Scene 1.

Setting Up the Scrollbars Movie Clip

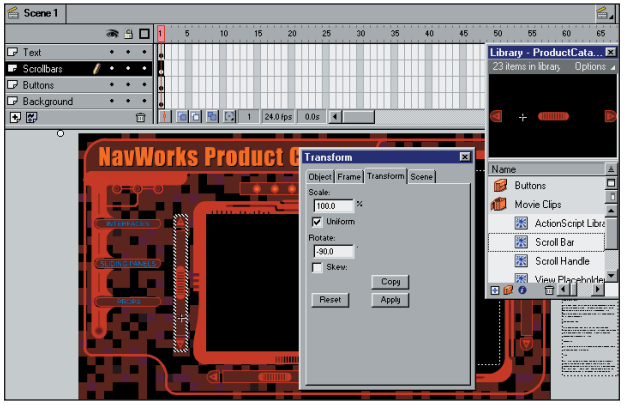
As always, you should be on the lookout for elements in your movies that can be reused as instances of a single symbol instead of wasting precious file size with unnecessary bloat. In this case, you use the identical Scroll Bar symbol four times, performing four completely different functions.

1 Select the Scrollbars layer and drag four instances of the Scrollbar Movie Clip from the Movie Clips folder onto the Stage.

You'll name and position each instance and then edit the symbol to activate it.

2 Double-click an instance of Scrollbar, name the instance **Zoom**, and position and rotate it:

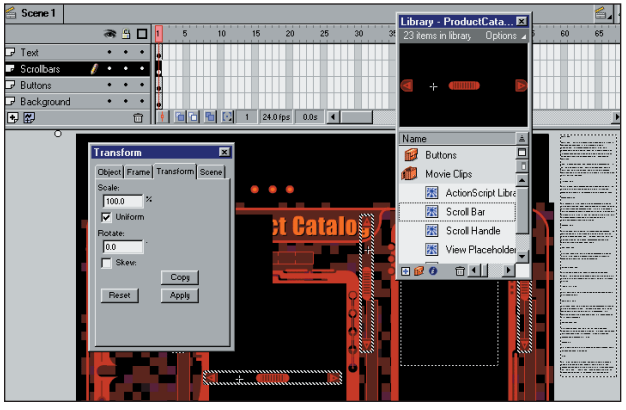
Object Inspector:	x: **112.1**
	y: **95.1**
Transform Inspector>Rotate:	**-90.0**



Name one of the Scrollbar instances Zoom and use the Transform Inspector to rotate it and the Object Inspector to position it.

3 Double-click another instance of Scrollbar, name the instance **ScrollX**, and position and rotate it:

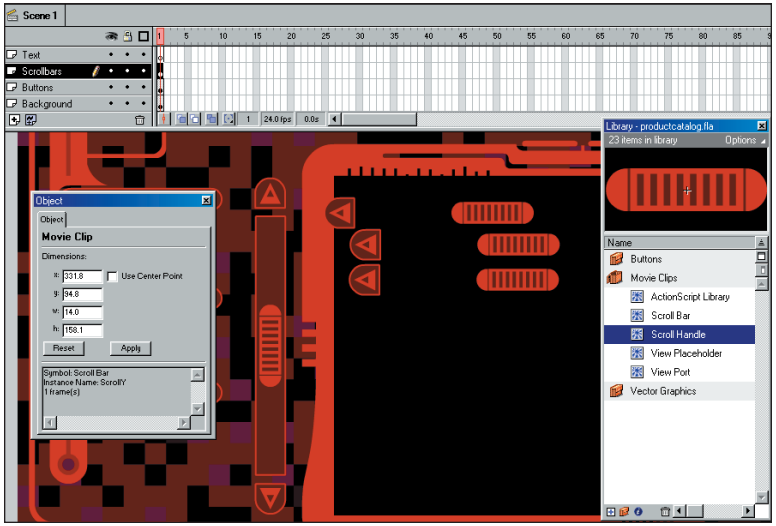
Object Inspector:	x: **149.8**
	y: **276.9**
Transform Inspector>Rotate:	**0.0**



Name one of the other Scrollbar instances and rotate and position it.

4 Double-click another instance of Scrollbar, name the instance **ScrollY**, and, using inspectors, position and rotate it:

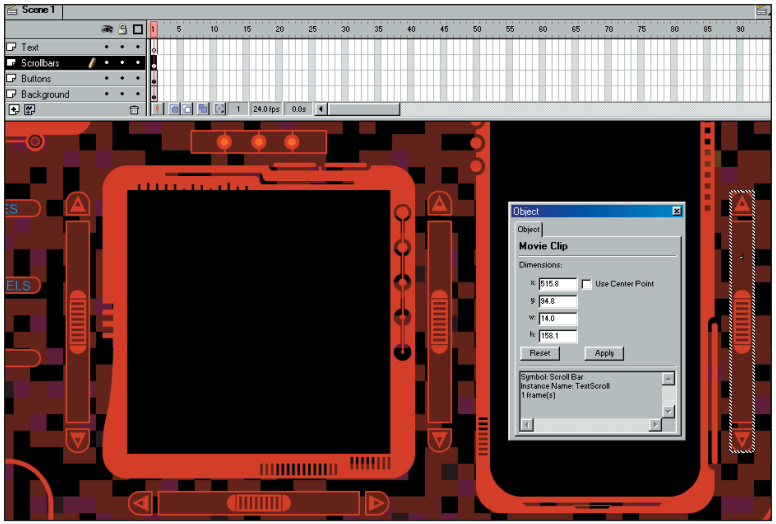
Object Inspector: x: **331.8**
y: **94.8**
Transform Inspector>Rotate: **90.0**



Name another Scrollbar instance and rotate and position it.

5 Double-click another instance of Scrollbar, name the instance **TextScroll**, and, using inspectors, position and rotate it:

Object Inspector: x: **515.8**
y: **94.8**
Transform Inspector>Rotate: **90.0**

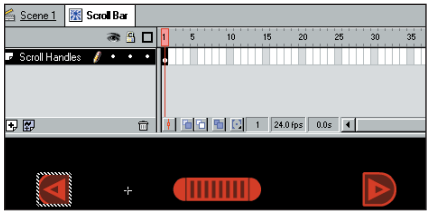


Name the fourth instance TextScroll and rotate and position it, using the inspectors.

EDITING THE SCROLLBAR

The Scroll Bar symbol consists of two scroll buttons and a middle "scroll handle" that is, in fact, a button nested within a Movie Clip. The Scrollbar works when users click the left or right buttons, or drag the scroll handle. Flash then uses the x coordinate information of the scroll handle to adjust the view, including zooming, scrolling, and text-scrolling.

1 Open your movie's Library and double-click the Scroll Bar symbol's icon to edit it.



Edit the Scroll Bar symbol to observe that it is made up of three components: two scroll buttons with a scroll handle between them.

2 Double-click the left button to open the Properties dialog box and, on the Actions tab, assign the following ActionScript statements to adjust the scroll handle's position:

This script checks to see if the scroll handle is within 5 pixels of the left end of the bar, and repositions it accordingly.

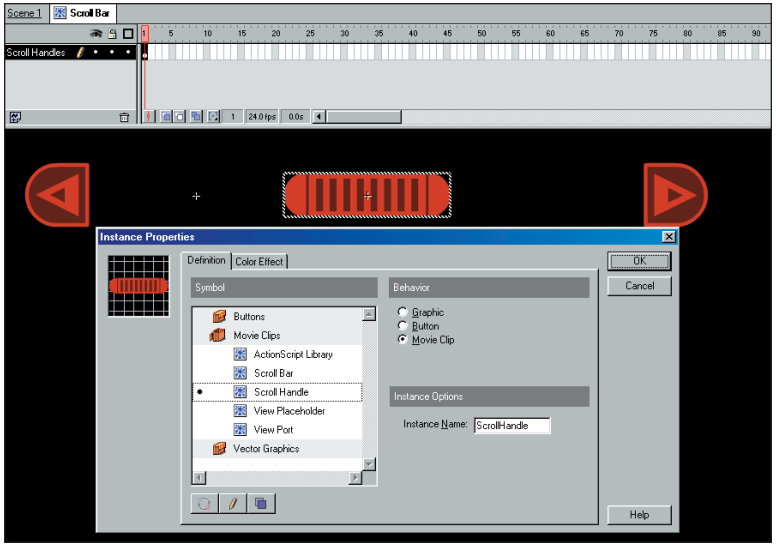
```
On (Release)
    If (GetProperty("ScrollHandle",_x) > 0)
        If (GetProperty("ScrollHandle",_x) > 5)
            Set Property ("ScrollHandle", X Position) =
                GetProperty("ScrollHandle",_x) - 5
        Else
            Set Property ("ScrollHandle", X Position) = 0
        End If
    End If
    Call ("/Library/:AdjustView")
End On
```

3 Double-click the other scroll button (to the far right) to open the Properties dialog box and, on the Actions tab, assign this complementary set of ActionScript statements:

This script is identical to the first except it moves the scroll handle right instead of left.

```
On (Release)
    If (GetProperty("ScrollHandle",_x) < 80)
        If (GetProperty("ScrollHandle",_x) < 75)
            Set Property ("ScrollHandle", X Position) =
                GetProperty("ScrollHandle",_x) + 5
        Else
            Set Property ("ScrollHandle", X Position) = 80
        End If
    End If
    Call ("/Library/:AdjustView")
End On
```

4 To enable the scroll handle itself, select the Scroll Handle symbol, name the instance **ScrollHandle**, right-click the symbol, and choose Edit in Place to access the symbol's contents.



Give the scroll handle an instance name.

5 Select the Scroll Handle Button layer, double-click the instance of Scroll Handle Button on the Stage, and, on the Actions tab, assign the ActionScript.

As mentioned earlier, the Scroll Handle button allows the scroll handle to be draggable. Its positioning information is then used by the **AdjustView** function to set the zooming and scrolling of the product viewer and text area.

```
On (Press)
    Start Drag ("", L=0, T=0, R=80, B=0)
End On
On (Release, Release Outside, Drag Out)
    Stop Drag
    Call ("/Library/:AdjustView")
End On
```

6 Exit symbol-editing mode and return to the main Timeline.

Creating the Text Area

Each product in the Product Catalog can have a block of accompanying text displayed to the right of the picture viewport. By taking advantage of the **.maxscroll** and **.scroll** properties of text fields, you can easily implement a scrollable text box.

First of all, you need the blocks of text for each product. The simplest way to do this is to have three separate text fields that sit offstage, storing all the data. Then, using

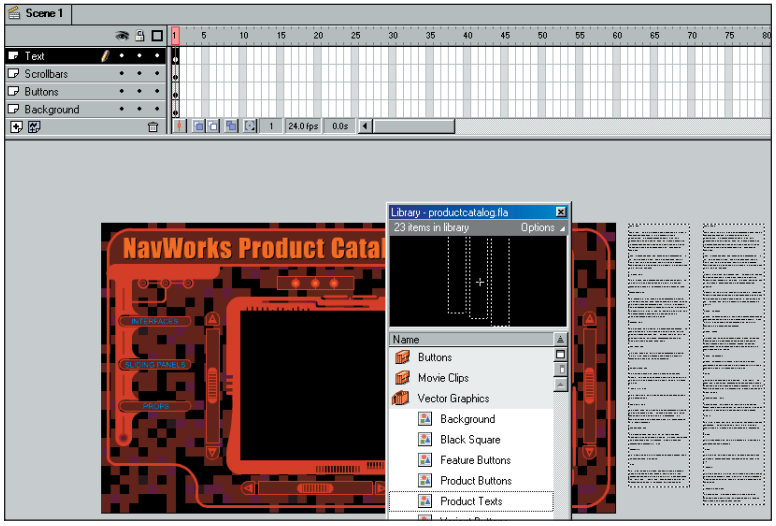
ActionScript, you can transfer the appropriate data into the main text box when a new product is selected.

In the Vector Graphics folder of ProductCatalog.fla, you'll find a symbol named Product Texts. This symbol contains three text fields: FlashPack1Text, FlashPack2Text and FlashPack3Text. The fields have been shrunk to make them more manageable.

1 To plug the data into your movie, select the Text layer and drag an instance of Product Texts off to the right of the Stage where it won't be seen.

The last text element required is the actual display field. You're going to set up its text properties.

These properties enable the text field to display data, be scrollable, and display consistently across platforms. Because you disabled editing but left selection enabled, users can copy and paste text about each product into an e-mail message or word processing document.



Drag an instance of the Product Texts symbol onto the Stage, but out of view.

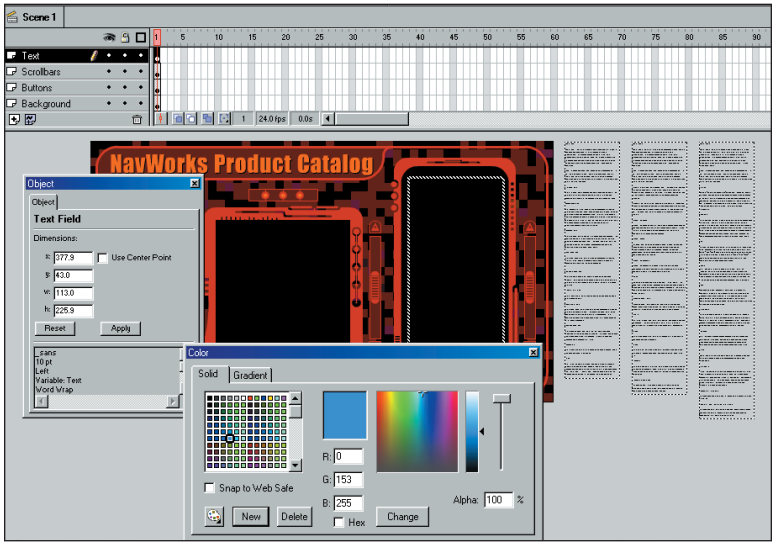
2 Select the Text tool and set the following text properties, making certain that the Text Field modifier is selected:

Font: _sans
Font Size: 10 pt
Color: R: **0**, G: **153**, B: **255**

Selecting the Text Field modifier creates a text field, not a static text box.

3 Draw out your text field over the far right black section of the interface.

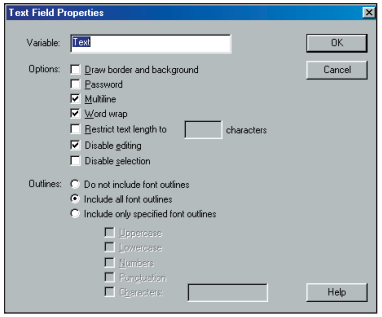
Your field should be approximately 225 pixels high and 113 pixels wide.



Place a text field over the large black area on the right side of the background.

4 Select the Arrow tool, select your text field, open the Text Field Properties dialog box, and set the following properties:

Variable:	**Text**
Selected Options:	Multiline
	Word Wrap
	Disable Editing
	Include All Font Outlines



Edit the properties of the text field as shown.

Working with the View Port

The Product Catalog relies heavily on the **Load Movie** action. In other examples, you loaded additional .swf files into different levels, creating a stack of movies. You can also replace any Movie Clip target with the loaded movie, which is precisely what you'll be doing here. The View Port consists of a Movie Clip symbol containing a mask layer, on which you use **Load Movie** to replace the masked object (another Movie Clip) with the currently selected picture. Sound confusing? Don't worry, you'll get the hang of it.

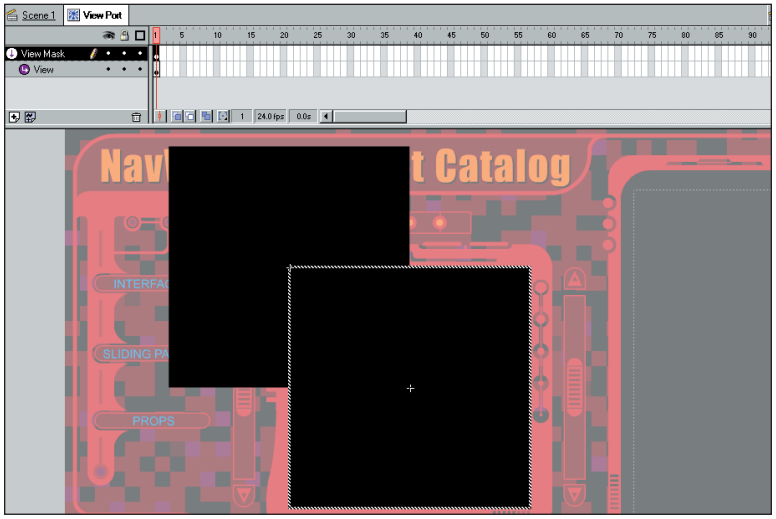
1 Select the View Port layer, drag an instance of the View Port Movie Clip from the Movie Clips folder onto the Stage, position the instance at x: **149.0,** y: **94.2**, double-click the instance, and name the instance **ViewPort**.

2 Use Edit in Place to access the contents of ViewPort.

Inside, notice the two displaced black squares. The top left square is the symbol that will be replaced by the loaded pictures. It's a peculiarity of Flash that the loaded movie inherits the scale, position, and rotation of whatever symbol it's replacing; hence, the View Placeholder's center point is the top left corner of the masking square.

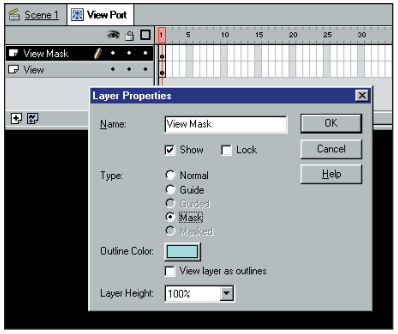
3 Double-click View Placeholder and name the instance **View**.

The point of using mask layers here is that the View instance can be zoomed and scrolled. By having a square mask, you can constrain the visible portion of the current picture.



Use Edit in Place to view the contents of the ViewPort instance.

4 To enable the masking, select the View Mask layer, double-click the View Mask layer symbol to edit its Layer Properties, and set the Type property to Mask.

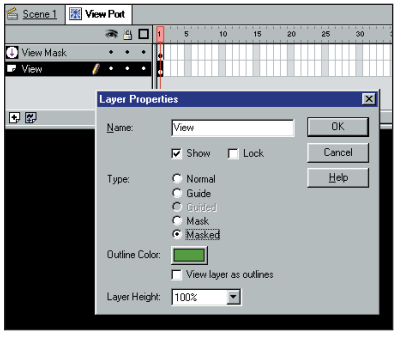


Use Layer Properties to change the View Mask layer's type to Mask.

5 Select the View layer, edit the Layer properties of the View layer, and set Type to Masked.

This completes the masking effect. Now only the parts of View covered by the View Mask will be visible to the user.

6 Exit symbol-editing mode.

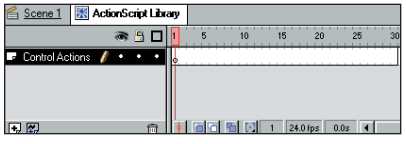


Use Layer Properties to change the View layer Type to Masked.

BUILDING THE ACTIONSCRIPT LIBRARY

In this application, the ActionScript Library symbol contains three functions: Initialize, AdjustView, and LoadView. You've already set up the **Call** statements for these elsewhere in the movie, so type them in now—you're nearly finished.

1 Select the ActionScript Library layer, ensure that no objects are selected on the Stage, and create a new Movie Clip symbol, naming it **ActionScript Library**.



Name the layer in the ActionScript Library Movie Clip and add 29 frames to the layer.

2 In the symbol-editing mode of the ActionScript Library Movie Clip, name the existing layer **Control Actions** and insert an additional 29 frames, creating a total of 30.

3 Select keyframe 1, label the frame **Initialize**, open the Properties dialog box, and assign the ActionScript.

The Initialize script sets a number of key variables, including the current product, feature, and variant. It also loads the appropriate text and resets the text scroller before calling the LoadView and AdjustView keyframes.

```
Stop
FS Command ("AllowScale", False)
Set Variable: "/:Product" = "flashpack1"
Set Variable: "/:Feature" = "interface"
Set Variable: "/:Variant" = "1"
Set Variable: "/:Text" = /:FlashPack1Text
Set Property ("/TextScroll/ScrollHandle", X Position) = 0
Call ("LoadView")
Call ("AdjustView")
```

4 Insert a blank keyframe at frame 11, label the frame **AdjustView**, open the Properties dialog box, and, on the Actions tab, assign the ActionScript.

AdjustView takes scroll-handle positioning information from each of the four Scrollbars in turn and sets the properties of **View** and **Text** accordingly. By getting the information from all four each time, you eliminate the need for separate scripts.

The next step is the grand finale—**LoadView**. **LoadView** is the simplest of the functions. It simply assembles a filename based on the current product, feature, and variant values and then loads the file into the \ViewPort\View location, thereby replacing the previous picture with the current one.

```
If (GetProperty("/Zoom/ScrollHandle",_x) > 0)
    Set Property ("/ViewPort/View", X Scale) = GetProperty("/Zoom/ScrollHandle",_x) * 2.5
    Set Property ("/ViewPort/View", Y Scale) = GetProperty("/Zoom/ScrollHandle",_x) * 2.5
Else
    Set Property ("/ViewPort/View", X Scale) = 5
    Set Property ("/ViewPort/View", Y Scale) = 5
End If
Set Property ("/ViewPort/View", X Position) = 80 - ((GetProperty("/ViewPort/View",
        _xscale) / 100) * (GetProperty("/ScrollX/ScrollHandle",_x) * 2))
Set Property ("/ViewPort/View", Y Position) = 80 - ((GetProperty("/ViewPort/View",
        _xscale) / 100) * (GetProperty("/ScrollY/ScrollHandle",_x) * 2))
Set Variable: "/:Text.scroll" = Int((/:Text.maxscroll / 80) *
GetProperty("/TextScroll/ScrollHandle",_x))
```

5 Insert a blank keyframe at frame 21, label the frame **LoadView**, open the Properties dialog box, and assign the ActionScript.

```
Load Movie (/:Product & "_" & /:Feature & /:Variant & ".swf", "/ViewPort/View")
```

How It Works

The Product Catalog works in two ways. First, it has a series of buttons that set the current product, feature, and variant. After storing this information in variables, the buttons cause the appropriate file to be loaded into the main viewer.

The main viewing area is simply a masked area containing the picture of the current product. By using a mask layer, you make only a set area visible to the user.

Last, there are the scrollbars. Each of these acts as a trigger, causing the picture and text views to be updated based on the positions of all four scroll handles. By capturing the position data of the scroll handles, it's possible to set the zoom, horizontal, and vertical scroll of the picture and vertical scrolling of the text.

Modifications

Well, that's the Product Catalog. But wait! There's one last thing left to explain: the picture files. At the beginning of this project, you made copies of a bunch of .swf files. These are the additional movies loaded by this particular example of the Product Catalog. Notice the naming scheme: product_featurevariant.swf. To build your own custom catalog, create a similar set of files using product and feature names appropriate to your situation. For example, if your first product were named Eggs and its first feature named Color, your corresponding .swf filenames for that product and feature would be

Eggs_Color1.swf
Eggs_Color2.swf
Eggs_Color3.swf

This naming information is set by the product, feature, and variant buttons and is a snap to change, making the Product Catalog a very flexible and versatile application indeed. Plus, because the pictures are downloaded "on demand" rather than all at once, the Product Catalog is perfect for delivery over slower connections.