Game Builders' Club
Guide to ActionScript Programming

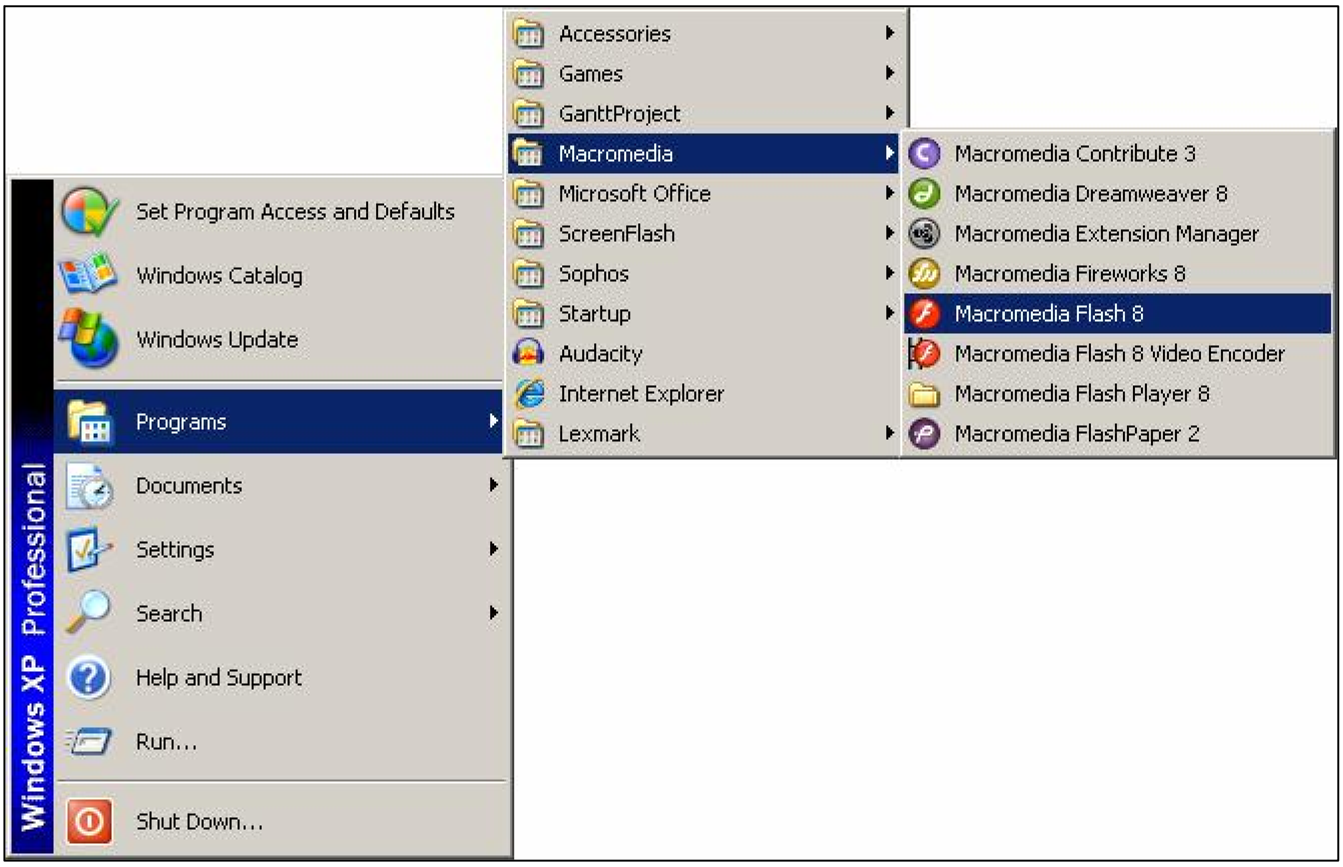
Contents

1. Using Flash 8

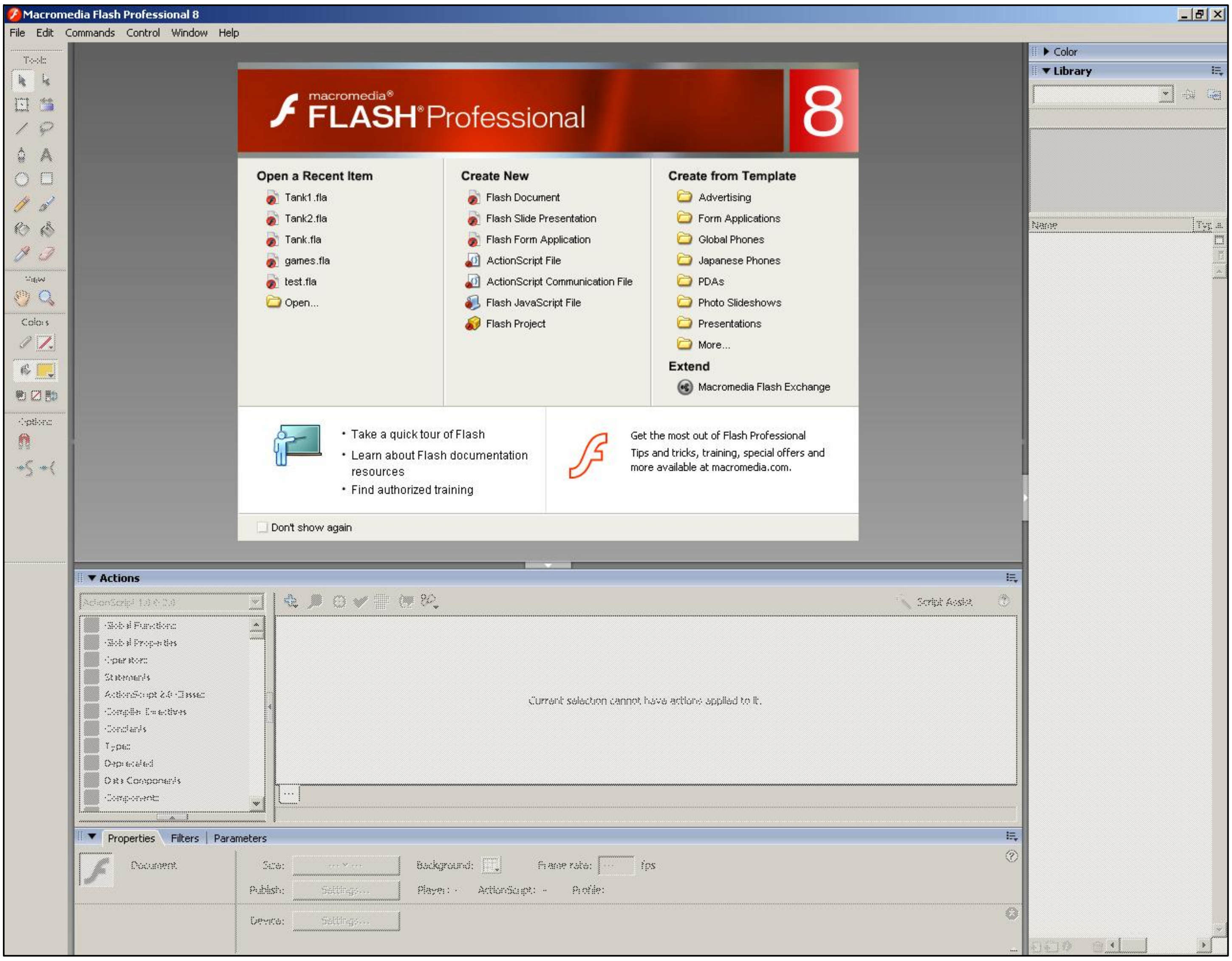
Whether you have used Flash 8 before or not, this guide will go through the basics of using the program to create vivid and interactive games.

The Flash 8 program is a very useful and dynamic tool. Learning to use it can be very simple, but if you have programmed using other systems before, put what you know about them to the back of your mind. While Flash uses many of the same programming concepts as C++, JavaScript and PHP, it also has its own array of unique capabilities.

To begin, please start the Flash 8 program. The icon for it may be on your desktop, or in your *Start -> Programs -> Macromedia* menu.



Once Flash 8 has finished loading, you should see a screen which looks a bit like this:



In the top left corner of the screen you should be able to see the File, Edit, Commands, Control, Window and Help menus.

Below this is the Tool Bar. The tools in this bar will allow you to draw shapes and use them in your Movie Clips and Buttons.

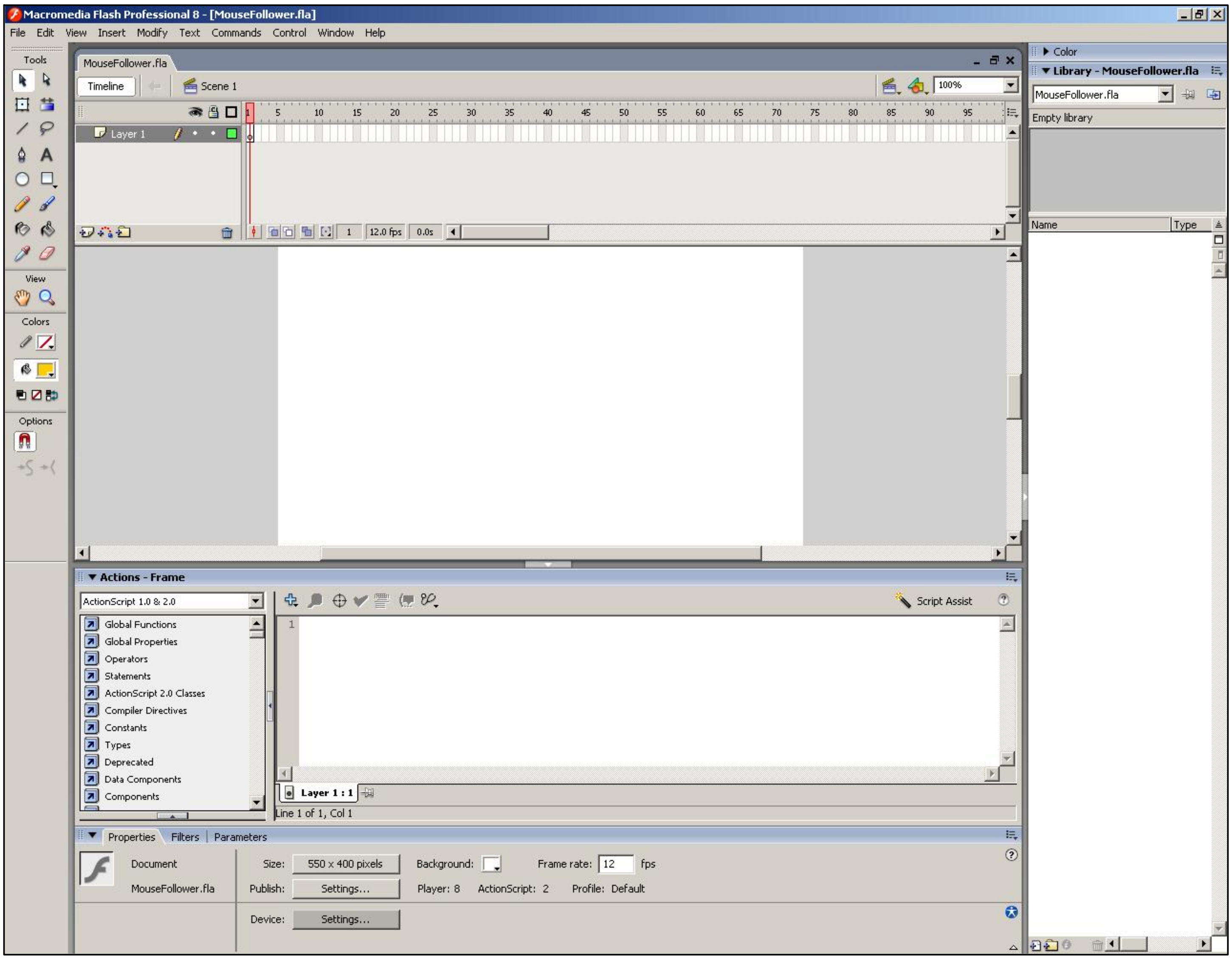
To the right you should be able to see the Library. This is the place you will use to store all of your Movie Clips, Buttons, Bitmaps, Sounds and other resources.

At the bottom of the window, between the Tool Bar and the Library is the Actions and Properties panes. The Actions pane is where you will type in your ActionScript. The Properties pane allows you to check and set the properties of various Movie Clips and Buttons.

To begin a new project, click on *File -> New...* or press Ctrl+N on your keyboard. The New Project dialogue box should appear. Make sure Flash Document is selected, and click OK.

The pane above the Actions pane should now contain a Timeline pane and the Stage. Click on *File -> Save As...* or press Ctrl+Shift+S and give your project a name. Be sure to save the project in your “My Documents” folder for safe storage.

This example uses MouseFollower as the project name. The screen should look like this:



ActionScript 1.0 & 2.0

Line 1 of 1, Col 1

Document: MouseFollower.fla

Size: 550 x 400 pixels

Background:

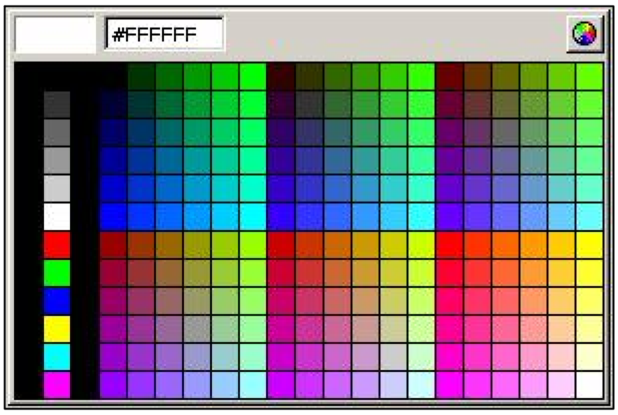
Frame rate: 12 fps

Publish: Settings...

Player: 8 ActionScript: 2 Profile: Default

Device: Settings...

The white rectangle in the middle is the Stage. Above this is the Timeline which contains one Layer. Inside this Layer is a Blank Keyframe (the white rectangle with a circle at the bottom). We will look into the Timeline in the next lesson.



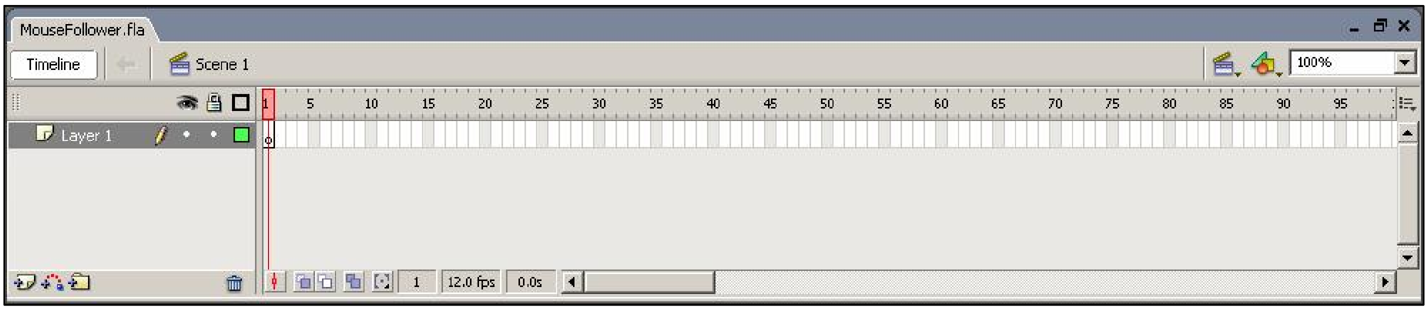
#FFFFFF

For now, please set the Size of your stage by clicking on the Size: button in the Properties pane. The smaller the stage is, the faster you can run the game, but the harder it will be to see what is going on. I have chosen 640 by 512 pixels. Next you may set the Background colour. Click on the small white rectangle next to the word “Background:”. A colour table will open. Clicking on one of these colours will set the background colour. If you wish to use a colour not on the table, please enter the six digit hexadecimal code for your desired colour into the text box above the table.

Finally set the Frame Rate. Do this by entering a number into the “Frame rate:” box. The default is 12fps, which is quite slow. Many programmers prefer 25fps which is fast enough, and doesn’t demand a high powered computer to run it.
Now it is time to save your project. Click on *File -> Save* or press Ctrl+S.

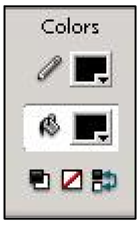
2. Timeline and Layers

The Timeline is used by Flash animators to control the rate of animation. If you have set the Frame Rate to 25fps, then each block of 25 frames represents one second of animation.



The default layer, Layer 1 contains a single Blank Keyframe. A Blank Keyframe is always white.

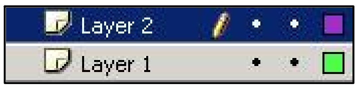
Now go to the Toolbox and click on the Rectangle Tool.

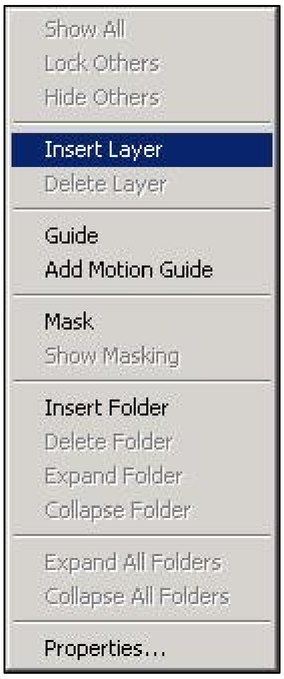


Further down in the Toolbox you should be able to see some colour settings, titled “Colors”. The top colour is the outline colour of the rectangle, while the lower colour is the fill colour. If you can see a white box with a red diagonal line, that means that the colour has been set to transparent. Set these colours to something which contrasts with the background colour.

Next draw a rectangle by clicking and dragging on the stage. You can set various properties for this rectangle in the Properties pane.

Finally, check the Keyframe in Layer 1. It should now be grey. This is no longer a Blank Keyframe.



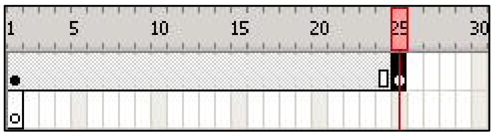


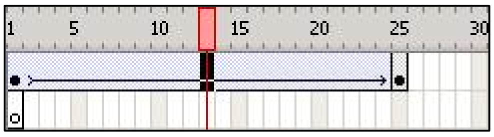
Now create a new layer by right-clicking on the text “Layer 1” and selecting “Insert Layer”. Flash 8 will now create a layer called “Layer 2”. You may rename your layers by double-clicking on their names. Notice that Layer 2 also contains a Blank Keyframe.

When working with Layers, it is important to remember that the last Layer you clicked on is the one you are working with. Click on Layer 2, and create another rectangle which looks different to the first rectangle.

Try moving the rectangles around. Notice that the second rectangle is always on top of the first rectangle if there is any overlap. To change this behaviour you can re-arrange the layers in the list by dragging them with your mouse. The topmost layer in the list always overlaps all of the others.

In either (or both layers), right-click on the timeline at frame 25, and choose “Insert Keyframe” from the list. Notice that the red marker in the timeline has now moved to frame 25. With the marker at frame 25, move the contents of your chosen layer. Then right-click on any frame between 1 and 25 on your chosen layer, and select “Create Motion Tween”. This will position the rectangle in your chosen layer at various points depending on which frame you are looking at.





Next save your project, and then publish it. To publish your project, click on *File -> Publish* or press Shift+F12. Navigate to your project file, and look for a .swf file. This should have an icon which looks like a piece of paper with a white circle on it. Open this file, and you should see the motion tween you have just created.

In order to view or test your project at various points during its development, you must publish it.

3. Movie Clips and Buttons

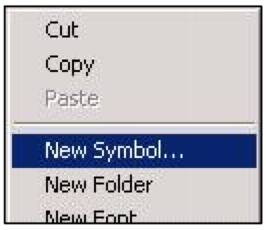
When working on items in the Flash 8 stage, you will usually want to work with either Movie Clips or Buttons.

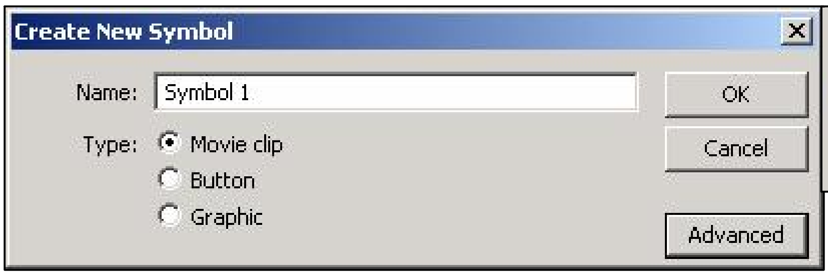
A Movie Clip is usually a small item based on a bitmap, vector, or series of frames. It is possible to create a small flash movie, publish it, and then import it into a larger flash movie in order to create an animated sprite.

A Button is a convenient way to make something happen using ActionScript. They cannot be animated as easily as Movie Clips, but you may still wish to use them.

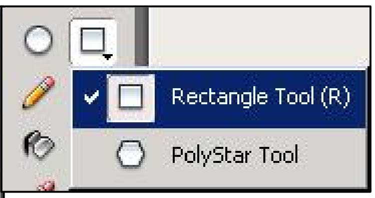
Before you begin, start a new project. You should now have an empty stage to work with.

In order to create a Movie Clip, right-click on the Library, and choose New Symbol. Give the Symbol a name, and set the Type to Movie Clip. The stage will now be replaced with the Symbol Editor pane. Use the Line Tool, Oval Tool, Rectangle Tool and PolyStar Tool to create some shapes.



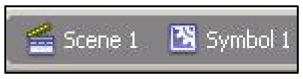


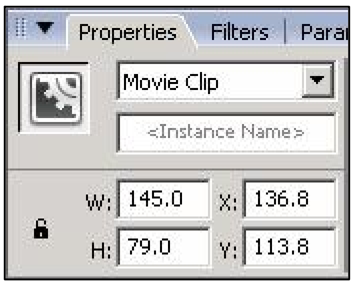
Name: Symbol 1
Type: [✓ Movie clip] Button Graphic



Note that the Symbol Editor has it’s own Timeline and Layers. If you create an animation lasting five frames, it will repeat itself every five frames in the main Movie once it has been published.

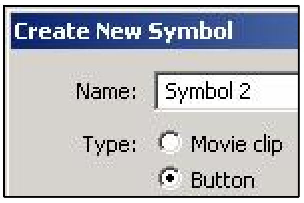
When you have finished drawing your movie clip, click on the word “Scene 1” above the timeline. This will bring you back to the main Stage. Your newly created Movie Clip will be listed in the Library. To bring it onto the stage, simply drag it from the library onto the stage.





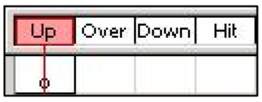
W: 145.0 X: 136.8
H: 79.0 Y: 113.8

When a Movie Clip has been dragged from the Library to the Stage, an “instance” of that Movie Clip has been created. Each instance of a Movie Clip can be given an Instance Name in the Properties pane. ActionScript uses these names when referencing Movie Clips, so if you wish for your ActionScript to move, rotate or do anything to a Movie Clip, be sure to give it a name. Be sure not to have any spaces in your instance names to avoid complications.

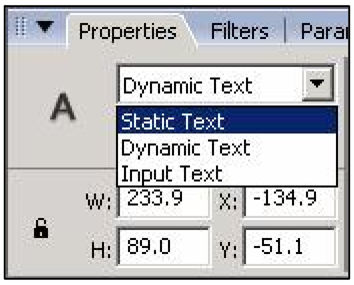


Name: Symbol 2
Type: Movie clip [✓ Button]

To create a Button, right-click on the Library, and choose New Symbol. Give the Symbol a name, and set the Type to Button. The stage will now be replaced with the Symbol Editor pane. Again, use the tools to create a button. While the Timeline for a button has Layers, it also has four main frames – Up, Over, Down and Hit.



The contents of the Up frame will determine how the button will look when the mouse is not over the button.
The contents of the Over frame will determine how the button will look when the mouse is over the button but not clicked on it.
The contents of the Down frame will determine how the button will look when the mouse is over the button and is clicked on it.
The contents of the Hit frame are invisible, but if there is anything drawn in here, the mouse will register as Over if it passes over it.

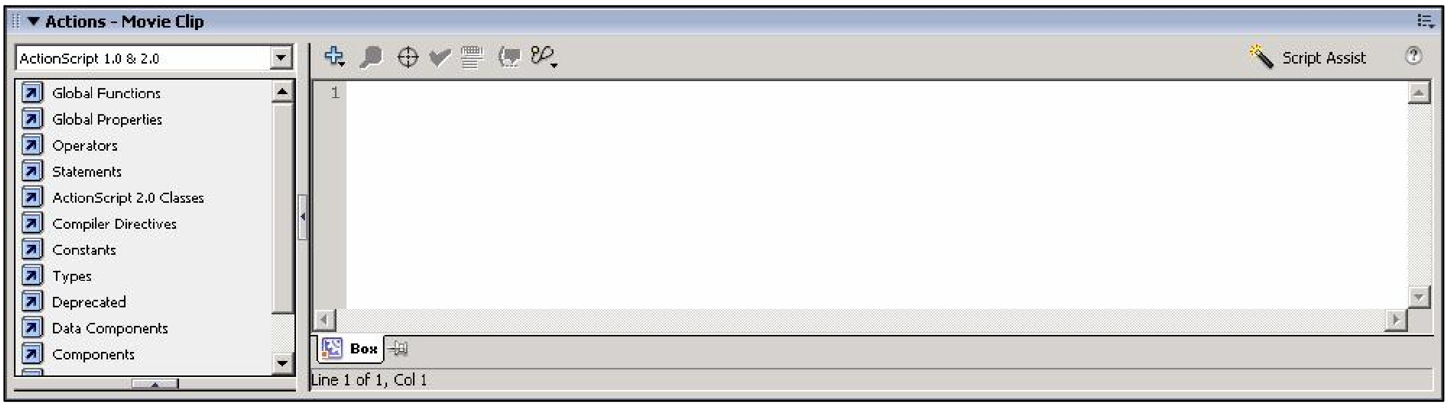


When adding text to a button, make sure the text type is set to "Static Text". This ensures that the user can click on the button even when the mouse is over the text.

Any Actions applied to a button will occur when the button is clicked. This makes buttons useful for testing as well as in the final production.

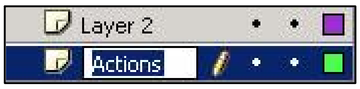
4. Introduction to ActionScript

Various objects in the Flash 8 programming environment can have Actions applied to them. These Actions are fragments of code written in a language called ActionScript.



Some objects or selections of objects cannot have Actions applied to them. To apply an Action to an object, you must enter the ActionScript code into the Actions pane.

For the purposes of this club, we will only apply Actions to Movie Clips, Buttons and the Stage.

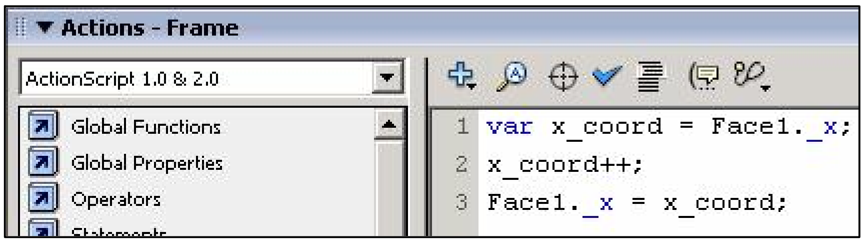


To begin this exercise, start a new project, and create a new Layer called "Actions". Make sure no objects are placed into this layer. Then add a second frame to each layer, and convert the second frame in the "Actions" layer into a Blank Keyframe.

Next, make a note of the names you have given to the instances of any Movie Clips you have on your stage. For this example, I will use the name "Face1". Put your Movie Clip to the left of your stage.

Select the first frame of the empty “Actions” Layer and click on the Actions pane, and type in the following code:

```
var x_coord = Face1._x;
x_coord++;
Face1._x = x_coord;
```



```
var x_coord = Face1._x;
x_coord++;
Face1._x = x_coord;
```

Then select the second frame in the “Actions” Layer and type the following code into the Actions pane:

```
gotoAndPlay(1);
```

Now save your project and publish it before viewing the created .swf file. The object named Face1 should move across the stage from left to right and then disappear.

The first line of code tells the Flash Player to create a new variable called “x_coord” and give it the value of Face1._x, which is the horizontal position of Face1 on the stage.
The second line of code increments, or adds one to the value of x_coord – so if x_coord was 200 to start with, it will be 201 when this line of code is processed.
The third line sets the value of Face1._x to that of x_coord.

Once the code in the first frame has been run, the frame remains visible until the next frame is to be displayed. The code in the second frame simply tells the Flash Player to go back to the first frame and play the movie from there.

The higher the value of Face1._x, the further to the right it moves. There is also a value for the vertical positioning of Face1, which is Face1._y . Experiment with this.

If you want to play a little more with your Movie Clip, use the variable _rotation to rotate the Movie Clip (for the example with Face1, this would become Face1._rotation).

Now that you have seen how easy it is to add ActionScript to your project, we will use it in further chapters. Advanced ActionScript will be covered in chapter 10.

Always remember that ActionScript is case sensitive - it sees a difference between upper case and lower case letters, so be careful when creating or copying code.

5. Rotating and Positioning

When adding Movie Clips to your stage, it may be desirable to rotate the objects at various stages. Doing this with the Free Transform Tool is very easy, but the ability to control rotation using the mouse or keyboard is often a must in any Flash game.

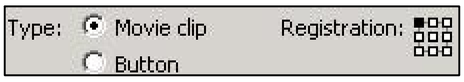
The _rotation variable, as discussed in the previous chapter, can be used to set the rotation of the Movie Clip being controlled.

When setting the _rotation variable, you must enter the desired angle in degrees – that is a number in the range of 0 to 359.

Imagine you have created a Movie Clip in the shape of an upward pointing arrow. When the _rotation variable is set to 0, the arrow should still point strait up. Setting the _rotation to 90 should turn the arrow clockwise by 90 degrees, pointing it towards the right-hand side of the stage. Experiment with different values for _rotation.

As well as rotating Movie Clips, you may also wish to position them. As we saw in the previous chapter, the _x and _y variables control the on-stage location of the Movie Clip. On a Flash Movie stage, the left-hand edge is 0 for the _x variable, and the top edge is 0 for the _y variable.

By default, the top-left of a Movie Clip is its origin point. You can change this in the Symbol Editor, as it may be desirable to position the origin towards the centre of the Movie Clip.


Type: [✓ Movie clip] Button
Registration:

The following example explains a simple mouse follower. Copy the project from the previous chapter, remove the ActionScript from the first frame, and create an upward pointing arrow Movie Clip, and create an instance of it called “Arrow1”, and position it towards the middle of the stage. In the Actions pane on the first frame of the “Actions” Layer, type in the following code:

```
1    onEnterFrame = function (){
2         Arrow1._x = Stage.width * 0.5;
3         Arrow1._y = Stage.height * 0.5;
4         var rAngle = Math.atan2(_root._xmouse - Arrow1._x , Arrow1._y -
          _root._ymouse);
5         var dAngle = rAngle * 180 / Math.PI;
6         Arrow1._rotation = dAngle;
7    }
```

Save and Publish your movie, and see what happens.
If you have typed the above code into the Actions pane correctly, the arrow should point towards the mouse cursor.

The first line tells the Flash player to run a programmer defined function when the frame is played. Because the Action of the next frame tells the player to go back to this frame, the function is called each time the frame is played.

Lines 2 and 3 work out how large the stage is by its width and height, and divides the measurements by two. The results are then passed on to the horizontal and vertical positions of the “Arrow1” Movie Clip, to make sure that it is in the middle of the Movie. Note that I have multiplied them by 0.5 - this does the same job as dividing by two, but computers handle multiplications faster than they can divisions.

Line 4 creates a variable called “rAngle”, which will store an angle measured in radians. Engineers, programmers and scientists use radians instead of degrees. There are approximately 3.141 radians in 180 degrees. The atan2 function performs the trigonometric function “inverse tan”. The great thing about programming is you don’t have to do complex calculations, you just need to know which one to use in your programs.

Line 5 creates a variable called “dAngle” and converts the radians calculated by the atan2 function into degrees.

Line 6 applies this number of degrees to the rotation of the “Arrow1” Movie Clip.

Finally, line 7 closes the function. This “closing” curly brace tells the Flash player where the function ends.

Do not let the complexity of the mathematics put you off - after a little practice, you will understand how and why it works, and you will have a practical use for all those strange things your maths teachers have been telling you about.

6. Inputs

Now that you have seen the basics of moving things round automatically, you will probably want to know how to use the mouse and keyboard to add input to your game.

Starting with the mouse, the location of the cursor in your Movie is stored as two properties of the _root object. Note that the _root object is the lowest level object of any Flash Movie. Everything stems from the root. In the previous example, we used _root._xmouse and _root._ymouse. As you can probably guess, _root._xmouse is the horizontal location of the cursor, and _root._ymouse is the vertical location of the cursor.

Next we have the mouse buttons. The Flash player can easily read the left mouse button, but programming for other buttons and the mouse wheel is a little more complicated. For the time being, we will create programs for single button mice. Other buttons will be examined in the Advanced ActionScript chapter. To check for the action of the mouse button, create a new project, create a wide text box with the Instance Name “debug” and add the following ActionScript to the frame:

```
1     onMouseDown = function(){
2          debug.text="Mouse button is down!";
3          updateAfterEvent();
4     }
5
6     onMouseUp = function(){
7          debug.text="Mouse button is up!";
8          updateAfterEvent();
9     }
10
11    onMouseMove = function(){
12         debug.text="Mouse has been moved!";
13         updateAfterEvent();
14    }
```

Publish the Movie, and try moving the mouse and clicking on it.
Each function in this group of three is quite similar, so I will only explain the first one in detail.

Line 1 calls the custom function which contains lines 2 and 3 each time the left mouse button is pressed down.
Line 2 sends the text between the double inverted commas to the text box called “debug”.
Line 3 is a pre-defined function which updates the output each time the “MouseDown” event occurs. This is needed because an event is something which happens for only a fraction of a second.

You can apply any of these to individual Movie Clips by adding the Instance Name of the Movie Clip followed by a full stop before the event name, for example:

```
1     Arrow1.onMouseDown = function(){
```

This would allow you to check if the mouse button has been pressed over the Movie Clip “Arrow1”.

Finally we come to the keyboard. Take your previous project, save it under a different file name and delete all of the ActionScript, and replace it with this:

```
1    var ASCII_KEY_CODE=65;
2    if(Key.isDown(ASCII_KEY_CODE)){
3         debug.text="Key has been pressed!";
4    }else{
5         debug.text="Key has not been pressed!";
6    }
```

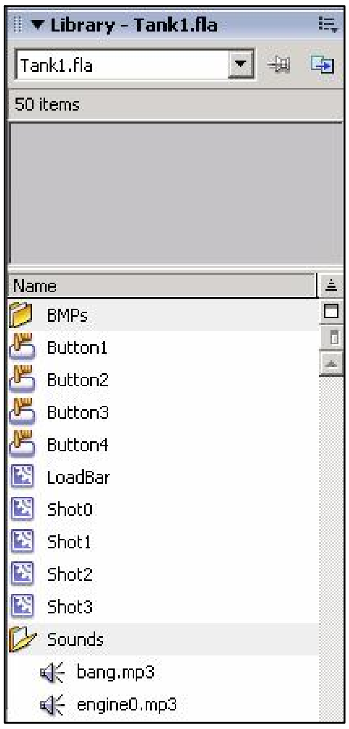
Line 1 creates a new variable which holds the value 65.
Line 2 checks the status of key 65. Usually Key 65 is the letter "A" on most keyboards.
Line 3 sends a message to the debug text box if the "A" key is currently held down.
Line 5 sends a different message to the debug text box if the "A" key is not held down.
Line 6 ends the IF statement.

A series of IF statements each checking different key codes may be used to keep tabs on the various keys used by a Flash game.

Note that different keyboard layouts may provide different results. Try to compare a standard Windows, Mac and Laptop keyboard if you have the opportunity.

A full list of virtual and ASCII key codes is provided in the Advanced ActionScript chapter.

7. Using the Library



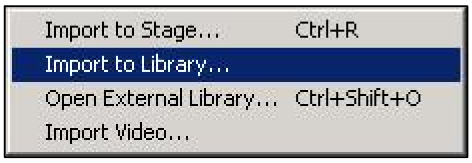
When working with Flash 8, a useful feature is the Library. The Library is a central storage point for every Movie Clip, Button, sound effect, and almost anything else you can imagine. The main advantage of the Library is that it saves space when saving and publishing your project.

Imagine you have a 50KB image of a chair, and you import this image into the Library. If you copy this image onto the stage, it is still 50KB. If you make another copy of this image on the stage, it is still 50KB. You could have hundreds of identical chairs all copied from one Library object, and they would still all total 50KB!

To start using the Library, left click on the empty space in the Library pane, and choose "Create New Symbol". Give it a name and select "Movie Clip" from the list of behaviours. The blank Movie Clip will now be added to the Library, and the Stage pane will be replaced by the Symbol Editor pane.
Use the tools and timeline to create a layered image. It could be anything you like. When you have finished, click on the word "Scene 1" above the timeline to return to the stage.

The Movie Clip you have just created is now in the Library and you may drag it into the Stage. If you drag it onto the stage more than once, you may give each instance of your Movie Clip a unique name (in the Properties pane) and control each one individually using ActionScript.

Now find an image from the internet, and save it in your document storage area. Next click on *File -> Import -> Import to Library...* , browse to the image file you have downloaded and click Open. The image file will now be added to the library.


Import to Stage... Ctrl+R
Import to Library...
Open External Library... Ctrl+Shift+O
Import Video...

To convert the newly added image into a Movie Clip, drag it into the Stage, right-click on it and choose Convert to Symbol. Give the Symbol a name and make sure the Movie Clip behaviour is selected.

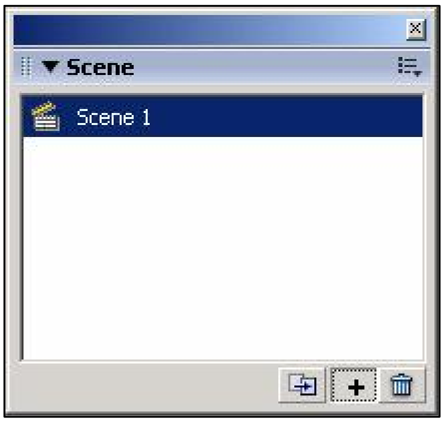
Whatever you do, don't delete the imported image, otherwise the newly created Movie Clip will become invisible on the stage.

Using the Import to Library method, you can also add animated images and sound files to the Library for use in your Movie. Note that any timings in animated GIF and PNG files will be lost on import, but you may edit them in the Symbol Editor to obtain the desired frame rate.

8. Sound

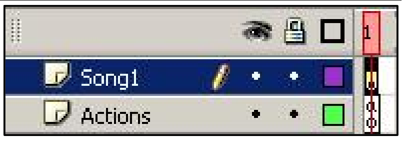
Flash can be used to apply various effects to sounds. We will focus on starting and stopping them, as well as controlling the volume and stereo balance.

Before you can use a sound, you must import it into the Library. Once it is there, right click on it and choose Linkage... Next you may enter an Identifier name, which I recommend you make unique but short. The Identifier will be used by the ActionScript to refer to this noise. Make sure "Export for ActionScript" is enabled, and disable "Export in first frame".

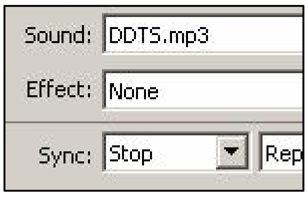


Because we have chosen not to export the sounds in the first frame, we must create some "decoy" frames to make sure the sounds are exported. Press Shift+F2 to open the Scene window, and click on the Add (+) button to create a new scene. Give the new scene a name, such as "Sounds" and move it above the main scene, which by default is called "Scene 1". Click on this new scene, and you will be given a blank Stage and Timeline. Rename the only Layer in the Timeline as "Actions" and type the following ActionScript into the Actions pane:

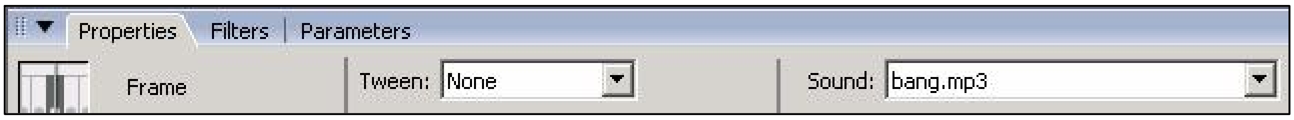
```
gotoAndPlay("Scene 1",1);
```



Now create a new Layer for each sound you wish to include in your finished Movie. Give each Layer the same name as each sound, and drag each sound from the Library into stage with the only Frame in each Layer. A line will appear in each frame once the sound has been successfully added, and you may confirm the correct sound is in the correct layer by checking the Sound: option in the Properties pane. Set the Sync: option to "Stop" on each sound. Then click on "Scene 1" in the Scene window to return to the main project.



Sound: DDTS.mp3
Effect: None
Sync: Stop



Properties | Filters | Parameters
Frame
Tween: None
Sound: bang.mp3

Next close the Scene window and create a Movie Clip in the Library. It doesn't need to be complicated - a square or circle will do. Drag it onto the stage, name it, and create a separate Layer for your ActionScript.

Now type the following ActionScript into the first frame of your Actions Layer:

```
_root.bSound=new Sound(Square1);
bSound.attachSound("Song1");
bSound.start(0,1000);
```

This example assumes the name of the Movie Clip is "Square1" and the name of the sound in the Library is "Song1".

Now create two new blank Frames for each Layer, and convert every Frame in the "Actions" Layer into Blank Keyframes.

In Frame 2 of the "Actions" Layer add the following ActionScript:

```
var swidth=Stage.width;
var sheight=Stage.height;
var bVolume=100-Math.floor(Square1._y/sheight*100);
var bPan=Math.floor(Square1._x/swidth*200)-100;
bSound.setVolume(bVolume);
bSound.setPan(bPan);
```

In Frame 3 of the "Actions" Layer, add the following ActionScript:

```
gotoAndPlay(2);
```

Next make sure the Movie Clip on the Stage is selected by clicking on it once, and enter the following ActionScript into the Actions pane:

```
on(press){
     startDrag(this);
}
on(release){
     stopDrag();
}
```

Finally, save and publish your project, then open the created .swf file.
Assuming everything has been typed in correctly, you should be able to drag your Movie Clip around the Stage using your mouse. Observe how the movements alter the sound.

Moving the Movie Clip up should make it louder, and moving it down should make it quieter.
If you have stereo speakers, moving the Movie Clip to the left should make it louder in the left speaker, and moving it to the right should make it louder in the right speaker.

The ActionScript in the first Frame creates a new Sound Object called "bSound" and connects it to the Movie Clip named "Square1".
Next it associates the sound named "Song1" to "bSound".
The final command in this block plays the "bSound" Sound Object 1000 times after an initial offset of 0 seconds.

The ActionScript in the second Frame works out the size of the Stage.
Line 3 finds out how far down the Stage the “Square1” Movie Clip is, and assigns the result to a new variable called “bVolume”. If the Movie Clip is near the bottom of the Stage, the value of “bVolume” approaches 0, and if it is near the top, the value approaches 100.
Line 4 finds out how far across the Stage the “Square1” Movie Clip is, and assigns the result to a new variable called “bPan”. If the Movie Clip is near the left of the Stage, the value of “bPan” approaches -100, and if it is near the right, the value approaches 100.
Line 5 applies the value of “bVolume” to the volume parameter of “bSound”.
Line 6 applies the value of “bPan” to the pan parameter of “bPan”.

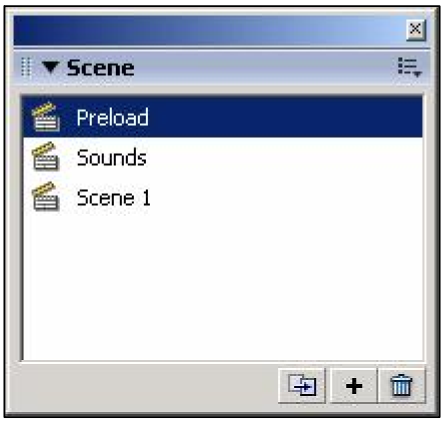
The ActionScript in the third Frame simply tells the Flash to repeat the second Frame.

Finally, the ActionScript in the “Square1” Movie Clip allows the user to drag “Square1” around the stage. The parameter called “this” in the startDrag command tells the ActionScript to start to drag the Object containing it - in this case, it is the “Square1” Movie Clip.

An important thing to remember with Flash is that it is unable to play more than eight sounds at the same time, so if you have that many running, calls to start playing another sound will be ignored until one or more of the existing sounds have finished playing.

<u>9. Pre-Loading</u>
A media-heavy Flash Movie with lots of sound and detailed graphics could be several megabytes in size. As a result, the download time for such a Movie would be at least a few seconds on a fast broadband connection, and several minutes for those still using dial-up.
A Flash Movie will, by default, start to play before it has finished loading. If it attempts to play a Frame or sound which has not yet been downloaded, this could cause problems.
To prevent these problems, you can add a “pre-loader” to the start of your Movie.

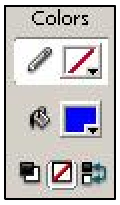
A typical pre-loader will loop until the entire Movie has finished downloading. Some will start the Movie automatically, others will invite the user to click on a button to start the Movie. This example will start the Movie automatically.



Open the project we used for the previous chapter and open the Scene window by pressing Shift+F2. Create a new scene and drag it to the top of the list. Name this scene “Preload”. Go to this new scene, and close the Scene window.

Next create a new Layer, and name one Layer “Actions” and the other Layer “Preload”. In each Layer, create a second Frame.

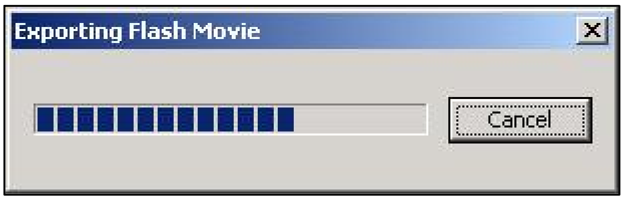
In the first Frame of the “Preload” Layer, create a Text Box and a borderless Rectangle.



In the Properties for the Text Box, set the Instance name to “Percentage” and make sure the drop-down list above is set to “Dynamic Text”.
To create a borderless Rectangle, make sure the outline colour is set to transparent before drawing it. Convert the Rectangle into a Movie Clip.

Set the instance name of the new Movie Clip to “LoadBar”, set the width to 1.0 and the height to 20.0. Convert the second Frame of the “Actions” Layer to a Blank Keyframe. In the Actions pane, enter the following ActionScript:

```
1    var loadedBytes=this.getBytesLoaded();
2    var totalBytes=this.getBytesTotal();
3    var pcent=Math.round(loadedBytes/totalBytes*100);
4    Percentage.text=loadedBytes+" of "+totalBytes+" bytes
     loaded...\n"+pcent+"%";
5    LoadBar._width=pcent;
6    if(loadedBytes==totalBytes){
7         gotoAndPlay("Scene 1",1);
8    }else{
9         gotoAndPlay(1);
10   }
```



To test your pre-loader, save your project and press CTRL+Enter, wait for the Movie to be exported, and press CTRL+Enter again. Flash 8 will now simulate a slow download speed. You should see your pre-loader count up. How fast it counts depends on the size of your published Movie and the download speed being simulated.

Line 1 creates a variable called loadedBytes, and reads how many bytes have been downloaded for the main Movie.
Line 2 creates a variable called totalBytes, and reads how many bytes the whole Movie is.
Line 3 creates a variable called pcent and works out the percentage of bytes downloaded so far based on the data from Lines 1 and 2.
Line 4 presents the data from Lines 1, 2 and 3 in the Text Box called Percentage. The “\n” creates a new line in the Text Box.
Line 5 sets the width of the LoadBar rectangle to match the value of pcent.
Lines 6 to 10 check to see if the Movie has finished loading. If it has, it starts the next scene and continues. If it has not finished, the PreLoader scene goes back to the first Frame and starts again.

10. Advanced ActionScript

ActionScript can be used to compute numbers, manage text, move, rotate and resize objects on the stage, and apply volume effects to sounds.
Please remember that you are not expected to commit this section to memory, but the ability to do so would be both impressive and useful.

Two important programming concepts are variables and arrays. If you have studied algebra, you will already be familiar with variables.
In the common example of y=mx+c, y, m, x and c are all variables. In ActionScript, you may give your variables longer, more meaningful names. An ActionScript variable must begin with an alphabetical character, either upper or lower case, and may contain any alphabetical or numerical character, or an underscore.
To create a variable, simply use the var constructor:

```
var m=2;
var x=30;
var c=5;
var y=m*x+c;
```

However, variables created in this way can only be read in the frame or Movie Clip in which they were created. If you wish to create a global variable, add “_root.” To the beginning of the variable name.

```
var _root.m=2;
var _root.x=30;
var _root.c=5;
var _root.y=m*x+c;
```

Text values as well as numbers may be stored in a variable. A text value in ActionScript is called a String. To create a String Object, use the String Constructor:

```
var name=new String();
name="Tony";
```

If you have a series of variables which represent a series of similar items, such as high scores, you may want to use an array. An array is a special object which has a single variable name, but may contain many variables.

To create an array, use the Array Constructor:

```
var hiScores=new Array(10);
```

This creates a series of 10 variables all with the name "hiScores". To access the data in an array, use square brackets ([and]) after the variable name to select an individual variable:

```
hiScores[0]=100;
hiScores[1]=200;
...
hiScores[9]=1000;
```

Note that the address values of an array always start at 0. If an array has 10 entries, the highest address is 9. To set or read the values of an array, you could use a method similar to the above example. However, for an array with 10 entries, you would need 10 lines of code to enter the data. Using a loop could cut this number of lines down to 3.

```
var hiScores=new Array(10);
for(i=0;i<10;i++){
     hiScores[i]=(i+1)*100;
}
```

This example uses a For Loop. Typically, a For Loop runs a set series of commands for a set number of times. In this case, the value of "i" is used to control the Loop. The initial value of "i" is set to 0. Then each time the loop starts to run, it checks to see if the value of "i" is less than 10. 0 is less than 10, so it processes the commands between the curly braces ({ and }). When it reaches the end, it adds 1 to the value of "i". The next time the Loop runs, the value of "i" is now 1. The next time it runs, it will be 2. The final time it runs, it will be set to 10. It will attempt to run again, but 10 is not less than 10, so the Loop stops running and any code after the Loop is allowed to be processed. There are other types of Loop.

A While Loop checks to see if a condition is met before processing the code it contains, whereas a Do While Loop processes the code before checking to see if a condition is met.

```
while(i<10){
     hiScores[i]=(i+1)*100;
     i++;
}
```

or

```
do{
     hiScores[i]=(i+1)*100;
     i++;
}while(i<10);
```

All three of these examples of Loops do the same thing, but you must usually choose the correct Loop for the job.

One of the most important objects available to ActionScript programmers is the Math object. This object is host to many useful methods and constants. Below are listed some of the more useful of these:

Math.abs(number)	Calculates the absolute value of a given numerical value, eg Math.abs(4) = 4 Math.abs(-4) = 4
Math.acos(number)	Calculates the arc-cosine of a given numerical value.
Math.asin(number)	Calculates the arc-sine of a given numerical value.
Math.atan(number)	Calculates the arc-tangent of a given numerical value.
Math.atan2(y,x)	Calculates the arc-tangent of a pair of given numerical Values where “x” is the horizontal measurement and “y” is the vertical measurement.
Math.ceil(number)	Rounds up a given numerical value.
Math.cos(number)	Calculates the cosine from a numerical value in radians.
Math.floor(number)	Rounds down a given numerical value.
Math.max(x,y)	Decides which of a pair of numerical values is the greatest.
Math.min(x,y)	Decides which of a pair of numerical values is the least.
Math.pow(x,y)	Calculates the result of “x” raised to the power “y”.
Math.random()	Calculates an almost random value which is greater than or equal to 0, and less than 1.
Math.round(number)	Rounds up or down a given numerical value.
Math.sin(number)	Calculates the sine from a given numerical value in radians.
Math.sqrt(number)	Calculates the square root of a given numerical value.
Math.tan(number)	Calculates the tangent from a given numerical value in radians.
Math.PI	The value of PI accurate to approximately 31 decimal places.

As well as these advanced functions, there is a range of mathematical operators available for ActionScript programmers to use:

+	Adds together two values:	5+6=11
-	Subtracts one value from another value:	7-4=3
*	Multiplies two values together:	5*7=35
/	Divides one value by another value:	40/8=5
%	Divides one value by another and finds the remainder:	44%8=4
++	Adds 1 (increments) a value:	5++ = 6
--	Subtracts 1 (decrements) from a value:	5-- = 4

Beyond the traditional range of mathematical operators, there is a range of Boolean logical operators:

==	Checks if two objects are equal:	5==5 is true
!=	Checks if two objects are not equal:	5!=5 is false
&&	Checks if two objects are “true”:	5==5 && 7!=7 is false

||	Checks if one or both of a pair of objects are “true”:	5==6 || 7==7 is true
<	Checks if one value is less than another:	5 < 8 is true
>	Checks if one value is greater than another:	5 > 8 is false
<=	Checks if one value is less than or equal to another:	5 <= 5 is true
>=	Checks if one value is greater than or equal to another:	9 >= 4 is true

The following Bitwise logical operators only work with positive whole numbers:

&	Bitwise AND:	1&11 = 1, 2&11 = 2, 4&11 = 0, 8&11 = 8
|	Bitwise OR:	1|11 = 11, 2|11 = 11, 4|11 = 11, 16|11 = 27
^	Bitwise XOR:	1^11 = 10, 2^11 = 9, 4^11 = 15, 16^11 = 27
~	Bitwise NOT:	~1 = -2, ~2 = -3, ~6 = -7

Text, like numerical values, can be managed in a number of useful ways. To enable this, the String object can be used. Note that any variable holding a text value is a String object and may have most of the following methods applied to them.
The following examples assume the value of “s” to be “Hello!”

String.charAt(number)	Finds the character at a given location: s.charAt(0) = “H” s.charAt(5) = “!”
String.charCodeAt(number)	Finds the ASCII code of a character at a given location.
String.fromCharCode(number)	Converts an ASCII code into a character.
String.indexOf(string)	Finds the first location of a short string in the main string: s.indexOf("lo") = 3 s.indexOf("l") = 2 s.indexOf("x") = -1
String.lastIndexOf(string)	Finds the final location of a given string in the main string: s.lastIndexOf("lo") = 3 s.lastIndexOf("l") = 3 s.lastIndexOf("x") = -1
String.slice(start,end)	Finds the string between the start and end points: s.slice(2,4) = “ll” s.slice(1,5) = “ello” s.slice(3) = “lo!”
String.substr(start,length)	Finds the string starting from the start point for a given length: s.substr(2,4) = “llo!” s.substr(1,3) = “ell”
String.substring(start,end)	Finds the string between the start and end points: s.substring(2,4) = “l” s.substring(1,5) = “llo”
String.toLowerCase()	Converts the string to lowercase: s.toLowerCase() = “hello!”
String.toUpperCase()	Converts the string to uppercase: s.toUpperCase() = “HELLO!”
String.length	Finds the length of a string: s.length = 6

When using ActionScript to control Movie Clips, you may use the following methods and properties:

movie_clip.gotoAndPlay(frame)	Plays the specified Movie Clip from a given frame.
movie_clip.gotoAndStop(frame)	Jumps to and stops the specified Movie Clip at a given frame.
movie_clip.hitTest(symbol)	Checks if the specified Movie Clip is touching or overlapping with the specified symbol.
movie_clip.hitTest(x,y,true)	Checks if the specified Movie Clip is touching or overlapping with the specified x and y coordinates.
movie_clip.play()	Plays the specified Movie Clip from the current frame.
movie_clip.startDrag(false)	Makes the specified Movie Clip draggable.
movie_clip.stop()	Stops the specified Movie Clip at the current frame.
movie_clip.stopDrag()	Stops dragging the current draggable Movie Clip.
movie_clip._height	The height of the specified Movie Clip.
movie_clip._rotation	The angle of rotation of the specified Movie Clip in degrees.
movie_clip._visible	A Boolean value which is true if the Movie Clip is visible, and false if it is not visible.
movie_clip._width	The width of the specified Movie Clip.
movie_clip._x	The horizontal position of the specified Movie Clip.
movie_clip._y	The vertical position of the specified Movie Clip.

Sounds can be controlled using ActionScript in various ways. Most of these concern the volume and duration of the sound. When creating a sound, you may wish to use the following constructor:

```
variable = new Sound(movie_clip)
```

In this example, the word “variable” represents the name of the variable you wish to give to the sound object you want to create. The word “movie_clip” represents the name of the Movie Clip you wish to attach your sound to. If you don’t specify a Movie Clip, any changes to the volume settings of this sound object will affect the global volume settings, and change the volume for all other sound objects.

sound.attachSound(sound_id)	Attaches a sound linked in the Library to the Movie to the specified sound object.
sound.getPan()	Finds the Pan value of the specified sound object.
sound.getVolume()	Finds the Volume value of the specified sound object.
sound.setPan(number)	Sets the Pan value for the specified sound object. A value of -100 sets the volume all the way to the left speaker, a value of 0 sets the volume half way between each speaker and a value of 100 sets the value all the way to the right speaker.
sound.setVolume(number)	Sets the Volume value for the specified sound object. A value of 100 sets the volume to maximum, and a value of 0 almost mutes the

	volume.
sound.start(offset,loop)	Starts playing the attached sound on the specified sound object. The offset value is a numerical value representing how many seconds after the start the sound should be played from - this is not a delay. The loop value is the number of times after the initial start the sound should loop for.
sound.stop(sound_id)	Stops playing the specified sound as identified by its link name in the Library. If no sound_id is specified, all sounds stop playing.
sound.duration	Finds the duration of a sound in milliseconds.
sound.position	Finds the number of milliseconds for which the specified sound object has been playing.
sound.onSoundComplete	An event which applies when a sound object stops playing. An example of how to use this is:

```
function replaySound(snd){
snd.start();
}
sound.onSoundComplete=replaySound(this);
```

Timing specific points of the game or finding the current time and date can often be useful for an ActionScript program.

getTimer()	Finds how long the current Movie has been playing in milliseconds.
date.getDate()	Finds the day of the month from 1 to 31 of the specified date.
date.getDay()	Finds the day of the week from 0 to 6 of the specified date. 0 is Sunday, 1 is Monday, etc
date.getFullYear()	Finds the four digit year of the specified date.
date.getHours()	Finds the hour from 0 to 23 of the specified date.
date.getMilliseconds()	Finds the milliseconds from 0 to 999 of the specified date.
date.getMinutes()	Finds the minutes from 0 to 59 of the specified date.
date.getMonth()	Finds the month from 0 to 11 of the specified date. 0 is January, 1 is February, etc
date.getSeconds()	Finds the seconds from 0 to 59 of the specified date.
date.getTime()	Finds the number of milliseconds since 0:00:00 01/01/1970.

When creating a new date object, you must use the date constructor:

```
now=new Date();
```

or

```
birthday=new Date(year, month, date, hour, minute, second,
millisecond);
```

If you wish to set a specific date, the year, month and date are all required. Other parameters are optional but must be set in the correct order. You cannot set the number of minutes without setting the number of hours.
Note that when setting the month, January is represented by 1, but when getting the month from a specified date, January is represented by 0.

The list of key codes and ASCII codes for a standard Windows UK Keayboard Layout is as follows:

KEY	**CODE**	**ASCII**	**ASCII+Shift**	**ASCII+Caps**	**ASCII+Num**
Backspace	8	8	8	8	8
Tab	9	9	9	9	9
Enter	13	13	13	13	13
Shift	16	0	0	0	0
Ctrl	17	0	0	0	0
Alt	18	0	0	0	0
Caps	20	0	0	0	0
Esc	27	27	27	27	27
Space	32	32	32	32	32
Page Up	33	0	0	0	0
Page Dn	34	0	0	0	0
End	35	0	0	0	0
Home	36	0	0	0	0
Left	37	0	0	0	0
Up	38	0	0	0	0
Right	39	0	0	0	0
Down	40	0	0	0	0
Insert	45	0	0	0	0
Delete	46	127	127	127	127
0	48	48	41	48	48
1	49	49	33	49	49
2	50	50	64	50	50
3	51	51	35	51	51
4	52	52	36	52	52
5	53	53	37	53	53
6	54	54	94	54	54
7	55	55	38	55	55
8	56	56	42	56	56
9	57	57	40	57	57
A	65	97	65	65	97
B	66	98	66	66	98
C	67	99	67	67	99
D	68	100	68	68	100
E	69	101	69	69	101
F	70	102	70	70	102
G	71	103	71	71	103
H	72	104	72	72	104
I	73	105	73	73	105
J	74	106	74	74	106
K	75	107	75	75	107
L	76	108	76	76	108
M	77	109	77	77	109
N	78	110	78	78	110
O	79	111	79	79	111
P	80	112	80	80	112
Q	81	113	81	81	113
R	82	114	82	82	114
S	83	115	83	83	115
T	84	116	84	84	116

KEY	**CODE**	**ASCII**	**ASCII+Shift**	**ASCII+Caps**	**ASCII+Num**
U	85	117	85	85	117
V	86	118	86	86	118
W	87	119	87	87	119
X	88	120	88	88	120
Y	89	121	89	89	121
Z	90	122	90	90	122
Windows	91	0	0	0	0
0(numpad)	45/96	0	0	0	48
1(numpad)	35/97	0	0	0	49
2(numpad)	40/98	0	0	0	50
3(numpad)	34/99	0	0	0	51
4(numpad)	37/100	0	0	0	52
5(numpad)	12/101	0	0	0	53
6(numpad)	39/102	0	0	0	54
7(numpad)	36/103	0	0	0	55
8(numpad)	38/104	0	0	0	56
9(numpad)	33/105	0	0	0	57
*(numpad)	106	42	42	42	42
+(numpad)	107	43	43	43	43
-(numpad)	109	45	45	45	45
. (numpad)	46/110	127	127	127	46
/(numpad)	111	47	47	47	47
F1	112	0	0	0	0
F2	113	0	0	0	0
F3	114	0	0	0	0
F4	115	0	0	0	0
F5	116	0	0	0	0
F6	117	0	0	0	0
F7	118	0	0	0	0
F8	119	0	0	0	0
F9	120	0	0	0	0
F10	121	0	0	0	0
F11	122	0	0	0	0
F12	123	0	0	0	0
Numlock	144	0	0	0	0
ScrollLock	145	0	0	0	0
;:	186	59	58	59	59
+=	187	61	43	61	61
,<	188	44	60	44	44
-_	189	45	95	45	45
.>	190	46	62	46	46
/?	191	47	63	47	47
‘@	192	39	34	39	39
[{	219	91	123	91	91
\ |	220	92	124	92	92
]}	221	93	125	93	93
~#	222	35	35	35	35