Flash 8

Flash Screen (Choose missing panels from Window menu)

Stage

Tools Panel (ctrl+F2)

Properties Panel (ctrl+F3)

Color Mixer/Swatches Panel/Align Panel (shift+F9/ctrl+F9/ctrl+k)

Info/Transform Panel (ctrl+i/cntrl+t)

Library Panel (ctrl+l)

Timeline (Layers and Frames)

Actions Panel (alt+F9)

Output Panel (F2)

Tools and Shortcuts

Selection Tool (v)

Properties: Size (width/height), Background color, Frame rate, and others

Modify ➜ Document brings up a window of this properties

Property Panel changes to reflect the kind of object selected

Double click to choose contiguous pieces

Shift-click to choose several objects

Edit ➜ Select All (ctrl+a) and Edit ➜ Deselect All (shift+ctrl+a)

Oval Tool (o)

Shift drag gives a circle

Drawing same color makes shapes merge together

Drawing different color causes a cut out from bottom shape

Show grid to help drawing

View ➜ Grid ➜ Show Grid (ctrl+')

Snap to grid to help drawing

View ➜ Snapping ➜ Snap to Grid (shift+ctrl+')

Undo (ctrl+z) and redo (ctrl+y) for mistakes

Stroke color (border) and fill color

Border and fill are distinct shapes (double click to select both)

Properties: Change stroke size, shape size, shape position, and colors

Rectangle Tool (r)

Shift drag gives a square

PolyStar Tool: Options (polygon or star) in Properties Panel

Pencil Tool (y)

Shift drag gives 90° lines

Properties: Change stroke size and kind of line

Properties: Set Cap and Join

Options: Straighten and Smooth

Paint bucket Tool (k)

In rectangle and ovals

In drawn figures

Paint bucket may not fill if gaps occur in figure

Ink Bottle Tool (s)

Used to change the color of strokes in the same way the Paint Bucket is used to change the color of fills.

Provides access to the Properties Panel where the stroke style, color, and size can be changed and then the Ink Bottle can be clicked on the stroke of a shape.

Can be used to add a stroke to a shape with no stroke.

Pen Tool (p)

Used to create Bézier curves, named for the mathematician Pierre Bézier.

Polygonal Figures

Alternate clicking and moving the mouse to create a polygon.

Close the polygon by returning to the starting point or by

Double clicking to end the drawing or

Selecting another drawing tool for the Tools Panel to end the drawing.

Curved Figures

Click to place an anchor point on the stage.

Click at another place and drag the mouse to describe an invisible tangent line to the curve being drawn from the anchor point.

(You can start with a drag if you want to establish a starting tangent.)

Clicking and dragging again continues the curve along another tangent line.

Just clicking also continues the curve, but clicking yet again continues the curve in a straight line.

The drawing can be finished by closing the figure or by double clicking.

Visual Help

Show the progress of Pen tool drawings:

Edit ➜ Preferences ➜ Drawing or

Flash Professional ➜ Preferences ➜ Drawings (Macintosh)

Check

Show pen preview (for Pen Tool)
Show solid points
Show precise curves

Editing Bézier Curves

Choose the Subselection tool and alter articulation points, curve segments, or tangent lines or

With the Pen tool selected, hold down the control key and edit the curve.

Pen States and Icons (Pen Tool chosen)

Pen displays a small **x** when it is simply over the stage.

When the Pen hovers over an endpoint, it display an **o** to indicate that is an endpoint.

When the Pen hovers over a corner point, it displays a **-** to indicate that clicking this corner point deletes it.

When the Pen is over a path between two points, it displays a **+** to indicate that clicking here adds point to the path.

When the Pen hovers over an existing point (not a corner), it displays a **^** to indicate that clicking that point turns in into a corner point.

Brush Tool (b)

Shift drag gives 90° lines

Drawing will merge (same color) or cut out (different colors)

Draws with fill color not the stroke color

Line Tool (n)

Shift drag gives 45° lines

Reshape by dragging

Subselection Tool (a)

Shows articulation points on line, brush, and pen drawings (and others)

Free Transform Tool (q)

Select what you want to transform

Use handles in the transform selection to make changes

Use the Transform Panel to make changes

Eraser Tool (e)

Erase any parts of a drawing by dragging eraser

Lasso Tool (l)

Select a figure that cannot be outlined by a rectangle

Option: Polygonal lasso by clicking and double clicking at end

Text Tool (t)

Properties: Font, size, color, style, alignment

Types: Static Text, Dynamic Text, Input Text

Just start typing or

Draw a rectangle to type inside

Eyedropper Tool (i)

Pick up a color to use as a fill with the Paint Bucket

Hand Tool (h)

Adjust the position of the stage

Spacebar provides a temporary hand without changing the tool selection

Zoom Tool (m,z)

Options allow enlarge or reduce

ctrl+= provides a temporary enlarge without changing tool selection

ctrl+- provides a temporary reduce without changing tool selection

Object Drawing (j)

This Options makes each drawn figure a different object (no merge, no cut out).

Drawing Guides

Need to have rulers: View ➜ Rulers (alt+shift+ctrl+R)

Drag guide down from ruler or to the right from ruler

Show guide (default) View ➜ Guides ➜ Show Guides (ctrl+;)

Snap to guides: View ➜ Snapping ➜ Snap to Guides (shift+ctrl+;)

Guides are invisible in movie

Grouping Drawn Objects

Modify ➜ Group (ctrl+g) and Modify ➜ Ungroup (shift+ctrl+g)

Make a duplicate of a drawn object: Edit ➜ Duplicate (ctrl+d)

Playing a Movie

Play	Enter
Advance one frame	.
Move back one frame	,
Create *swf* file and display movie	ctrl+Enter
Publish (create *swf* and *html*)	File ➜ Publish (ctrl+F12)

Layers

Shapes and figures drawn on separate layers do not merge or cut out.

A layer may contain:
- a drawing
- a sound
- ActionScript (code)
- an animation

Layers should be given descriptive names (double click on layer name, type a new name, and Enter).

Add a layer

Insert ➜ Timeline ➜ Layer or

Press button on left below layer window

Active layer is highlighted with a pencil.

Choosing a layer selects all the objects on that layer.

Selecting an object on the stage may change the active layer.

Three States of a Layer

show or hide (eye icon)

lock or unlock (padlock icon)

display objects as outlines or not (square icon)

A hidden layer still appears in final movie.

To hide all but one layer

alt+click below eye icon or right-click and use menu.

A locked layer cannot be altered.

To lock all but one layer

alt+click below padlock icon or right-click and use menu.

Outline show what belongs to a layer using colored outlines (each layer has its own color).

To show outlines all but one layer

alt+click below square icon or right-click and use menu.

To change Outline properties

double click on outline icon or right shift ➜ Properties.

More than one layer may be selected at a time: shift+click or ctrl+click

Delete a layer

Select and click Trash can on bottom of layer window.

Copy a layer

Select the layer
Edit ➜ Timeline ➜ Copy Frames
Create a new layer
Select the new layer
Edit ➜ Timeline ➜ Paste Frames

Order of layers: Top layer is front of movie, bottom layer in back.

Drag layers to reorder them.

Layers may be placed in folders for organization.

Layer Properties: Modify ➜ Timeline ➜ Layer Properties or
right click on layer and choose Properties

Several objects on the stage can be placed in separate layers

Modify ➜ Timeline ➜ Distribute to Layers (shift+ctrl+d)

Symbols

Three kinds

Graphic symbols (static)

Movie clip symbols (may have animation)

Button symbols (may have animation and behavior)

Creating Symbols

Select a drawing and use Modify ➜ Convert to Symbol (F8)

Insert ➜ New Symbol (ctrl+F8): Brings up a new drawing surface

Return to Scene from editing a symbol

ctrl+e

click on scene name in title bar

Edit ➜ Edit document

Change Properties of a Symbol

right click on symbol in Library and choose Properties.

Place an instance of a symbol on stage: Drag it from Library.

Editing a Symbol

Double click on Symbol icon in Library.

Double click on an instance of the Symbol.

Select a Symbol instance, right click, and choose

Edit

Edit in Place

Edit in a New Window

Editing a Symbol Instance

Select instance

Use Properties Panel to change instance with the Color Menu:

Brightness, Tint, Alpha, Advanced

Animation

Three methods

Frame by frame (tedious)

Tweening

Timeline Effects

We can change a movie only on Keyframes.

Insert a Keyframe

Select frame and Insert ➜ Timeline ➜ Keyframe (F6)

Insert a (regular) frame

Select frame and Insert ➜ Timeline ➜ Frame (F5)

Example: Disks jump from position to position according to Keyframe drawings

Tweening

Two kinds: Motion and Shape

Motion Tweening

Can act on Symbol instances, groups, or text instances

Change: position, size, rotation, skew, color, transparency

Creating a Motion Tween

Place two Keyframes on timeline, say frame 1 (already a Keyframe) and frame 30.

Select Keyframe 1 and place Symbol instance or text on stage.

Select Keyframe 30 and place Symbol instance or text on stage.

Create Motion Tween (three methods): Arrow with light blue background

Click on left Keyframe and select Motion from Properties panel Tween menu

Right click anywhere between Keyframes and choose Create Motion Tween

Click anywhere between Keyframes and choose
Insert ➜ Timeline ➜ Create Motion Tween

Examples: Moving Symbols, Rotation word, Skewing symbol, Text moving down and increasing in size, Symbol changing color and transparency

Note that changes in Symbol instance are made using Transform Tool or the Color menu in Properties Panel.

Shape Tween

Works with a drawn object, not with a Symbol or text object.

Change a Symbol or text object into a drawing

Modify ➜ Break Apart (ctrl+b)

Once for a single text character

Twice for a text object with multiple characters

Creating a Shape Tween

Place two Keyframes on timeline, say frame 1 (already a Keyframe) and frame 30.

Select Keyframe 1 and create a drawing on stage.

Select Keyframe 30 and create a drawing on stage.

Create Shape Tween: Arrow with light green background

Click on left Keyframe and select Shape from Properties panel Tween menu.

Shape Hints

Labeled a to z (up to 26 hints)

Create a Shape Hint

- Modify ➜ Shape ➜ Add Shape Hint or
- ctrl+shift+H

Arrange shape hints on start drawing (yellow hints) and on end drawing (green hints)

Place hints counter-clockwise from upper left

Moving Object Along a Path

Only possible with a Motion Tween

Steps

- Insert Keyframes left and right.
- Choose layer and create a Motion Guide layer (three methods):
  - Insert ➜ Timeline ➜ Motion Guide or
  - Click second icon at bottom of Layers window
  - Right click and choose Add Motion Guide
- Select Motion Guide layer.
- Draw a path on this layer (pencil, brush, or other drawing tool).
- Select View ➜ Snap to Objects (the default setting).
- Place a Symbol at beginning of path on left Keyframe (snap center point).
- Place a Symbol at end of path on right Keyframe (snap center point).
- Create a Motion Tween on Motion Guide layer.

Motion Guide layer may be locked and/or hidden.

Editing Animation

Drawing on stage can be changed only on Keyframes.

Stretch or contract a tween: Double click on a Keyframe and drag either way.

Reversing animation: Select animation layer and Modify ➜ Timeline ➜ Reverse frames.

Add regular frames in a tween to slow animation down.

Change frame rate to speed up or slow down animation.

Differences between Motion Tweens and Shape Tweens

Motion: Works on Symbol instances, grouped drawings, and text objects.

Shape: Works on drawn shapes and figures.

Movement along a path only possible with a Motion Tween.

Morphing a shape only possible with a Shape Tween.

Movie Clip Symbols

Insert ➜ New Symbol (ctrl+F8)

  Name symbol and select Movie Clip for type

Edit Symbol as a new movie with its own layers and tweens.

Example

  Fat oval to a Skinny oval and then back to a Fat oval

  Center drawing (Cut and Paste)

  Create a tween from Fat oval to Fat oval so that the start and end pictures line up

  Create a Keyframe in the middle with the Skinny oval using Transform to shrink it

  Return to main scene (ctrl+e)

  Place several of the new Movie Clip Symbols on the stage

Button Symbols

Insert ➜ New Symbol (ctrl+F8)

  Name symbol and select Button for type.

Edit Symbol: Create graphics for

  Up state

  Over state

  Down state

  Hit state (maybe): a solid shape covering the button graphic

Need a Keyframe for each state that we edit.

Add a Sound

Import a sound file

  File ➜ Import ➜ Import to Library (or Scene)

  Edit Button Symbol

  Put a Keyframe at Down state

  Place sound

Drag sound from Library toe Down frame or

Select sound from Properties Panel for Button Symbol.

Animated Button

Place a Movie Clip Symbol on the Keyframe state that should be animated.

Action for a Button

Two possibilities

Write ActionScript code (later)

Use Window ➜ Behaviors (shift+F3)

Place a Button Symbol on the stage.

Choose the Behaviors Panel.

Click the plus sign and choose category and subcategory.

Example

+ ➜ Web ➜ Go to URL

Enter a complete URL with protocol in window.

Options: _self uses the same browser window
_blank opens a new browser window

Example

Place text on stage: "Enter a web address: ".

Use Text Tool with Input Text selected to place a text field on the stage and give the field an instance name, say *webAddress*.

Draw a rectangle around the text field with white fill.

Place a Button Symbol on the stage and give it an instance name, say *myButton*.

Create a new layer called *Actions*,

Choose Actions layer, open Window ➜ Actions (alt+F9), and type

```
myButton.onRelease = function()
{
    getUrl("http://"+webAddress.text, "_self");
}
```

Masks

Actually portholes for spotlights

Place an image on the stage

- File ➜ Import ➜ Import to Library (or Scene)
- Match stage size with image size

Create another layer

- Place a Symbol on the layer at different positions for a left Keyframe and a right Keyframe
- Create a Motion Tween between the Keyframes
- Turn new layer into a Mask: right click and choose Mask or use Layer Properties
- Mask layer is locked when it becomes a Mask

Place matching Keyframes on the background (image) layer.

The tween on the Mask layer could as well be a Shape Tween.

Mask with a Motion Guide

Place an image on the stage.

Create a new layer, which will be the motion mask.

On the motion mask layer,

- Draw a shape, say a disk.
- Convert it to a Symbol as a Movie clip (F8).
- Double click on the disk to go into Symbol edit mode.
  - This edit mode provides a new layer to work on.
  - Convert the disk itself into a Symbol as a Movie clip (F8).
  - Create a Keyframe at frame 60 of the symbol editing layer.
  - Add a Motion Guide to the new Symbol editing layer.
  - Draw a path from the disk Symbol on the Motion Guide layer.
  - Select Keyframe 60 of the Motion Guide layer.
  - Place a disk Symbol at the end of the path on the Motion Guide layer.
  - Create a Motion Tween on the main layer for the Symbol edit.

Return to the main scene.

Make the motion mask layer into a Mask.

Timeline Effects

Flash provides eight Timeline Effects that act as shortcuts for creating animations and drawings.

Copy to Grid

Insert ➜ Timeline Effects ➜ Assistants ➜ Copy to Grid

Options

- Grid Size
- Grid Spacing

Examples: Matrix of stars, Matrix of spam cans

Distributed Duplicates

Insert ➜ Timeline Effects ➜ Assistants ➜ Distributed Duplicates

Options

- Number of Copies
- Offset Distance
- Offset Rotation
- Offset Start Frame (delay in frames for creating duplicates)
- Scaling (Exponential or Linear)
- Color, Final Color, and Alpha

Example: Duplicate a die

Blur

Insert ➜ Timeline Effects ➜ Effects ➜ Blur

Options

- Effect Duration
- Resolution (number of copies displayed)
- Scale (factor by which object grows)
- Allow Horizontal Blur
- Allow Vertical Blur
- Direction of Movement

Example: fire.jpg

Expand

- Insert ➜ Timeline Effects ➜ Effects ➜ Expand
- Object expanded must be a Symbol
- Options
    - Effect Duration
    - Expand, Squeeze, or Both
    - Direction of Movement
    - Shift Group Center by
    - Fragment Offset
    - Change Fragment Size by
- Example: Beating heart

Explode

- Insert ➜ Timeline Effects ➜ Effects ➜ Explode
- Options
    - Effect Duration
    - Direction of Explosion
    - Arc Size
    - Rotate Fragments by
    - Change Fragment Size by
    - Final Alpha
- Examples: Herky, fire.jpg

Drop Shadow

- Insert ➜ Timeline Effects ➜ Effects ➜ Drop Shadow
- Options
    - Color
    - Alpha Transparency
    - Shadow Offset
- Examples: Herky, star

Transform

- Automatic Tweening
- Insert ➜ Timeline Effects ➜ Transform/Transition ➜ Transform
- Options
  - Effect Duration
  - Change Position by
  - Scale
  - Rotate or Spin
  - Change Color
  - Final Alpha
  - Motion Ease (Slow at Start or End)
- Example: star

Transition

- Fade in/out and Wipe
- Insert ➜ Timeline Effects ➜ Transform/Transition ➜ Transition
- Options
  - Effect Duration
  - Fade in/Fade out
  - Wipe: object appears bit by bit along a line
  - Direction
  - Motion Ease (Slow at Start or End)
- Examples: fire.jpg and stars

Notes on Timeline Effects

- Cannot edit Timeline Effects Symbols in the normal Symbol editing mode.
- Cannot add Keyframes on a layer with a Timeline Effect.
- To alter a Timeline Effect, select
  Modify ➜ Timeline Effects ➜ Edit Effect.
- To delete a Timeline Effect, select its layer and click
  Modify ➜ Timeline Effects ➜ Remove Effect.

Scenes

Sequential animation can be organized into scenes to make development easier.

Flash documents open with one scene, called "Scene 1".

Open another scene (two ways)

Insert ➜ Scene

Window ➜ Other Panels ➜ Scene (shift+F2) and click **+** at bottom of Scene Panel.

Scenes can be renamed by double clicking on them in the Scene Panel.

Drag scenes in the Scene Panel to reorder them.

Changing from one scene to another scene:

Select scene from Scene Panel or

Choose scene from the menu below the movie clapperboard icon in the title bar.

Delete a Scene: Choose it in Scene Panel and click on trash can.

ActionScript Examples

ActionScript is a programming language similar to JavaScript.

An action (a code fragment) can be attached to a Keyframe or to an object (a symbol or component).

Example

Create a Button Symbol.

Give it an instance name, say "xyz".

Select the button instance an open the Actions Panel using
Window ➜ Actions (alt+F9)

Enter the following code in the Actions Panel:

```
on (release)
{  stopDrag();  }

on (press)
{  startDrag(xyz);  }
```

Create a Scene 2 with some animation, say a Transform Timeline Effect.

Play the movie and note that Scene 1 moves directly into Scene 2.

Return to Scene 1, choose frame 1, a Keyframe, and enter the following code:

```
stop();
```

Create another Button Symbol on Scene 1.

Select the new button, open the Actions Panel (if not open), and enter this code:

```
on (release)
{  gotoAndPlay("Scene 2", 1);  }
```

Try the movie now.

Custom Colors

Consider the **Color Mixer Panel**, choosing Stroke icon or Fill icon.

Note Option button on the right side of the title bar.

Choose RGB (the default) or HSB .

RGB

Red-Green-Blue: eight bits each, represented as an unsigned byte, 0 to 255.

HSB

Hue-Saturation-Brightness: eight bits each, represented as an unsigned byte, 0 to 255.

Hue: Color type (red, blue, etc.)

Ranges from 0 to 360 degrees of a color wheel.

red	0°	yellow	59°
green	120°	cyan	179°
blue	240°	magenta	301°

Saturation: Intensity of color

Ranges from 0% to 100%.

0% is no color, a shade of grey between black and white.

100% is an intense color.

Brightness (or value): Amount of white in the color

Ranges from 0% to 100%.

0% is black.

100% is white or a more or less saturated color.

Color Selection Square (Rectangle): Click to choose a color.

Luminance Slider: Shade of color

Hexadecimal value: #000000 to #FFFFFF (RGB or HSB).

Alpha Percentage: Ranges from 0% to 100%.

Three Special Buttons: Black/White, No Color, and Swap Stroke and Fill.

Select a Color from Elsewhere:

Click and drag on the fill (or stoke) icon in the Color Mixer Panel.

Color can be chosen from outside of Flash, say a Web page or an image.

When the mouse (an eyedropper) is released, the color it is over becomes the fill color.

Save a Custom Color

Options menu ➜ Add Swatch

Swatch is placed below existing swatches in the Color Swatches Panel.

Alternatively, switch to the Color Swatches Panel and click in the region below the existing swatches (pointer is a paint bucket.).

Click on Stroke icon or Fill icon

Produces Color Swatches

216 web-safe colors

Swatches that we have added

Some gradient swatches

Color Picker Button (on right at top): Produces different panels on PCs and Macs.

Color Swatches Panel

Options Menu

Delete swatch
Web 216
Sort by Color
Save Colors
Replace Colors

Gradients

Coloring effect that blends bands of color into each other.

Linear gradients: Straight left-to right, which can be rotated to other orientations (see Gradient Transform tool).

Radial gradients: Bands of color that begin in the center of a circle and radiate outward.

Six gradient swatches reside on the Color Swatches Panel.

Custom Gradients

Enter the Color Mixer Panel.

Choose Linear or Radial from the Type menu.

Current gradient shows in rectangle at the bottom of the Panel.

Above the sample is a narrow rectangle that acts as a slider with two pointers, a dark pointer and a light pointer.

The pointers can be moved and even reversed.

Change Pointer Color

Select pointer (its arrowhead is black).

Choose a color using RGB sliders, clicking in the Color Square, entering a new hexadecimal value, and/or moving the luminance slider.

Also, by clicking a pointer until a swatches panel drops down for selecting a new color.

Add a New Pointer

Click on the narrow slider rectangle.

Delete a Color Pointer

Drag the point down away from the narrow rectangle.

A gradient may be switched between Linear and Radial, maintaining the same color bands, using the Type menu.

Gradient Transform Tool (f)

Edits gradients already applied to a fill or stoke of a drawing on the stage.

Can be applied to gradient fills and to bitmap fills.

Handles for the Gradient Transform

Round white circle: Center point of the gradient.

Circle with delta: Used to rotate the gradient.

Square with arrow: Used to stretch or shrink the gradient along one axis.

Del symbol:
Focal point of gradient; used to move the highlight (radial gradients only).

Circle with arrow on ring:
Used to scale gradient symmetrically (radial gradients only).

Linear Gradients under the Gradient Transform

Vertical lines to left and right of gradient, marking the ends of the gradient.

Can be moved using the Square with an arrow.

When the gradient is smaller than the shape that contains it, the Overflow rule fills the rest of the shape.

Radial Gradients under the Gradient Transform

A circle surrounds the gradient, marking the edge of the gradient.

Can be moved using the Square with an arrow or with the Circle with an arrow.

When the gradient is smaller than the shape that contains it, the Overflow rule fills the rest of the shape.

Overflow Options

Extend (default): The color(s) at the edge(s) of the gradient is (are) continued.

Reflect: A reversed version of the gradient continues from the edge(s).

Repeat: A new copy of the gradient continues from the edge(s).

ActionScript

Code is executed when an event occurs:

- Movie clip is loaded.
- Keyframe on timeline is entered (a frame event).
- User clicks a button.

We write an event handler to respond to the event with an action.

Two groups of code

- Scripts for events that occur on the timeline.
- Scripts for object instances (buttons, movie clips, and components).

Write code in

- Actions Panel (alt+F9)
- Script window (File ➜ New and select ActionScript File) creates a file with an *as* extension.

Actions Panel

- Script Pane: Write code here.
- Actions Toolbox: List of language elements.
- Script Navigator: Clickable tree showing elements that have scripts.
- Tab at bottom left of Script Pane shows the element the current script is attached to.
- Note that a timeline Keyframe with a script has a little *a* on it.

Add an item to a script (Double-click item or drag it to Actions Pane)

- Item from Actions Toolbox
- Item from menu below plus sign

Options Menu

- Go to Line
- Find and Replace (ctrl+f)
- Auto Format (ctrl+shift+F)
- Check Syntax (ctrl+t)
- Print
- Hidden Characters (ctrl+shift+8)
- Preferences (ctrl+u)
- Import Script … (ctrl+shift+I)
- Export Script … (ctrl+shift+X)

Example: Script for Frame 1.
Button with a script.
Script for Frame 20.

Global Variables

Visible to every timeline and symbol in document.

Creating:

```
_global.gName = "Herky";
```

Can be hidden by timeline variables and local variables in inner scopes, but can always be accessed using *_global.gName*.

Global variables cannot be typed because they are created without the **var** keyword.

Timeline Variables

Visible to any script on that particular timeline (all layers).

Creating:

```
var tName = 66;
```

Local Variables

Visible only in the method where they are declared using **var**.

Can hide Global variables and Timeline variables.

Actionscript does not allow anonymous blocks (braces not belonging to a method).

Declaring Variables

Untyped (can be assigned any kind of value):

```
var x;
```

Typed:

```
var n:Number;
var s:String;
```

With Initialization:

```
var y = 123;
var z:Boolean = true;
```

Identifiers

Names of variables, methods, classes, etc.

Rule: A letter, underscore, or dollar sign followed by zero or more letters, digits, underscores, and dollar signs.

Avoid the Reserved Words: 51+12 Keywords and 93 Class and Interface names.

Avoid several predefined constants: *true*, *false*, *null*, *undefined*

Primitive Data Types

Boolean

Values: *true* and *false*

Number

Digits with an optional sign in front, an optional decimal point, and an optional exponent using *e* or *E*.

Number is an integer if no decimal point and no exponent.

Number.MAX_VALUE: 1.79769e308

Number.MIN_VALUE: 4.940546e-324

String

Sequence of Unicode characters delimited by quote symbols or apostrophes.

undefined

One value: *undefined*

Value of uninitialized variables.

null

One value: *null*

Means "no value".

Default data type for all classes that define complex data types except Object.

Complex Data Types

Object

Base class for all class definitions.

MovieClip

Its methods are used to to control Movie Clips.

Array

Used to type arrays.

Void

One value: *void*

Used to designate the return type of methods that return no value.

Coercion

In the absence of typed declarations, primitive type values are coerced if possible.

If not possible, the value *NaN* is produces.

Without typed declarations, ActionScript has dynamic typing and virtually no type checking.

Examples

```
var n = 12;
var b = true;
var s = "abc";
var sd = "123";

if (n) trace(true);       // true
else trace(false);

if (s) trace(true);       // true
else trace(false);

trace(sd-5);              // 118

trace(s-5);               // NaN

trace(b+100);             // 101
```

With typed declarations, ActionScript has static typing so that errors are caught by the compiler and reported in the Output Panel.

Trace Method

The method *trace* takes one parameter, converts it to a string, and prints it in the Output Panel, which automatically appears.

This Output Panel also appears if you script has a syntax error.
The error will be reported in the Panel.

Example: Prime Number Tester

Scene 1

A text string: Prime Number Tester

A text string: Enter an integer

An input text field (Text Tool) surrounded by a rectangle; instance name of field is *testNum*.

A Button Symbol with the message "Test Number" and an ActionScript:

```
on (release)
{
        var prime:Boolean;
        var num:Number = Number(testNum.text);
        if (isNaN(num) || num==1)
        {
            prime = false;
        }
        else
        {
            if (num == 2)
                prime = true;
            else if (num % 2 == 0)
                prime = false;
            else
            {
                prime = true;
                var div:Number = 3;
                while (div<=Math.sqrt(num) && prime)
                {
                    if (num % div == 0)
                        prime = false;
                    else
                        div = div+2;
                }
            }
        }
        if (prime)
            gotoAndStop("Scene 2", 1);
        else
            gotoAndStop("Scene 3", 1);
}
```

Scene 2

A text string: Number is Prime

A Button Symbol with the message "Return to Start Page" and an ActionScript that returns to Scene 1 when the button is released.

Scene 3

A text string: Number is Not Prime

A Button Symbol with the message "Return to Start Page" and an ActionScript that returns to Scene 1 when the button is released.

Make certain that the labels on the buttons are static text.

Dynamic Text (and Input Text)

Two ways to access contents:

Instance name: *abc*

Var: *xyz*

Both must be ActionScript identifiers.

Access text

```
abc.text = "Herky";
xyz = "Hawk";
```

Buttons on Properties Panel

Selectable (can be copied and pasted)

Render as HTML

Show border around text

Example

Keyframes at 1, 20, and 40 that change the value of a dynamic text field.

Input Text

Additional field in Properties Panel: Maximum characters

Movie Clip Control

ActionScript code can be used to control the execution of a Movie Clip.

Example

1. From Keyframe 1 to Keyframe 20 have a word (Herky) move from off the stage to the center using a Motion Tween.
2. From Keyframe 20 to Keyframe 30 have the word rotate once: Create a Motion Tween in the Properties Panel and select Rotate: CW.
3. On Keyframe 30 enter code: gotoAndPlay(20);

What if we stretch or compress the tweens? Keyframe 20 may no longer be the beginning of the rotation action.

4. Select Keyframe 20 and enter a Frame label "Start Loop". Change the code at Keyframe 30 to gotoAndPlay("Start Loop");

Buttons to Control Action

Continue Example

5. Create a new layer called Buttons so we do not affect the existing tweens.
6. Create a Button Symbol and place two instances of it on the stage, one with label "Start" and one with "Stop".
7. Add code for Start button
```
on (press)
{  play();  }
```
8. Add code for Stop button
```
on (press)
{  stop();  }
```
9. On Keyframe 1 of either layer (or a new one) enter
```
stop();
```

Put All Code on Timeline

Continue Example

10. Remove code from two buttons.
11. Give each button an instance name, say *start* and *stop*.
12. Add a layer called Actions.
13. Enter this code on Keyframe 1 of the Actions layer.
```
stop();

start.onPress = function()
                {  play();  }

stop.onPress = function()
                {  stop();  }
```

Note that these two functions have no names; they are anonymous functions.

Nested Movie Clips

Flash documents typically have Movie Clip instances on their main timeline.

These Movie Clips may have other Movie Clip instances inside of themselves, and so on.

Example

1. Create a spoked wheel; Draw a circle with one line bisecting it. Use Transform Panel, Rotate 60° with "Copy and apply transform" button.
2. Change the color of several of the spokes so the motion can be seen.
3. Convert drawing to a Movie Clip Symbol called Wheel.

4. Select the instance of Wheel and convert it to a Movie Clip Symbol called RotatingWheel.
5. Edit RotatingWheel, which contains a Wheel Symbol inside of it.
6. Place a Keyframe at from 30 and create a Motion Tween, selecting CW from the Rotate menu in the Properties Panel.
7. Place another RotatingWheel instance on the stage and draw a car over the two wheels. Select all and create a Movie Clip Symbol called Car.
8. Add a new layer (Buttons) and place two instances of a Button Symbol on the new layer.
9. On one button enter the code

```
on (press)
{  stop();  }
```

   and try the movie (the wheels do not stop).

10. On the other button enter the code

```
on (press)
{  play();  }
```

11. Edit the Car Symbol. Give instance names to the two RotatingWheels, say *frontWheel* and *backWheel*. Give the Car instance a name, say *car*.
12. Select the stop button and change the code to

```
on (press)
{
    stop();
    car.frontWheel.stop();
    car.backWheel.stop();
}
```

13. Select the start button and start a new line below play();
    Click "Insert a target path" button (a circle over a cross) and choose car ➜ backWheel. Then type "play();".
    Do the same with frontWheel to get the following code.

```
on (press)
{
    play();
    this.car.frontWheel.play();
    this.car.backWheel.play();
}
```

Bouncing Ball

This example is designed to show how to complete the next project.

Example

1. Draw a green disk on the stage and convert it to a Movie Clip Symbol called GreenBall.
2. Movie Clip Symbols recognize a number of predefined properties such as *_x*, *_y*, *_width*, and *_height*. By altering the *_x* and *_y* properties, we can cause the Movie Clip Symbol to move across the stage.
3. Just as Buttons respond to events (*press* and *release*), Movie Clips respond to events such as *load*, *enterFrame*, *mouseDown*, *keyDown*, among others.
4. In the same way we handle Button events with an *on* command, we handle Movie Clip events with an *onClipEvent* command.
5. Add the following code to the instance of GreenBall.

```
onClipEvent (load)
{
    speedX = 8;
}
onClipEvent (enterFrame)            // executed as each frame is entered
{
    _x = _x + speedX;
}
```

   The ball will move horizontally across the stage, continuing off the right side.

6. To get the ball to bounce against the right edge to the stage, change the code as follows.

```
onClipEvent (enterFrame)
{
    _x = _x + speedX;
    if (_x >= 550)
    {
        _x = 550;                   // bring back to the edge
        speedX = -speedX;           // change directions
}
```

   Problem: Ball misses the edge by a little because the *_x* property is the registration point of the Movie Clip instance (the plus sign) usually at the center of the object.

7. Assuming the registration point is at the center of the disk, change the code as follows.

```
onClipEvent (enterFrame)
{
    _x = _x + speedX;
    if (_x + _width/2  >= 550)
    {
        _x = 550 - _width/2;
        speedX = -speedX;
}
```

8. Now the ball bounces correctly on the right but continues forever moving to the left off the stage. Another problem is that if we change the dimensions of the stage, the 550 numeral in the code may no longer be accurate. The next version of the code solves both of these problems.

```
onClipEvent (enterFrame)
{
    _x = _x + speedX;
    if (_x + _width/2  >= Stage.width)
    {
        _x = Stage.width - _width/2;
        speedX = -speedX;
    }
    else if (_x - _width/2  <= 0)
    {
        _x = width/2;
        speedX = -speedX;
    }
}
```

9. Variations: Change *onClipEvent (enterFrame)* to *onClipEvent (mouseDown)* or to *onClipEvent (keyDown)* and observe that pressing the mouse or pressing any key will cause the ball to move.

Movie Clip Events (parameters to *onClipEvent*)

load, unload, enterFrame, mouseDown, mouseMove, mouseUp, keyDown, keyUp

Button Events (parameters to *on*)

press, release, releaseOutside, rollOver, rollOut, dragOver, dragOut, keyPress "some key"

History Panel

Records each action taken as you create a Flash document.

Window ➜ Other Panels ➜ History (ctrl+F10)

Default: History Panel stores 100 actions, a number that can be altered in Flash Preferences.

Options Menu (or right button click)

Replay Steps (also a button at bottom of panel).

Copy Steps (also a button at bottom of panel).

Save As Command (also a button at bottom of panel).

Clear History (no undo).

View (Arguments in Panel or JavaScript in Panel).

Example

Make a rectangle Movie Clip Symbol.

Replay commands.

Save As Command.

Redo it.

Add a text field.

Choose View ➜ Arguments in Panel.

Choose View ➜ JavaScript in Panel.

Drag arrow in History Panel up.

Then drag it back down.

CopyPasteMove

On History Panel choose View ➜ Arguments in Panel.

Place a colored disk on the stage.

Select the disk.

- Edit ➜ Copy (ctrl+c).
- Edit ➜ Paste (shift+ctrl+V).
- Move object to the left about 100 pixels.

Select last three commands from History Panel.

Select Save As Command and name the command "CopyAndMove".

Use the new command: Commands ➜ CopyAndMove.

This behavior is essentially the same as the Edit ➜ Duplicate (ctrl+d) command.

Managing Commands

Commands ➜ Manage Saved Commands

Select a command.

The command may be renamed or deleted.

CenterOnStage

Select an object on the stage.

In Align Panel, click To Stage modifier, click Align horizontal center, and click Align vertical center.

Select the last two actions in the History Panel and Save As Command with the name "CenterOnStage".

This new command can be very useful.

Filters

Accessed by a tab on the Properties Panel.

Filters can be applied to a Symbol *instance* (Button or Movie Clip only) or to a text object.

Procedure

Open Filters Panel.

Select the object to be altered.

Select a filter to apply using the Add filter button + menu.

Use the Remove filter button − to delete the chosen filter.

The Add filter button also allows Remove All, Enable All, and Disable All.

More than one filter may be applied to an object.

The order of filter application may make a difference.

Blur Filter

This filter blurs the entire content of the instance.

Parameters allow more or less blurring in the x or y direction.

Example

Enter some fairly large text, say "Flash Filters" with a sans serif 40 point font.

Select the Blur filter.

Place a Keyframe at frame 40 and change the Blur x and Blur y values to zero there.

Place a motion tween between Keyframe 1 and Keyframe 40.

Change Blur x and Blur y values to 70 at Keyframe 1.

To reduce flicker in the tween, change the Quality value to High at both Keyframes.

Glow Filter

This filter makes a duplicate of the instance's shape and blurs it.

It is like having a blurred red figure with a copy of your original figure layered on top.

Drop Shadow Filter

This filter creates a single-color copy of the instance's shape, slips it underneath the instance, and offsets its location.

Options include the strength (alpha), blur, angle, and offset distance.

Example

Enter some large text, say "Flash Filters" with white characters on a black background.

Select the Glow filter.

- High Strength, say 380%.
- Medium Quality.
- Select a green Color.

Select Drop Shadow filter.

- Medium Quality.
- Distance equal to 7.
- Select a yellow Color.
- Try different Angle values.

Try some other options.

- Knockout: Removes original figure.
- Inner Shadow: Puts glow or shadow inside the figure.
- Hide Object (Drop Shadow only): Shows only the shadow.

Controls

Strength: Amount of the effect, usually changing the brightness and alpha of the effect.

Quality: Higher quality means better transitions (smoother) and greater aliasing, but also more computation (slower).

Bevel Filter

This filter gives the instance an embossed look.

This effect is produced by lightening the upper left of the drawing and darkening the lower right as if there is a light source coming from the upper right.

Options allow a change in the direction of the light source and the colors of the effect.

Example

Create a Movie Clip Symbol from a blue rectangle.

Apply the Bevel filter to an instance of the Symbol.

Options: Blur x, Blur y, Strength, Quality, Angle, Distance, Type, and Knockout.

Note: A negative distance reversed the bevel, which can be useful in showing a button press.

Do the same thing with large colored text.

Gradient Glow and Gradient Bevel Filters

These filters are essentially the same as the Glow and Bevel filters except that they allow a gradient color for the effects.

Adjust Color Filter

This filter provides tools for manipulating the color properties of the instance.

Since instances made from drawings or text objects can be manipulated by other means, this filter is useful primarily for instances formed from a bitmap picture.

Slider Options: Brightness, Contrast, Saturation, and Hue.

Try an example on a *jpeg*, *gif*, *png*, or some other picture type.

Keypad Example

This example illustrates the use of buttons to produce and process data.

1. Create a new Button Symbol, called TheButton.

Up:	Red square with a Bevel filter, Blur x, Blur y: 10 and Quality: High).
Over:	Adjust Color filter, Brightness - 35.
Down:	Remove Adjust Color filter and change Bevel Distance to -5.

2. Place four buttons on the stage in a horizontal row. Give them instance names, *one*, *two*, *three*, and *four*. Call this layer "Buttons".
3. On an new layer, called "Labels", above the Buttons layer, enter four digits, 1, 2, 3, and 4, on top of the buttons using Arial 18 point bold black font.
4. Build another Button Symbol, called ResetButton,

Up:	Blue rectangle with a Bevel filter, Blur x, Blur y: 8 and Quality: High).
Over:	Add a Glow filter with Strength: 190% and Color: White.
Down:	Remove Glow filter and change Bevel Distance to -5.

5. Place one of the ResetButton Symbols on the stage with the instance name *reset*.
6. On the Labels layer, enter "Reset" over the new button using Arial 18 point bold white font.

7. Between the digit buttons and the reset button place a Dynamic Text field using Arial 24 point bold black font. Give it the instance name *countField*.

8. Above the four digit buttons place another Dynamic Text field using bold 36 point Arial in black. Click the border button and the right justify button in the Properties Panel. Give this object the instance name *display*.

9. Add another layer above the previous two. Call it "Actions".

10. Select Keyframe 1 in the Actions layer and enter the following code.

```
display.text = "";
countField.text = "Number of digits: 0";
var count:Number = 0;                          // KeyPad.fla

one.onRelease = function()
{       keyPressed("1");
}

two.onRelease = function()
{       keyPressed("2");
}

three.onRelease = function()
{       keyPressed("3");
}

four.onRelease = function()
{       keyPressed("4");
}

reset.onRelease = function()
{       keyPressed("reset");
}

function keyPressed(key:String)
{
        if (key == "reset")
        {
              countField.text = "Number of digits: 0";
              count = 0;
              display.text ="";
              return;
        }
        display.text = display.text + key;
        count++;
        countField.text = "Number of digits: " + count;
}
```

Conversion Functions

These functions can be used to convert values from one primitive type to another.

parseFloat : String → Number
parseInt : String → Number
Number : String → Number

toString : Number → String   (an instance method)
String : Number → String

Number : Boolean → Number

Boolean : Number → Boolean

Boolean : String → Boolean

toString : Boolean → String   (an instance method)
String : Boolean → String

The three conversion functions, Number, Boolean, and String, can be applied to values of any type, resulting in some kind of value no matter what.

Number Function

Parameter	**Result**
undefined	0
null	0
Boolean	1 if true, 0 if false
Number string	Equivalent number if string is a properly formatted number
"Infinity"	Infinity
"-Infinity"	-Infinity
"NaN"	NaN
Other strings	NaN
Array	NaN
Object	Result from applying valueOf() to the object
Movieclip	NaN

String Function

Parameter	**Result**
undefined	""
null	"null"
Boolean	"true" if true, "false" if false
NaN	"NaN"
Numeric value	String representation of the number
Infinity	" Infinity"
-Infinity	" -Infinity"
Array	Comma-separated list of array values
Object	Result from applying toString() to the object
Movieclip	Path to movie clip instance

Boolean Function

Parameter	**Result**
undefined	false
null	false
NaN	false
0	false
Infinity	true
-Infinity	true
Other numeric value	true
Empty string	false
Nonempty string	true
Array	true
Object	true
Movieclip	true

Object Oriented Programming in ActionScript

Classes

- Classes are syntactic units used to define objects.
- A class may contain instance variables, which will occur in each instance of the class, instance methods, which can be executed by objects of the class, and *one* constructor, which is called automatically when an object is created using **new**.
- Classes may also have class variables and class methods, but these belong to the class itself and have no direct effect on the objects. ActionScript does not allow *final* as a modifier.
- A class must be defined in a file of its own using the name of the class as the file name and using *as* as the extenstion. The following example will be defined in the file *MyClass.as*.

```
class MyClass
{
        private var value : Number;
        public function MyClass(n : Number)
        {       value = n;      }
        public function perform(m : Number) : Void
        {
                for (var k:Number = 1; k<=value; k++)
                        trace(m*k);
        }
        public function compute() : Number
        {       return value*value;          }
}
```

Modifiers	public	accessible anywhere
	private	accessible in this class and any subclass
	no modifier	same accessibility as public

Objects

- Objects are created from a class using the **new** operator, which invokes a constructor with matching parameter types.
- These objects may be assigned to variables declared of the type given by the class name.
- Each object has a copy of every instance variable in its class definition and in every superclass of that class.
- Instance methods in a class can be called only with an object of the class type (or a subclass).
- This object is called the receiver of the method and can be referred to by the keyword **this** inside of the method.

- The following code might appear in a Flash document. If so, the class file MyClass.as must be in the same directory as the Flash document.

```
var first : MyClass = new MyClass(5);
var second : MyClass = new MyClass(3);

first.perform(6);
        Prints: 6  12  18  24  30

second.perform(-4);
        Prints: -4  -8  -12
```

Constructors

- A constructor is a method that is called automatically when an object is created.
- If the programmer supplies no constructor, a default constructor with no parameters is provided.
- This default constructor disappears if the programmer writes a constructor in the class.
- A class can have at most one constructor since ActionScript does not support the overloading of methods.
- In a constructor, **super**(…) calls a constructor of its superclass with the given parameters.

Inheritance

- A new class can be defined as a subclass of an existing class using the **extends** keyword.
- Then every object of the new subclass will have copies of the instance variables from its superclass (and its superclass and so on) as well as its own instance variables.
- It can also call instance methods defined in its superclass.
- Any method in the superclass can be overridden (re-defined) by writing a method in the subclass with the same signature.
- Any class definition without an extends clause is a subclass of Object by default.
- A variable of the superclass type may refer to an object of its class or an object of any of its subclasses.
- If an overridden method is called on a variable of the superclass, the class of the object referred to determines which version of the overridden method will be executed. This property is known as polymorphism or dynamic binding.
- In an instance method, the identifier **this** refers to the object, the receiver, that is currently executing the instance method.
- The identifier **super** can be used to access instance methods (and variables) that have been overridden (and shadowed) in the subclass.

- The file containing a subclass definition must reside in the same directory as the superclass file.

```
class MySub extends MyClass
{                                                       // File: MySub.as
        private var okay : Boolean;
        public function MySub(n : Number, b : Boolean)
        {
                super(n);
                okay = b;
        }
        public function compute() : Number
        {
                if (okay) return super.compute();
                else return -1;
        }
}
var mc : MyClass = new MyClass(7);
var ms : MySub = new MySub(12, true);
var mcs : MyClass = new MySub(-9, false);
```

```
        mc.perform(4);          calls superclass method         // 4, 8, 12, ..., 28
        ms. perform(9);         calls superclass method         // 9, 18, 27, ..., 108

        mc.compute()            calls superclass method         // 49
        ms.compute()            calls subclass method           // 144
        mcs.compute()           calls subclass method           // -1
```

Upcasting and Downcasting

- Upcasting refers to the mechanism in which an object from a subclass is assigned to a variable declared of the superclass type. No special operator is required since the subclass object "is-an" object of the superclass type automatically.
- Downcasting refers to the assignment of a superclass variable to a subclass variable, and it requires an explicit cast to the type of the subclass.
- A variable of a superclass type can be cast to a variable of a subclass type only if it refers to an object of that same subclass type.
- If the object referred to by the superclass variable is not an object of the subclass, the result of the downcast is the value *null* (not an exception).
- Upcasting and downcasting of object types also applies to parameter passing and to any situation where an object of one type is impersonating another class type.

```
Upcasting:          mc = ms;
Downcasting:        ms = MySub(mcs);
```

Polymorphism Example

Define a collection of classes representing geometric solids and including a method for computing their volumes.

The superclass provides a String instance variable for identification and a volume method to be overriden.

```
class Solid
{                                              // File: Solid.as
        private var kind : String;

        public function getKind() : String
        {   return kind;   }

        public function volume() : Number       // This code is never executed
        {   return 0.0;   }
}

class Sphere extends Solid
{                                              // File: Sphere.as
        private var radius : Number;

        public function Sphere(r : Number)
        {
                radius = r;
                kind = "Sphere";
        }

        public function volume() : Number
        {       return 4.0/3.0*Math.PI*radius*radius*radius;}
}

class Cube extends Solid
{                                              // File: Cube.as
        private var length : Number;

        public function Cube(g : Number)
        {
                length = g;
                kind = "Cube";
        }

        public function volume() : Number
        {       return length*length*length;       }
}

class Cone extends Solid
{                                              // File: Cone.as
        private var radius : Number;
        private var altitude : Number;
        public function Cone(r : Number, a : Number)
        {
                radius = r;
                altitude = a;
                kind = "Cone";
        }
        public function volume() : Number
        {       return Math.PI*radius*radius*altitude/3.0;     }
}
```

To make use of Solid and its subclasses, create a Flash document with the following ActionScript code at Keyframe 1. Its file must be in the same directory as *Solid.as*, *Sphere.as*, *Cube.as*, and *Cone.as*. Save it before you try to test it.

```
    var list : Array = new Array(6);       // View as an array of Solid
    list[0] = new Cube(10);
    list[1] = new Cube(5);
    list[2] = new Sphere(10);
    list[3] = new Sphere(8);

    list[4] = new Cone(3, 5);
    list[5] = new Cone(8, 2);

    for (var k:Number=0; k<list.length; k++)
    {
        trace(list[k].getKind() + " volume = " + list[k].volume());
    }
```

Execution

```
    Cube volume = 1000
    Cube volume = 125
    Sphere volume = 418.879020478639
    Sphere volume = 268.082573106329
    Cone volume = 47.1238898038469
    Cone volume = 134.041286553164
```

Abstract Classes

- ActionScript does not allow abstract classes.
- Interfaces must be used instead.

Interfaces

- An interface is a class-like mechanism that provides a specification of behavior (the syntax) that classes must implement to make the behavior available. It contains only public instance method headers (no bodies). Variables of any kind are not allowed.
- No objects can be instantiated directly from an interface. A class **implements** an interface by giving complete definitions of all of the methods in the interface.
- Then a variable declared of the interface type can be made to refer to such an object that implements the interface.
- Polymorphism can be performed with a group of classes that implement a common interface.

```
interface Printable
{                                                   // File: Printable.as
        function printNum(n : Number) : Void;
}

class FirstImpl implements Printable
{                                                   // File: FirstImpl.as
        private var name : String;

        public function FirstImpl(s : String)
        {       name = s;       }

        public function printNum(n : Number) : Void
        {
                trace(name + " prints " + n);
        }
}

class SecondImpl implements Printable
{                                                   // File: SecondImpl.as
        private var ID : Number;

        public function SecondImpl(n : Number)
        {       ID = n;         }

        public function printNum(n : Number) : Void
        {
                trace("Number" + ID + " prints " + n);
        }
}
```

To make use of Printable and its implementations, create a Flash document with the following ActionScript code at its first Keyframe. It file must be in the same directory as *Printable.as*, *FirstImpl.as*, and *SecondImpl.as*, and must be saved before it is tested.

```
var fi : FirstImpl = new FirstImpl("Claude");

var si : SecondImpl = new SecondImpl(55);
var printer1 : Printable = fi;
var printer2 : Printable = si;

printer1.printNum(13);
printer2.printNum(-99);
```

Output

```
Claude prints 13
Number55 prints -99
```

Symbols as Classes

Movie Clip Symbols and Buttons Symbols can be viewed as classes that are instantiated when they are placed on the stage.

Example

Create a green ball as a Movie Clip Symbol, called GreenBall.

This operation implies a class definition of the form:

```
class GreenBall extends MovieClip
{   }
```

Put an instance of GreenBall on the stage, calling it *green*.

We can provide properties (instance variables) and behavior (instance methods) for the *green* instance using ActionScript in the Timeline.

```
green.dx = 8;
green.dy = 4;

green.move = function()
{
    green._x = green._x + green.dx;
    green._y = green._y + green.dy;
}
```

Control the animation using frame events attached to the main timeline Movie Clip, which is referred to by the global variable *_root*.

```
_root.onLoad = function()
{
    green.dx = 8;
    green.dy = 4;
}
```

```
_root.onEnterFrame = function()
{
        green.move();
        green.checkBounds();
        green.checkHit();                    // dealt with later
}
```

The decision making for bouncing off of the edges of the stage is encapsulated in an instance method *checkBounds* for the object named *green*.
Actually, the *_root* object is redundant, which *onEnterFrame* responds to the main timeline MoveClip.

```
green.checkBounds = function()                          // Assumes registration point
{                                                       // of the ball is at its center
        if (green._x + green._width/2 >= Stage.width)               // right edge
        {
                green._x = Stage.width - green._width/2;
                green.dx = -green.dx;
        }
        else if (green._x - green._width/2 <= 0)                    // left edge
        {
                green._x = 0 + green._width/2;
                green.dx = -green.dx;
        }
        if (green._y + green._height/2 >= Stage.height)             // bottom edge
        {
                green._y = Stage.height - green._height/2;
                green.dy = -green.dy;
        }
        else if (green._y - green._height/2 <= 0)                   // top edge
        {
                green._y = 0 + green._height/2;
                green.dy = -green.dy;
        }
}
```

Add Start and Stop buttons (Symbols) to the stage with instance names *myStart* and *myStop*.
Place the following code on Keyframe 1 with the code that is already there.

```
myStop.onPress = function()
{
          green.saveX = green.dx;
          green.saveY = green.dy;
          green.dx = green.dy = 0;
}
```

```
myStart.onPress = function()
{
        green.dx = green.saveX;
        green.dy = green.saveY;
}
```

We have added four instance variables (*dx*, *dy*, *saveX*, *saveY*) to the object named *green*, which also has access to instance variables (*_x*, *_y*, *_width*, *_height*, among others) inherited from its superclass MovieClip.

Add a blue Square Symbol instance to the stage. Make it 150 by 150 pixels.
Call its instance *square*.

Problem: We want the green ball to bounce off of the blue square.

Hit Test

Instance method in MovieClip:

hitTest : Object → Boolean

When the MovieClip *square* and the object *green* overlap,

square.hitTest(green) returns *true*.

Add an instance method:

```
green.checkHit = function()
{
      if (square.hitTest(green))
      {
            green.dx = - green.dx;
            green.dy = - green.dy;
      }
}
```

Problem: This behavior is not the way a ball bounces.

When the ball hits the left side or the right side of the square,
only the *dx* value should reverse.

When the ball hits the top side or the bottom side of the square,
only the *dy* value should reverse.

Solution: Test whether the ball is within |dx| distance of the left or right side of the square and whether it is within |dy| distance of the top or bottom of the square.

The one-dimensional distance between two points is $|x_1 - x_2|$.

The new version of the *checkHit* instance methods follows.

```
green.checkHit = function()
{                               // Assumes registration points are at centers of symbols
    if (square.hitTest(green))
    {
        if (Math.abs((green._x – green._width/2)
            – (square._x + square._width/2)) <= Math.abs(green.dx))
        {
            green.dx = -green.dx;               // right
        }
        else if(Math.abs((square._x – square._width/2)
            – (green._x + green._width/2)) <= Math.abs(green.dx))
        {
            green.dx = -green.dx;               // left
        }
        if (Math.abs((square._y – square._height/2)
            – (green._y + green._height/2)) <= Math.abs(green.dy))
        {
            green.dy = -green.dy;               // top
        }
        else if (Math.abs((green._y – green._height/2)
            – (square._y + square._height/2)) <= Math.abs(green.dy))
        {
            green.dy = -green.dy;               // bottom
        }
    }
}
```

Problem: We want to hear the green ball strike the blue square.

Adding Sound Objects

Sound is a predefined class in ActionScript.

Suppose we have a sound file, called *bonk.wav*, that should play whenever the green ball hits the blue square.

- Import the sound file:
  File ➜ Import ➜ Import to Library
- The sound can be previewed in the Library when selected.
- Set the linkage properties for the sound:
  Select the sound in the Library.
  Choose Linkage from the Options menu or the right-click menu.
  Check Export for ActionScript in the Linkage Properties window.
  This last step ensures that the sound is included in the SWF file even though it is not on the stage.

- Choose Properties from the Options menu or the right-click menu to alter the compression technique.
- Create a Sound object in the ActionScript code.

  ```
  snd = new Sound();
  ```

  Use one object for each different sound.
- Attach the Library sound to the Sound object using the following code.

  ```
  snd.attachSound("crash.wav");
  ```

  The string parameter to this method is the name of the Identifier set in the Linkage window, which is the file name by default. It can be altered.
- Play the sound using its *start* method.

  ```
  snd.start();
  ```

Changes to GreenBall.fla

```
onLoad = function()
{
    green.dx = 8;
    green.dy = 4;
    bonk = new Sound();
    bonk.attachSound("bonk.wav");
}

green.checkHit = function()
{
    if (square.hitTest(green))
    {
        bonk.start();
    // Rest of the method is the same.
```

Sound Instance Methods and a Property

```
stop()
getVolume()         // a number between 0 and 100
setVolume(n)
start(s)            // start at the s second of the sound
position            // number of milliseconds a sound has been playing
```

Example: MySounds.fla

Import the following sounds to the stage: *crash.wav* and *Huntingdon.mp3*.

At Keyframe 1 play the crash sound.

At Keyframe 10 (create it) make a Sound object for Huntingdon.mp3 and place five buttons on the stage: *start*, *stop*, *continu*, *louder*, *softer*.

Enter the follow code at Keyframe 10.

```
snd = new Sound();
snd.attachSound("Huntingdon.mp3");      // Remember Export for ActionScript
snd.setVolume(50);
stop();

start.onRelease = function()
{
    snd.stop();
    snd.start();
}

stop.onRelease = function()
{
    pos = snd.position/1000;
    snd.stop();
}

continu.onRelease = function()
{   snd.start(pos);   }

louder.onRelease = function()
{
    level = snd.getVolume();
    level = Math.min(100,level+5);
    snd.setVolume(level);
    trace(level);
}

softer.onRelease = function()
{
    level = snd.getVolume();
    level = Math.max(0,level-5);
    snd.setVolume(level);
    trace(level);
}
```

Using the Mouse

The MovieClip class contains a number of events that respond to frame events and mouse events.

Here are some that are not shared by the Button class.

- onLoad
- onEnterFrame
- onMouseDown
- onMouseUp
- onMouseMove

To use the mouse for drawing on the stage we use the following MovieClip methods.

clear()	Removes all graphics creadted during runtime using draw methods.
lineStyle(thickness : Number, rgb : Number, alpha : Number)	Specifies line style for subsequent drawing
moveTo(x : Number, y : Number)	Moves current drawing position to (x, y)
lineTo(x : Number, y : Number)	Draws a line from the previous drawing position to (x, y) using the current line style.

Example: Drawing.fla

Increase the Frame rate to 24 to reduce flicker.

The MovieClip properties, *xmouse* and *ymouse*, hold the current position of the mouse. Without an object specification, we get *_root*, the main timeline Movie Clip.

In a new Flash document, enter the following code at Keyframe 1.

```
onMouseDown = function()
{
      lineStyle(4, 0x0000ff, 100);
      moveTo(_xmouse, _ymouse);
}

onMouseUp = function()
{
      lineStyle(0, 0x000000, 0);
}

onMouseMove = function()
{
      lineTo(_xmouse, _ymouse);
}
```

Erasing the Drawing

As a preview of key stroke handling, we alter the onMouseDown handler to erase the current drawing if the mouse is clicked with the shift key depressed.

```
onMouseDown = function()
{
    if (Key.isDown(Key.SHIFT))
    {
        clear();
    }
    else
    {
        lineStyle(4, 0x0000ff, 100);
        moveTo(_xmouse, _ymouse);
    }
}
```

Mouse Class

Two class methods in Mouse make the mouse cursor disappear and reappear.

```
Mouse.hide();
Mouse.show();
```

Example: NewCursor.fla

Draw a small graphic, say a three-pointed start, on the stage and convert it to a Movie Clip Symbol.

Give the instance on the stage the name *pointer*.

Increase the Frame rate to 24.

Enter the following code at Keyframe 1.

```
Mouse.hide();
onEnterFrame = function()
{
    pointer._x = _xmouse;
    pointer._y = _ymouse;
}
```

Handling Key Strokes

ActionScript provides several ways to recognize key presses.

Events shared by MovieClip and Button Classes

onPress	onDragOver
onRelease	onDragOut
onReleaseOutSide	onKeyDown
onRollOver	onKeyUp

Example: MyEvents.fla

Place a blue rectangle MovieClip instance on the center of the stage and call it *box*.

To get the *box* object to recognize key presses, the focus needs to be set on the *box*.

Enter the following ActionScript code at KeyFrame 1.

```
box.useHandCursor = false;          // otherwise cursor changes to a hand over box
box.focusEnabled = true;
Selection.setFocus(box);

box.onRollOver = function()
{    trace('RollOver'); }

box.onRollOut = function()
{    trace('RollOut'); }

box.onReleaseOutside = function()
{    trace('ReleaseOutSide'); }

box.onRelease = function()
{    trace('Release');
     Selection.setFocus(box);
}

box.onDragOver = function()
{    trace('DragOver'); }

box.onDragOut = function()

{    trace('DragOut'); }

box.onPress = function()
{    trace('Press'); }

box.onKeyDown = function()
{    trace('KeyDown'); }

box.onKeyUp = function()
{    trace('KeyUp'); }
```

Key Class

The Key class contains several class methods.

Key.getAscii()	Returns ASCII code of last key pressed.
Key.getCode()	Returns key code value of last key pressed, ignoring modifying keys.
Key.isDown(code : Number)	Returns true if the key specified by the code is pressed (now).
Key.addListener(kl : Object)	Registers an object to receive *onKeyDown* and *onKeyUp* notifications.

Example: KeyPresses.fla

```
onEnterFrame = function()
{
    if (Key.isDown(65))
    {
        trace("Letter 'a' pressed");
    }
    else if (Key.isDown(Key.SPACE))
    {
        trace("Space pressed");
    }
    else if (Key.isDown(Key.UP))
    {
        trace("Up arrow pressed");
    }
}

keyListener = new Object();
keyListener.onKeyDown = function()
{
    trace("Key down");
}
keyListener.onKeyUp = function()
{
    trace("Key up");
    trace("   Ascii = " + Key.getAscii());
    trace("   Code = " + Key.getCode());
    trace("   " + String.fromCharCode(Key.getCode()));
}
Key.addListener(keyListener);
```

Since any object can serve as the key listener, we can write the last part of the code using the *_root* object as the key listener.

```
_root.onKeyDown = function()
{
      trace("Key down");
}
_root.onKeyUp = function()
{
      trace("Key up");
      trace("   Ascii = " + Key.getAscii());
      trace("   Code = " + Key.getCode());
      trace("   " + String.fromCharCode(Key.getCode()));
}
Key.addListener(_root);
```

The Key class has 18 predefined constants for special keys.

Key.BACKSPACE	8
Key.SPACE	32
Key.UP	38
Key.DOWN	40
Key.LEFT	37
Key.RIGHT	39
Key.SHIFT	16
Key.TAB	9
Key.ENTER	13
Key.CONTROL	17
Key.ESCAPE	27
Key.END	35
Key.CAPSLOCK	20
Key.DELETEKEY	46
Key.HOME	36
Key.INSERT	45
Key.PGDN	34
Key.PGUP	33

Example: Arrows.fla

Place an instance of a GreenBall Movie Clip Symbol on the stage and call it *green*.

In this example we want to move the ball left and right, up and down, under control of the four arrow keys on the keyboard.

When the ball reaches the edge of the stage, have it disappear and then reappear on the opposite edge of the stage.

Enter the following ActionScript code at KeyFrame 1.

```
onLoad = function()
{
    green.dx = 5;
    green.dy = 8;
}

onEnterFrame = function()
{
    if (Key.isDown(Key.LEFT))
    {
        newx = green._x-green.dx;
        if (newx <= 0)
        {   newx = Stage.width+newx;   }
        green._x = newx;
    }
    if (Key.isDown(Key.RIGHT))
    {
        newx = green._x+green.dx;
        if (newx >= Stage.width)
        {   newx = newx-Stage.width;   }
        green._x = newx;
    }
    if (Key.isDown(Key.UP))
    {
        newy = green._y-green.dy;
        if (newy <= 0)
        {   newy = Stage.height+newy;   }
        green._y = newy;
    }
    if (Key.isDown(Key.DOWN))
    {
        newy = green._y+green.dy;
        if (newy >= Stage.height)
        {   newy = newy-Stage.height;   }
        green._y = newy;
    }
}
```

Array Class

Arrays in ActionScript are nonhomogenous since you can put values of any type into them.

They are dynamic in the sense that they can grow at runtime.

There appears to be no way to type the elements of an array as a means of getting strong type checking for array elements.

Array Creation

```
var a : Array = new Array();           // an empty array
b = new Array(4);                      // an array with 4 undefined values
c = [1, 2, 3, 4, 5];                   // an array literal
d = ["abc", 2+2, Math.PI, 'xyz'];      // arrays can be nonhomogeneous
e = new Array(["a", "b", "c"]);
f = [[1,2,3],[4,5,6]];                 // a two dimensional array
g = new Array(['A','B'], ['Y','Z']);   // a two dimensional array
```

```
                          Output (result of toString())
trace(a);                 none
trace(b);                 undefined,undefined,undefined,undefined
trace(c);                 1,2,3,4,5
trace(d);                 abc,4,3.14159265358979,xyz
trace(e);                 a,b,c
trace(f);                 1,2,3,4,5,6
trace(g);                 A,B,Y,Z

for (k=0; k<b.length; k++)
{
      b[k] = (k+1)*(k+1);
 }
trace(b);                 1,4,9,16
b[6] = -99;
trace(b.toString());      1,4,9,16,undefined,undefined,-99
```

Regular Expressions

x^*	Zero or more copies of the item x.
x^+	One or more copies of the item x.
$x^?$	One copy of x or none at all.

In the following definitions, the regular expression modifiers apply to the entire parameter.

Array Instance Methods

push(v : Object$^+$) : Number	Adds one or more elements to the end of this array and returns the new length of the array.
pop() : Object	Removes the last element from this array and returns the value of that element.
unshift(v : Object$^+$) : Number	Adds one or more elements to the beginning of this array and returns the new length of the array.
shift() : Object	Removes the first element from this array and returns the value of that element.
reverse() : Void	Reverses this array in place.
slice(startindex : Number$^?$, endindex : Number$^?$) : Array	Returns a new array consisting of a range of elements from this array without modifying this array; the returned array includes the *startindex* element and all elements up to but not including the *endindex* element. Negative index values are measured from the end of the array.
toString() : String	Returns a string containing all of the elements of this array separated by commas.
concat(v : Object*) : Array	Concatenates the elements specified by parameter to the end of this array and returns it without changing this array.
join(delimiter : String$^?$) : String	Returns a string containing all of the elements of this array separated by the delimiter or by commas if no delimiter.
splice(startindex : Number, deleteCount : Number$^?$, v : Object$^?$) : Array	Adds element to and removes elements from this array by modifying the array; *deleteCount* is the number of elements removed at the *startindex* position; the elements specified by the third parameter are inserted at this point; the return value is an array of the elements removed; if the *deleteCount* parameter is omitted, all elements from *startindex* to the end are delete.

Stack Behavior

Use *push* and *pop*.

Queue Behavior

Use *push* and *shift*.

Examples

```
h = c.concat();
i = c.concat(e,e);
trace(h);                1,2,3,4,5
trace(i);                1,2,3,4,5,a,b,c,a,b,c
trace(c);                1,2,3,4,5

trace(c.join("; "));     1; 2; 3; 4; 5

trace(c.reverse());      5,4,3,2,1
trace(c);                5,4,3,2,1

x = ["bat", "cat", "elk", "emu", "fox",  "owl","rat", "yak"];
y = x.slice(1,4);
trace(y);                cat,elk,emu
trace(x);                bat,cat,elk,emu,fox,owl,rat,yak

z = x.splice(2, 3);
trace(x);                bat,cat,owl,rat,yak
trace(z);                elk,emu,fox
w = x.splice(3, 0, z);
trace(x);                bat,cat,owl,elk,emu,fox,rat,yak
trace(w);                none
```

The *slice* and *splice* operations did not always work correctly for me when applied to the array resulting from an application of *concat*.

MovieClip Properties

These public instance variables in the MovieClip class are accessible to each MovieClip object and to objects of a subclass of MovieClip.

The list contains only the most useful properties.

_width : Number	_name : String
_height : Number	_rotation : Number
_x : Number	_alpha : Number (0..100)
_y : Number	_parent : MovieClip
_xscale : Number (nonnegative)	_xmouse : Number (read-only)
_yscale : Number (nonnegative)	_ymouse : Number (read-only)
_visible : Boolean	

Vector Motion

An object moving on the stage has speed and direction, which are represented by the concept of a vector.

A vector can be viewed as an arrow whose angle gives the direction and whose length represents the speed.

Problem: Translate a vector into the *dx*, *dy* values that are used to specify motion in Flash.

XY (Cartesian) Coordinate System

The origin (0,0) of the system is the upper left corner. The positive x axis moves from left to right, and the positive y axis moves from top to bottom.

Angles are measured clockwise from the positive x direction.

Right	0°
Down	90°
Left	180°
Up	270°

But angles in Flash are computed clockwise from the North direction in the Transform panel and with the *_rotation* property of MovieClip objects.

obj._rotation = 0	North
obj._rotation = 90	East
obj._rotation = 180	South
obj._rotation = 270	West

We need to convert a Flash angle into the XY-coordinate system.

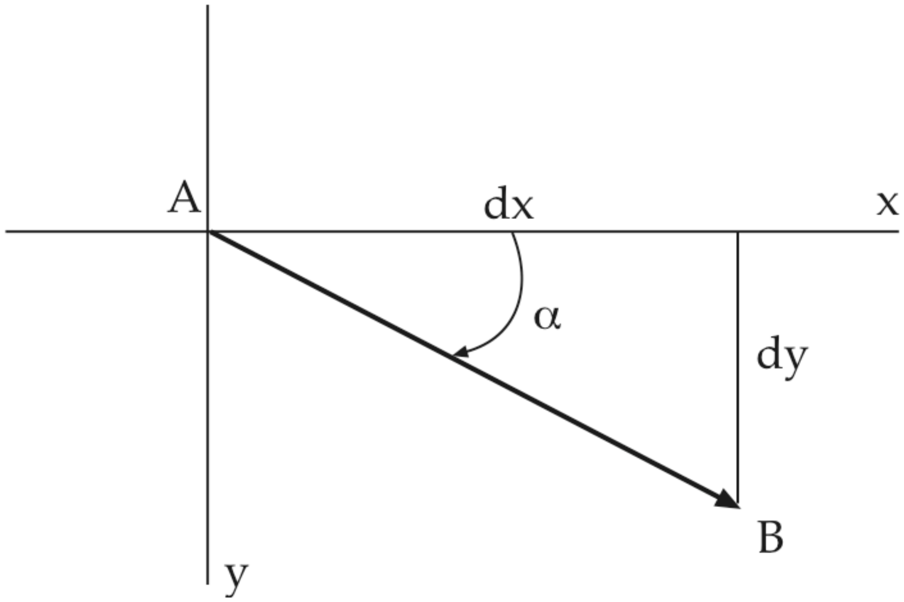
Let *flang* be the Flash angle.

Let *degrees* be the mathematical angle in the XY-coordinate system.

Then

degrees = flang - 90;

Consider a vector in the XY-coordinate system.



The motion is from A to B.

Vector: angle = α

speed = length of vector = distance from A to B.

Translate to dx, dy

sin α = dy/speed
cos α = dx/speed

dx = speed • cos α
dy = speed • sin α

Problem: Math.sin and Math.cos in ActionScript expect angles to be in radians.

Conversion

180° is equivalent to π radians

degrees/180 = radians/ π

radians = π • degrees / 180

Suppose a MovieClip object has the following properties:

    speed
    direction (in degrees)
    dx
    dy

Suppose the *speed* and *direction* are known. We want to compute the *dx* and *dy*.

```
function turn(mc : MovieClip)
{
        degrees = mc.direction - 90;
        radians = Math.PI * degrees / 180;     // convert to radians
        mc.dx = mc.speed * Math.cos(radians);
        mc.dy = mc.speed * Math.sin(radians);
}
```

Problem: We turn the object by altering the direction property, say +5 or -5.

If we want to display the direction, we want the value to be between 0° and 360°.

Each time we add or subtract 5, the result should be calculated modulo 360, which is the positive remainder on dividing by 360.

Note: The sine and cosine functions don't care about the magnitude of the angle; they work just as well with negative angles and angles above 360°.

Problem: The remainder operator (%) in ActionScript does not compute the modulo function correctly.

Example: Divide by 4

n	**n%4**	**mod(n, 4)**
5	1	1
4	0	0
3	3	3
2	2	2
1	1	1
0	0	0
-1	-1	3
-2	-2	2
-3	-3	1
-4	0	0
-5	-1	3

Solution: When the remainder is negative, add the divisor to it.

```
function mod(num, div)
{
        r = num % div;
        return r<0 ? r+div : r;
}
```

Example: TurningCar.fla

Create a small MovieClip object that looks like a car seen from above.

Place it on the stage pointing NorthEast with the instance name *car*.

When the *car* object reaches the edge of the stage, it continues on the opposite edge.

Control:		
	Left arrow	Rotate car 5° counterclockwise.
	Right arrow	Rotate car 5° clockwise.
	Up arrow	Increase speed by 1.
	Down arrow	Decrease speed by 1.

The car has a maximum speed of 15. Of course, the minimum speed is 0.

Enter the following ActionScript code at KeyFrame 1.

```
car.speed = 0;
car.direction = 45;              //  degrees
car.maxSpeed = 15;

onEnterFrame = function()
{
      checkKeys();
      turn(car);
      move(car);
}

function checkKeys()
{
     if (Key.isDown(Key.UP))
     {
          car.speed = Math.min(car.speed+1, car.maxSpeed);
     }
     if (Key.isDown(Key.DOWN))
     {
          car.speed = Math.max(car.speed-1,0);
     }
     if (Key.isDown(Key.RIGHT))
     {
          car.direction = mod(car.direction+5, 360);
     }
     if (Key.isDown(Key.LEFT))
     {
          car.direction = mod(car.direction-5, 360);
     }
}
```

```
function turn(mc : MovieClip)
{
        degrees = mc.direction - 90;
        radians = Math.PI * degrees / 180;
        mc.dx = mc.speed * Math.cos(radians);
        mc.dy = mc.speed * Math.sin(radians);
}

function move(mc : MovieClip)
{
        mc._x = mc._x + mc.dx;
        mc._y = mc._y + mc.dy;

        mc._rotation = mc.direction;

        if (mc._x > Stage.width)
        {    mc._x = 0;  }

        if (mc._x < 0)
        {    mc._x = Stage.width;  }

        if (mc._y > Stage.height)
        {    mc._y = 0;  }

         if (mc._y < 0)
        {    mc._y = Stage.height;  }
}

function mod(num, div)
{
        r = num % div;
        return r<0 ? r+div : r;
}
```

Classes, Objects, and Dynamic Properties

Put the following ActionScript code in a file named *Box.as*.

```
class Box
{   }
```

Put this code in a Flash document.

```
b = new Box();
b.size = 10;                    // illegal
```

We cannot add a property to a Box object dynamically in this case.

Rewrite the Box class as follows.

```
dynamic class Box
{   }
```

Put this code in a Flash document.

```
b = new Box();
b.size = 10;          // legal now
```

When a class has the **dynamic** modifier, properties may be added to its objects outside of its class definition.

Recommendation: Dynamic classes are considered poor OOP design, although this usage is found in Flash code frequently. A better approach would be to use a subclass of the class to add a property.

The MovieClip, Array, and Object classes are defined as **dynamic**.

Subclasses of dynamic classes are normally **dynamic**.

But subclasses of MovieClip and Object are *not* **dynamic**, by default.

Kinds of Variables in Classes

property	An instance variable defined in the class and found in each object of the class.
class property	A class variable defined in the class and belonging to the class itself.
dynamic property	An instance variable defined outside of the class definition; it belongs to only one object, the one it is defined for.

Visibility Modifiers

public and **private**

No modifier on a member of a class means that it is **public**.

ActionScript does not allow *final* as a modifier.

Viewing Properties

Change the Box class to read as follows.

```
dynamic class Box
{
    var width : Number = 25;
    var height : Number = 50;
    var description : String = "red box";
}
```

Consider this ActionScript code in a Flash document.

```
b = new Box();
trace(b.width);       // prints 25
trace(b.height):      // prints 50
```

Alternate syntax

```
trace(b["width"]);      // prints 25
fld = "height";
trace(b[fld]);          // prints 50
```

Contrast Syntax

With the syntax *b.width* and *b.height*, the field (property) name must be static, which means it is known to the compiler).

With the syntax *b["width"]* and *b[fld]*, the field (property) name can be dynamic, which means it may not be known until runtime.

Object Literals

ActionScript allows literal expressions that represent Object objects with dynamic properties belong to this new object only.

```
var ob : Object = {x:33, y:44, color:"blue", visible:false};
```

Displaying the Properties

```
for (prop : String in ob)
{
    trace("------------------------------------------");
    trace("Property name: " + prop);
    trace("Property value: " + ob[prop]);
}
trace("------------------------------------------");
```

Output

```
----------------------------------------
Property name: x
Property value: 33
----------------------------------------
Property name: y
Property value: 44
----------------------------------------
Property name: color
Property value: blue
----------------------------------------
Property name: visible
Property value: false
----------------------------------------
```

Note: The *ob[prop]* syntax works only with dynamic properties.

Array Example

```
playOffs = new Array();
playOffs.push("Red Sox", "Indians", "Rockies", "Diamondbacks");
for (team in playOffs)
{
      trace(playOffs[team]);
}
```

Create a MovieClip Instance Dynamically

Create a MovieClip Symbol called Ball.

In the Properties window (Option menu or right-click in the Library Panel), select
Export for ActionScript

Use the MovieClip instance method *attachMovie* on the *_root* object.

```
public attachMovie(symbolLinkage : String,        // name of the class (symbol)
                   instanceName : String,
                   depth : Number,
                   initObject : Object? ) : MovieClip
```

The *depth* parameter is an integer that describes the level at which the instance will be placed. Each instance on the stage must lie on a unique level. Either maintain a counter variable to ensure that different instances created dynamically lie on difference levels or use the instance method *getNextHighestDepth*.

The *initObject* parameter can be used to initialize the instance coming into existence, say by an anonymous object literal such as *{ _x : 100, _y : 150 }*.

Removing a MovieClip

Global class method

```
removeMovieClip(instanceName);
```

MovieClip instance method

```
instanceName.removeMovieClip():
```

Returning to the example

Create an instance of Ball called *theBall*.

```
_root.attachMovie("Ball", "theBall", 10);  or  attachMovie("Ball", "theBall", 10);
```

Position the ball.

```
theBall._x = 200;
theBall._y = 130;
```

eval Function

This function can be used to build an identifier dynamically that refers to an instance.

```
eval(expr : String) : Object
```

If *expr* is a variable identifier or a property identifier (as a String), the value of the variable or property is returned as the result.

If the item named in *expr* cannot be found, *undefined* is returned.

Note that the *eval* function only returns an R-value.

Suppose we have three MovieClip objects, *box1*, *box2*, and *box3*.
Rotate each to 45°.

```
for (k=1; k<=3; k++)
{
      eval("box" + k)._rotation = 45;
}
```

Other methods can be used to create an L-value dynamically.

Examples

```
k = 5;
var5 = "first";                   // these four commands have the same effect
_root["var" + k] = "first";
this["var" + k] = "first";        // assuming code in on main timeline
set("var" + k, "first");
```

Example: MakeBalls.fla

Create a MovieClip Symbol called Ball (a green disk).

Create a Button Symbol and place an instance of it on the stage with the label "Make a Ball" and with instance name *theButton*.

Place a dynamic text field on the stage with instance name *display*.

Enter the following ActionScript code at KeyFrame 1.

```
k = 0;
theButton.onRelease = function()
{
      k++;
      _root.attachMovie("Ball", "ball"+k, 10*k);
      currentBall = eval("ball"+k);
      w = currentBall._width;
      h = currentBall._height;
      currentBall._x = Math.random()*(Stage.width-w)+w/2;
      currentBall._y = Math.random()*(Stage.height-h)+h/2;
      currentBall.index = k;
```

```
		currentBall.onRelease = function()
		{
			display.text = "Last ball clicked: Index = " + this.index;
			this.removeMovieClip();
		}
	}
```

Notes

- Each disk is positioned on the stage using the identifier *currentBall*. The random calculations for *_x* and *_y* ensure that the disk lies entirely on the stage.
- The identifier *index* is a new property created dynamically for each instance.
- A MovieClip can act in the same way as a Button, responding to a click using the *onRelease* event.
- When a MovieClip disk is clicked on, it disappears and its *index* value is reported in the dynamic text field.

ActionScript Code for a Symbol

Create a MovieClip Symbol named Ball.

We can provide properties and behavior for Ball objects using a class definition that specifies Ball as a subclass of MovieClip.

Enter the following ActionScript code into a file named *Ball.as*.

```
	class Ball extends MovieClip
	{
		private var dx:Number,
				dy:Number,
				xspeed:Number,
				yspeed:Number;
		function Ball()
		{
			_x = Math.random()*(Stage.width-_width)+_width/2;
			_y = Math.random()*(Stage.height-_height)+_height/2;
			dx = xspeed=Math.random()*20-10;		// -10 ≤ dx < 10
			dy = yspeed=Math.random()*20-10;		// -10 ≤ dy < 10
		}
		function onEnterFrame()
		{
			move();
			checkBounds();
		}
```

```
        function move()
        {
                _x = _x+dx;
                _y = _y+dy;
        }
        function stop()
        {   dx = dy= 0;   }


        function start()
        {
                dx = xspeed;
                dy = yspeed;
        }
        function checkBounds()
        {
                if (_x + _width/2 >= Stage.width)
                {    dx = -dx;   }
                if (_x - _width/2 <= 0)
                {   dx = -dx;   }
                if (_y + _height/2 >= Stage.height)
                {   dy = -dy;   }
                if (_y - _height/2 <= 0)
                {   dy = -dy;   }
        }
        function onRelease()
        {
                if (Key.isDown(Key.SPACE))
                {   removeMovieClip(this);   }
                else if (Key.isDown(Key.UP))
                {   start();   }
                else if (Key.isDown(Key.DOWN))
                {  stop();   }
        }
}
```

Using Ball.as

Create a Flash document *ManyBall.fla* with following code at KeyFrame 1.

```
for (k = 0; k< 50; k++)
{
        attachMovie("Ball", "ball" +k, k + 10);
}
```

Gravity

Gravity is a constant force acting to accelerate the downward velocity (*dy*) of an object over time.

Since the positive direction of *y* is down, we simply need to add a positive constant to the downward velocity *dy* in each frame, the measure of time in Flash.

Acceleration = Change in Velocity

Velocity = Change in Position

Relationships

position = velocity • time + initialPosition

velocity = acceleration • time + initialVelocity

Consider two points in time, t_1 and t_2.

$p_2 = v \bullet t_2 + \text{initialPosition}$

$p_1 = v \bullet t_1 + \text{initialPosition}$

Subtract

$p_2 - p_1 = v \bullet (t_2 - t_1)$

Let $t_2 = t_1 + 1$.

$p_2 = p_1 + v$

This corresponds to the command _x = _x + dx.

$v_2 = a \bullet t_2 + \text{initialVelocity}$

$v_1 = a \bullet t_1 + \text{initialVelocity}$

Subtract

$v_2 - v_1 = a \bullet (t_2 - t_1)$

Let $t_2 = t_1 + 1$.

$v_2 = v_1 + a$

This corresponds to the command dx = dx + a.

Note: Both velocity and acceleration may be functions of time and not just constants.

Example: Gravity.fla

Create a MovieClip Symbol Ball and place one instance of it on the stage with the name *ball*.

Enter the following ActionScript code at KeyFrame 1.

```
ball.dx = 10;
ball.dy = 5;

gravity = 3;
```

```
onEnterFrame = function()
{
	ball._x = ball._x + ball.dx;
	ball._y = ball._y + ball.dy;
	if (ball._x + ball._width/2 > Stage.width)
	{
		ball._x = Stage.width - ball._width/2;
		ball.dx = -0.9*ball.dx;
	}
	if (ball._x - ball._width/2 < 0)
	{
		ball._x = 0 + ball._width / 2;
		ball.dx = -0.9*ball.dx;
	}
	if (ball._y + ball._height/2 > Stage.height)
	{
		ball._y = Stage.height- ball._height/2;
		ball.dy = -0.9*ball.dy;
	}
	if (ball._y - ball._height/2 < 0)
	{
		ball._y = 0 + ball._height/2;
		ball.dy = -0.9*ball.dy ;
	}
	ball.dy = ball.dy + gravity;
}
```

Notes

- This program is similar to previous bouncing ball programs, but with two modifications to make it more realistic.
- When the ball bounces off a surface, its reversed speed is only 90% of the impact speed to model the fact that balls lose speed when they bounce.
- With each passing frame, the velocity in the *y* direction (*dy*) is modified by adding the *gravity* constant to model the decreasing velocity when the ball is rising and the increasing velocity when the ball is descending.
- The actual value of the *gravity* constant cannot be determined physically since we are modeling the world in Flash. Try different values for the constant until the simulation looks good.

Example: GravityTrace.fla

Create a MovieClip Symbol in the shape of a cannon and place an instance of it in the lower left corner of the stage with the instance name *gun*.

Create another MovieClip Symbol that is a small black disk and place an instance of it behind the cannon with the instance name *bullet*.

Place two dynamic text fields near the bottom edge of the state with the variable names (Var: field) *bullet.direction* and *bullet.speed*.

Enter the following ActionScript code at KeyFrame 1.

```
gun.direction = 0;
gun.charge = 0;
gravity = 0.5;
onEnterFrame = function()
{
	checkKeys();	move(bullet);
}
function checkKeys()
{
	if (Key.isDown(Key.LEFT))
	{
		clear();
		gun._rotation = gun.direction = gun.direction - 2;
	}
	if (Key.isDown(Key.RIGHT))
	{
		clear();
		gun._rotation = gun.direction = gun.direction + 2;
	}
	if (Key.isDown(Key.UP))
	{
		clear();
		gun.charge = gun.charge + 5;
	}
	if (Key.isDown(Key.DOWN))
	{
		clear();
		gun.charge = gun.charge - 5;
	}
	if (Key.isDown(Key.SPACE))
	{
		bullet._x = gun._x;
		bullet._y = gun._y;
```

```
                bullet.speed = gun.charge;
                bullet.direction = gun.direction;
                clear();
                lineStyle(2,0x000000,100);
                moveTo(gun._x, gun._y);
                turn(bullet);
            }
    }

    function turn(mc)
    {
            degrees = mc.direction -90;
            radians = degrees / 180 * Math.PI;
            mc.dx = mc.speed*Math.cos(radians);
            mc.dy = mc.speed*Math.sin(radians);
    }
    function move(mc)
    {
            mc._x = mc._x + mc.dx;
            mc._y = mc._y + mc.dy;
            _root.lineTo(mc._x, mc._y);
            if ((mc._x > Stage.width) || (mc._x < 0) || (mc._y > Stage.height) )
            {
                mc._x = -100;                //stop mc and move it off stage
                mc._y = -100;
                mc._speed = 0;
                mc.dx = 0;
                mc.dy = 0;
                lineStyle(0,0x000000,0);     //turn off line drawing
            }
            mc.dy = mc.dy + gravity;
    }
```

Notes

- A line drawing traces the path of the cannon ball using the methods *lineStyle*, *moveTo*, and *lineTo*.
- The line drawing that traces the cannon ball path is erased (*clear*) whenever a key is recognized by the code.
- If the cannon passes through the left edge, the right edge, or the bottom of the stage, it is moved to position (-100, -100), which is off the stage, and it is stopped.
- The top edge of the stage is no barrier at all. The cannon ball passed off the top of the stage and then returns if it has not met one of the edges.

Using Dynamically Created Objects and an Array

Example: CursorTrail.fla

Create a MovieClip Symbol called Cursor that is a small red disk with a black border.

Enter the following ActionScript code at KeyFrame 1.

```
onLoad = function()
{
    for (var k=0; k<10; k++)           // create 10 Cursor objects
    {
        attachMovie("Cursor", "cursor"+k, k);
    }
    trail = new Array();               // to hold mouse positions
}
onEnterFrame = function()
{
    // mark the mouse location
    cursorLoc = { x : _xmouse,  y : _ymouse };     // an anonymous object
    trail.push(cursorLoc);
    if (trail.length > 10) trail.shift();          // keep only last 10 positions
    for (var k=0; k<trail.length; k++)         // change the positions of cursor followers
    {
        _root["cursor"+k]._x = trail[k].x;
        _root["cursor"+k]._y = trail[k].y;
        _root["cursor"+k]._alpha = (k+1)*10;    // decreasing transparency
    }
}
```

Global Function: setInterval

The *setInterval* function calls a method at periodic intervals while a Flash movie plays.

Two versions

```
setInterval(functionRef : Function, interval : Number) : Number
setInterval(objectRef : Object, methodName : String, interval : Number) : Number
```

interval	time in milliseconds.
return value	an identifying number that can be passed to the method *clearInterval* to cancel the operation.

In the next example, a method in a class definition contains a nested method inside of itself. This nesting of methods cannot be done in Java or other languages based on C.

Example: TimeTracer.as

Create an ActionScript file containing the following code.

The method *startTimeDisplay* takes a reference to a dynamic text field as its parameter.

Note that ***new** Date().toString()* returns a string of the form:

```
Tue Nov 27 15:10:45 GMT-0600 2007
```

The displayTime method puts a new string of the form `Tue Nov 27 15:10:45` into the dynamic text field every second.

```
class TimeTracer
{
    private var timerID : Number;
    public function startTimeDisplay(disp) : Void
    {
        var begunAt : String = new Date().toString();
        trace("Timer started at " + begunAt);
        timerID = setInterval(displayTime, 1000);
        function displayTime() : Void
        {
            var s : String = new Date().toString();
            var n : Number = s.indexOf("GMT");
            disp.text = s.substring(0, n);
        }
    }
    public function stopTimeDisplay() : Void
    {
        clearInterval(timerID);
    }
}
```

Using TimeTracer: TryTimeTracer.fla

Create a dynamic text field, centered on the stage, with the instance name *timeField*.

Use a large font for the field.

Enter the following code at KeyFrame 1:

```
var tt : TimeTracer = new TimeTracer();
tt.startTimeDisplay(timeField);
```

Enter the following code at KeyFrame 100:

```
tt.stopTimeDisplay();
stop();
```

Using TimeTracer for an Analogue Clock

Add two methods to *TimeTracer.as*.

```
        public function getHour() : Number
        {
            return new Date().getHours();    // note names of Date methods
        }
        public function getMinute() : Number
        {
            return new Date().getMinutes();
        }
```

Flash Document: Clock.fla

Draw a clock face on the stage. Create arrow shaped MovieClip Symbols for the minute hand (*minuteHand*) and the hour hand (*hourHand*) with the rotation points at the base of the arrows.

Enter the following ActionScript code at KeyFrame 1.

```
    var tt : TimeTracer = new TimeTracer();

    tt.startTimeDisplay(null);

    onEnterFrame = function()
    {
        minute = tt.getMinute();
        minuteHand._rotation = minute*6;
        hourHand._rotation = (tt.getHour()%12)*30 + minute/2 ;
    }
```

Using Frame Count for Timing: WordServer.fla

In this example, the words of a sentence are displayed on the screen one at a time with a uniform delay.

Place a dynamic text field at the center of the stage using a large font, with the instance name *display*.

Enter the following ActionScript code at KeyFrame 1.

```
    onLoad = function()
    {
        sentence = "Imagination is more important than knowledge";
        wordList = sentence.split(" ");           // produces an array of strings
        wordNum = 0;
        frameDelay = 12;
        frameCount = frameDelay;                  // prime for first word
    }
```

```
onEnterFrame = function()
{
	if (frameCount == frameDelay)
	{
		display.text = wordList[wordNum];
		wordNum++;
		if (wordNum >= wordList.length)
		{
			wordNum = 0;
		}
		frameCount = 0;
	}
	frameCount++;
}
```

With a *frameDelay* value of 12 and the Frame Rate set to 12 frames per second, the word displayed changes every second.

Note that the variable *wordNum* cycles through the values 0, 1, 2, 3, 4, 5 repeatedly, and the variable *frameCount* cycles through the values 12, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 repeatedly.

Both of these cycles can be calculated using the modulo operator, as shown below.

```
onLoad = function()
{
	sentence = "Imagination is more important than knowledge";
	wordList = sentence.split(" ");
	wordNum = 0;
	frameDelay = 12;
	frameCount = 0;
}
onEnterFrame = function()
{
	if (frameCount % frameDelay == 0)
	{
		display.text = wordList[wordNum];
		wordNum = (wordNum+1) % wordList.length;
	}
	frameCount++;
}
```

Creating Movie Clips Dynamically

We have drawn on the main Timeline Movie Clip using the *lineStyle*, *moveTo*, and *lineTo* methods.

Now we create a new Movie Clip on which to draw.

MovieClip Methods

createEmptyMovieClip(instanceName : String, depth : Number) : MovieClip

This method creates an empty MovieClip object as a child of an existing MovieClip, the receiver of the method, giving it the name *instanceName*.

It returns the new MovieClip whose registration point is in its upper-left corner.

Drawing Methods

Some of these methods have been described earlier.

lineStyle(thickness : Number, rgb : Number, alpha : Number) : Void

moveTo(x : Number, y : Number) : Void

lineTo(x : Number, y : Number) : Void

curveTo(ctrlX : Number, ctrlY : Number, anchorX : Number, anchorY :Number) : Void

This last method draws a curve from the previous drawing position to the point (anchorX, anchorY) using the point (ctrlX, ctrlY) to shape the curve.

beginFill(rgb : Number, alpha : Number$^?$) : Void

This method established a fill color for a closed shape that is drawn subsequently. Note that the figure automatically closes itself if we do not close it.

endFill() : Void

This method marks the end of the drawing of the shape to be filled.

Note: ActionScript has other fill methods, *beginGradientFill* and *beginBitmapFill*

Example: Fill.fla

Enter the following ActionScript code at KeyFrame 1.

```
createEmptyMovieClip("mc", 10);
mc.lineStyle(4, 0x000000, 100);
mc.beginFill(0xff6600);
    mc.moveTo(50, 50);
    mc.lineTo(150, 50);
    mc.lineTo(150, 200);
    mc.lineTo(50, 200);
mc.endFill();
```

with Command

The with command specifies an object in whose context a group of commands is executed.

This context may apply to variables and methods in the group of commands.

```
with (obj : Object)
{
    commands
}
```

Example: FillWith.fla

Enter the following ActionScript code at KeyFrame 1.

```
createEmptyMovieClip("mc", 10);
with (mc)
{
    lineStyle(4, 0x000000, 100);
    beginFill(0xff00cc);
        moveTo(50, 50);
        lineTo(150, 50);
        lineTo(150, 200);
        lineTo(50, 200);
    endFill();
}
```

Example: Lines.fla

Enter the following ActionScript code at KeyFrame 1.

```
createEmptyMovieClip("lines", 10);
lines.moveTo(Stage.width/2, Stage.height/2);
lines.lineStyle(6, 0xffff00, 100);
onMouseDown = function()
{
    lines.lineTo(_xmouse, _ymouse);
}
onEnterFrame = function()          // moves the MovieClip object
{
    if (Key.isDown(Key.SPACE))
    {
        lines._x = lines._x + 5;
        lines._y = lines._y + 5;
    }
}
```

Example: Curves.fla

Enter the following ActionScript code at KeyFrame 1.

```
createEmptyMovieClip("curves", 10);
var frameCount = 0;
onEnterFrame = function()
{
    color = (100000*frameCount) % 0x1000000;      // 0x000000 to 0xffffff
    with (curves)
    {
        lineStyle(2, color, 100);
        moveTo(100, 200);
        curveTo(_xmouse, _ymouse, 450, 200);       // curve from (100,200) to (450,200)
    }
    frameCount++;
}
onMouseUp = function()
{
    curves.clear();
}
```

Example: Snowflakes.fla

Set the background of the stage to a deep blue.

Enter the following ActionScript code at KeyFrame 1.

```
numSnowflakes = 50;
for (var k=0; k<numSnowflakes; k++)
{
    mc = createSnowflake(k);
    with (mc)
    {
        _x = Math.random()*Stage.width;             // 0 to Stage.width
        _y = Math.random()*Stage.height;            // 0 to Stage.height
        mc.speed = Math.random()*2 + 4;             // 2 to 6
        mc.drift = Math.random()*3 - 1.5;           // -1.5 to 1.5
        mc.rotate = Math.random()*18 - 9;           // -9 to 9
    }
}
onEnterFrame = function()
{
    moveSnowflakes();
}
```

```
function createSnowflake(n)
{
        createEmptyMovieClip("snowflake"+n, n);
        mc = this["snowflake"+n];
        mc.lineStyle(2, 0xffffff, 50);                    // white, half transparency
        numSpikes = Math.floor(Math.random()*5)+5;       // 5..9
        spikeLength = Math.random()*5+5;                 // 5 to 10

        for (var k=0; k<numSpikes; k++)
        {                          // create each spike as line from center to point on circle
                mc.moveTo(0,0);
                spikeAngle = 2.0*Math.PI*k/numSpikes;
                x = spikeLength*Math.cos(spikeAngle);
                y = spikeLength*Math.sin(spikeAngle);
                mc.lineTo(x,y);
        }
        return mc;
}
function moveSnowflakes()
{
        for (var k=0; k<numSnowflakes; k++)
        {
                mc = this["snowflake"+k];              // mc = eval("snowflake"+k);
                with (mc)
                {
                        _x = _x + drift;
                        _y = _y + speed;
                        _rotation = _rotation + rotate;
                        if (_y > Stage.height) _y = 0;    // bring back to top
                        if (_x < 0) _x = Stage.width;     // one side to another
                        if (_x > Stage.width) _x = 0;
                }
        }
}
```

Observe that when the instance variables *speed*, *drift*, and *rotate* are first introduced in the first *with* command, they must be fully qualified with the identifier *mc* to bring them into existence.

Duplicating a Movie Clip

The class MovieClip has an instance method that can be used to duplicate a MovieClip object, the receiver of the method.

```
duplicateMovieClip(instanceName : String,
                        depth : Number, initObj : Object?) : MovieClip
```

Note: There are two global functions for duplicating a MovieClip object, which should not be confused with this instance method.

Example: Duplicate.fla

Enter the following ActionScript code at KeyFrame 1.

```
createEmptyMovieClip("box", 0);
var w : Number = 400;
var h : Number = 20;
with (box)
{
    lineStyle(4, 0x0000, 100);
    beginFill(0x66ffff);
        moveTo(50, 40);
        lineTo(50+w, 40);
        lineTo(50+w, 40+h);
        lineTo(50, 40+h);
        lineTo(50, 40);
    endFill();
}
trace(box);

var spacer : Number = 5;
var duplicate : MovieClip;
for (var k:Number=1; k<=10; k++)
{
    var newY : Number = k*(box._height + spacer);
    duplicate = box.duplicateMovieClip("clip"+k, k, { _y : newY });
    trace(duplicate);
}
```

Example: Particle.fla

Create a Movie Clip Symbol as a 80 by 80 pixel disk at the center of the stage with a white to gray radial gradient.

Call the instance on the stage *particle*.

Enter the following ActionScript code at KeyFrame 1.

```
for (var k : Number = 1; k<=10; k++)
{
    particle.duplicateMovieClip("particle"+k, k);
}

onEnterFrame = function()
{
    xDist = _xmouse - Stage.width/2;      // distances from center to mouse

    yDist = _ymouse - Stage.height/2;

    for (var k : Number = 1; k<=10; k++)
    {
        pt = _root["particle"+k];

        // move particle k tenths of the distance from center to the mouse
        pt._x = Stage.width/2 + xDist*k/10;
        pt._y = Stage.height/2 + yDist*k/10;
    }
}
```

Loading and Sending Data

A LoadVars object can be used to read data values from a text file, similar to a property list, and can be used to send parameters to a server using GET and POST.

Loading Data

Create a text file composed of name-value pairs delimited by ampersands (&).

The data should be URL-encoded (spaces replaced by + and many symbols represented by their hexadecimal values in the form %2B).

Example File: data

```
username=Ken&passwd=lalala&color=blue&message=Hello&
```

This file may be placed on the same computer as the flash document or may be place on a web server.

Steps for Loading Data

1. Create a LoadVars object.

```
lv = new LoadVars();
```

2. Load the information.

```
lv.load("data");                         // local file
lv.load("http://www.cs.uiowa.edu/~slonnegr/flash/data");     // web site
```

3. Notification of completed loading.

```
lv.onLoad = function(success)
{   trace("Load Complete: " + success);   }
```

   The Boolean parameter *success* will be *true* if the load was successful.

4. Display the values.

   The names in the data file become instance variables in the LoadVars object *lv*.

```
    trace(lv.username);
```

Example: LoadVars.fla

Place two buttons on the stage with the labels LOADVARS and SHOWVARS and the instance names *load* and *show*.

Enter the following ActionScript code at KeyFrame 1.

```
var ready = false;
load.onRelease = function()
{
    lv = new LoadVars();
    lv.load("data");         // OR
    //  lv.load("http://www.cs.uiowa.edu/~slonnegr/flash/data");
    lv.onLoad = function(success)
    {
        trace("Load Complete: " + success);
        ready = success;
    }
}
show.onRelease = function()
{
    if (ready)
    {
        trace(lv.message);     trace(lv.color);
        trace(lv.passwd);      trace(lv.username);
    }
    else trace("Variables not loaded yet.");
}
```

Output When Buttons Pressed in Correct Order

```
Load Complete: true
Hello
blue
lalala
Ken
```

Output When Buttons Pressed in Reverse Order

```
Variables not loaded yet.
```

The load is "successful" even if the data file has an incorrect file. It is unsuccessful only if the file is missing.

Output When LOADVARS Button Pressed and File Does Not Exist

```
Load Complete: false
Error opening URL "file:///Cornfed/Users/slonnegr/Desktop/CurrentFlash/data"
```

Sending Data

Server programs expect data to be sent to them as name-value pairs separated by ampersands.

For a GET request the parameter pairs go at the end of the URL.

For a POST request that parameters go in the body of the request.

Sending the Request

1. Create a LoadVars object.

```
sendVars = new LoadVars();
```

2. Store the pairs as instance variables in the new object.

```
sendVars.username = "Claude";
sendVars.age = 17;
```

3. Send the request.

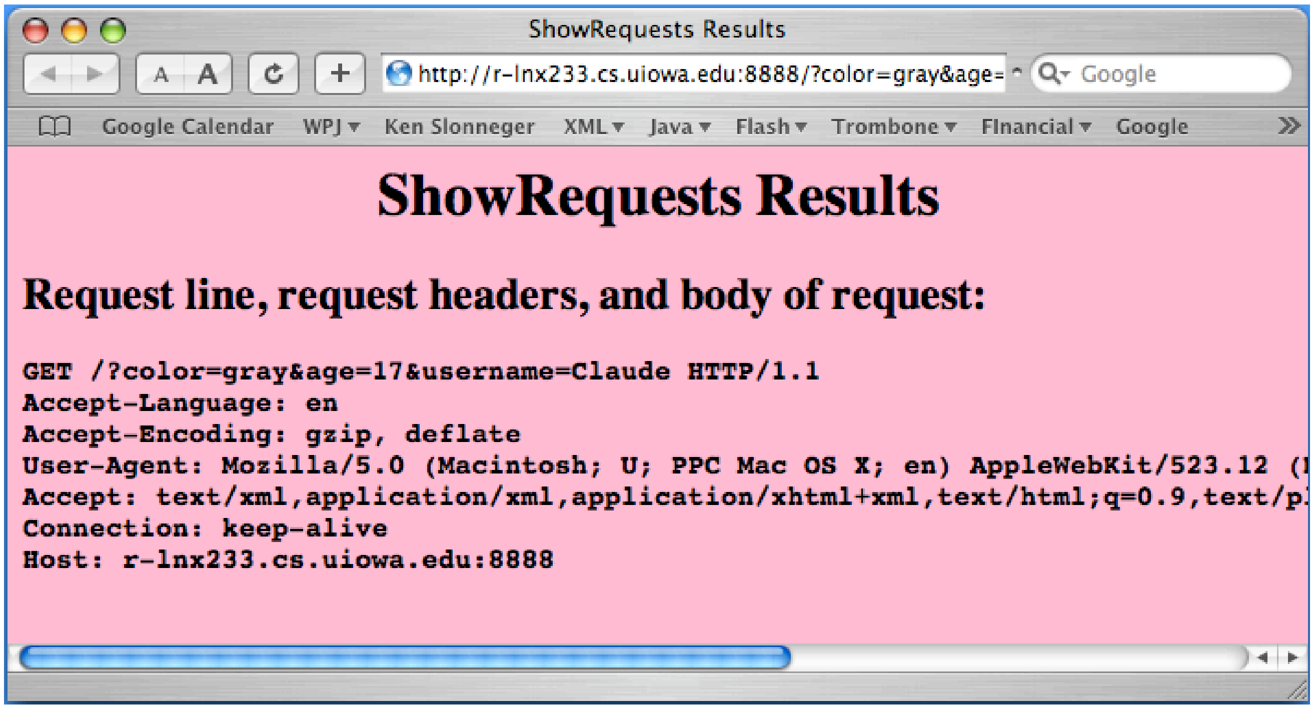
```
sendVars.send(url, "_self", "GET");
```

Example: SendData.fla

Enter the following ActionScript code at KeyFrame 1.

```
    sendVars = new LoadVars();

    sendVars.username = "Claude";
    sendVars.age = 17;
    sendVars.color = "gray";

    sendVars.send("http://r-lnx233.cs.uiowa.edu:8888", "_self", "GET");
        OR
    sendVars.send(
            "http://webdev.divms.uiowa.edu/slonnegr-tomcat4/ShowParams",
            "_self", "GET");
```

When the request is sent to a Java server program running at port 8888 on the computer *r-lnx233.cs.uiowa.edu*, the result is the following web page.

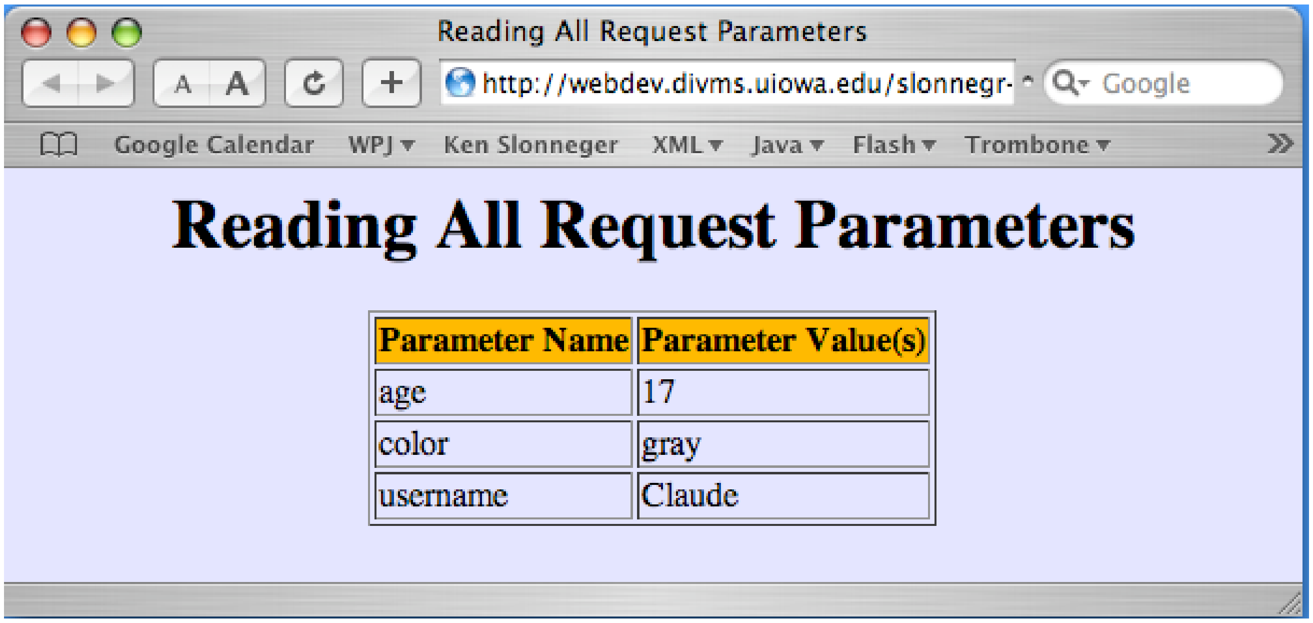


ShowRequests Results

Request line, request headers, and body of request:

```
GET /?color=gray&age=17&username=Claude HTTP/1.1
Accept-Language: en
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/5.0 (Macintosh; U; PPC Mac OS X; en) AppleWebKit/523.12 (
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/p
Connection: keep-alive
Host: r-lnx233.cs.uiowa.edu:8888
```

When the request is sent to a Java Servlet running on the computer *webdev.divms.uiowa.edu*, the result is the following web page.


Reading All Request Parameters
http://webdev.divms.uiowa.edu/slonnegr-
Google
Google Calendar
WPJ
Ken Slonneger
XML
Java
Flash
Trombone

Reading All Request Parameters

Parameter Name	Parameter Value(s)
age	17
color	gray
username	Claude

I could not get a POST request to work correctly.

Shared Objects

Flash has a mechanism that allows an executing Flash movie to store and/or retrieve limited amounts of data on the user's computer similar to the way browser cookies store data for web pages.

Some possible uses of shared objects:

- Store the user name and password on the local computer so it need not be typed by the user each time it is requested.
- Store the high scores from the games played by the user on this machine.
- Store user preferences between sessions.
- Store local copies of data obtained from a web site so that web accesses can be minimized.

The process begins with the creation of a StoredObject object using the class method *getLocal*.

```
    var so : SharedObject = SharedObject.getLocal("foobar");
```
or
```
    var so : SharedObject = SharedObject.getLocal("foobar", "/");
```
In the first version, the information stored can be accessed only by the same executing Flash movie.

In the second version, the information stored can be accessed by all executing Flash movies on a particular computer.

By specifying other path names, accessibility falls between these two extremes.

The information to be stored and retrieved is accessed through an instance variable *data* belonging to the SharedObject object.

Example

Create a Flash document *SharedObjectSave.fla* with the following code at KeyFrame 1.

```
var so:SharedObject = SharedObject.getLocal("foobar","/");
so.data.nums = new Array(121, 143, 165, 187);
so.data.isHungry = false;
so.data.userName = "Claude";
so.data.ob = { x:33, y:55, z:77};
so.flush();
```

The *flush* command forces the values to be saved locally, but closing the *swf* file will have the same effect.

Create another Flash document *SharedObjectRecall.fla* with the following code at KeyFrame 1.

```
var so : SharedObject = SharedObject.getLocal("foobar","/");
trace(so.data.nums);
trace(so.data.isHungry);
trace(so.data.userName);
trace(so.data.ob.x);
trace(so.data.ob.y);
trace(so.data.ob.z);
```

Output

```
121,143,165,187
false
Claude
33
55
77
```

Observe that information is stored in the local file using name-value pairs similar to attributes for the property *data* of the SharedObject object.

All the information stored in a particular SharedObject can be removed using the *clear* instance method.

```
so.clear();
```

Example: SharedObjectLogin.fla

Create two input text fields with the instance names *myLogin* and *myPasswd* and label them appropriately.

Create four button with instance names *saveButton*, *loadButton*, *clearButton*, and *resetButton* and with labels "Save", "Load", "Clear", and "Reset", respectively.

Enter the following ActionScript code at KeyFrame 1.

```
var mySO:SharedObject = SharedObject.getLocal("myCookie");
saveButton.onRelease = function()
{
	mySO.data.myLoginData = myLogin.text;
	mySO.data.myPassword = myPasswd.text;
	mySO.flush();
}
loadButton.onRelease = function()
{
	myLogin.text = mySO.data.myLoginData;
	myPasswd.text = mySO.data.myPassword;
}
clearButton.onRelease = function()
{
	mySO.clear();
}
resetButton.onRelease = function()
{
	myLogin.text = "";
	myPasswd.text = "";
}
```

When this movie is executed, we must click in each of the text fields before typing.

The problem of automatically moving the focus to the first text field and using the tab key to move from field to field seems to have no easy solution.

Here is a solution using the left and right arrow keys to put the focus in one or the other text field. It makes use of the Selection class.

Enter the following ActionScript code above the previous code.

```
var listener : Object = new Object();
listener.onKeyDown = function()
{
    if (Key.isDown(Key.LEFT))
    {
        Selection.setFocus(myLogin);
    }
    if (Key.isDown(Key.RIGHT))
    {
         Selection.setFocus(myPasswd);
    }
}
Key.addListener(listener);
```

Creating Text Fields Dynamically

The MovieClip class has an instance method that can be used to create a text field dynamically.

```
createTextField(instanceName : String,
                depth : Number,
                x : Number, y : Number,        // position of upper-left corner
                width : Number, height : Number) : TextField
```

A TextField object has a number of properties that can be altered to customize the text field. Here are some of them.

Property	Default Value
type	"dynamic"
border	false
background	false
multiline	false
embedFonts	false
selectable	true
wordWrap	true
variable	null
maxChars	null

A text field created using the *createTextField* method receives a default TextFormat object with the following settings (not default values for TextFormat).

Property	Setting	Comments
font	"Times New Roman"	"Times" on Mac OS
size	12	
color	0x000000	
bold	false	
italic	false	
underline	false	
url	""	hyperlink for text in field
target	""	_self, _blank
align	"left"	"center, "right", "justify"
leftMargin	0	
rightMargin	0	
indent	0	
leading	0	
blockIndent	0	
bullet	false	
kerning	false	
display	block	
tabStops	[ ]	empty array

These values my be altered by creating a new TextFormat object, defining values for some of the properties, and then attaching the TextFormat object to the TextField object.

```
createTextField("myTF", 99, 100, 50, 300, 40);
myTF.text = "Nobody goes there anymore. It’s too crowded.";
myFmt = new TextFormat();
myFmt.size = 24;
myFmt.color = 0x00ffff;        // cyan
myFmt.bold = true;
myTF.setTextFormat(myFmt);
```

Example: FlyingText.fla

This Flash document creates and array of words that are placed into individual dynamic text fields, which are created in MovieClip objects named *word0*, *word1*, *word2*, and so on.

The k^th^ MovieClip object has an instance variable *scale* that ranges from -k*200 to 300 by having 10 added as each frame is entered. When the scale value is between 10 and 300 the word shows on the stage scaled by that value so that the word appears to be moving toward the front. When scale reaches 300, the MovieClip object is made invisible.

Enter the following ActionScript code at KeyFrame 1.

```
words = "Multimedia,Web,Programming,with,Flash";
wordList = words.split(",");
createAllWords(wordList);

function createAllWords(wordList)
{
    for (var k=0; k<wordList.length; k++)
    {
        mc = createWord(k, wordList[k]);        // create movie clip with this word
        mc._x = Math.random()*(Stage.width-300)+150;
        mc._y = Math.random()*(Stage.height-100)+50;
        mc._xscale = mc._yscale = 0;
        mc.scale = 0-k*200;        // set scale variable to a nonpositive amount
    }
}

function createWord(n, word)
{
    createEmptyMovieClip("word"+n, n);
    mc = this["word"+n];
    myFormat = new TextFormat();
    myFormat.font = "Arial";
    myFormat.color = 0x000000;
    myFormat.size = 24;
    myFormat.align = "center";
    myFormat.bold = true;
    mc.createTextField("myTextField", 1, -100, -20, 200, 40);
    mc.myTextField.text = word;
    mc.myTextField.embedFonts = true;
    mc.myTextField.setTextFormat(myFormat);
    return mc;
}
```

```
onEnterFrame = function()
{
    moveWords();
}

function moveWords()
{
    for (var k=0; k<wordList.length; k++)
    {
        mc = this["word"+k];
        mc.scale = mc.scale + 10;        // increase scale of this movie clip
        if (mc.scale > 300)
        {                                // hide movie clip when scale is too big
            mc._visible = false;
        }
        else if (mc.scale > 0)
        {                        // set scale of movie clip to scale when positive
            mc._xscale = mc.scale;
            mc._yscale = mc.scale;
        }
    }
}
```

Flash Built-in Components

Flash has three kind of components, pre-built and compiled Movie Clips, that we can use in our documents.

Data Components
Playback Components
User Interface Components

Component Panel accessed using Window ➜ Components (ctrl+F7).

We concentrate on the User Interface Components only.

Accordion	Label	ScrollPane
Alert	List	TextArea
Button	Loader	TextInput
CheckBox	Menu	Tree
ComboBox	MenuBar	UIScrollBar
DataGrid	NumericStepper	Window
DateChooser	ProgressBar	
DateField	RadioButton	

Example: Form.fla

This example uses Flash Components to build a form similar to what can be done with HTML form tags.

TextInput

Drag a TextInput Component onto the stage, select it, and give it the instance name *username*.

Open the Properties Panel (ctrl+F3) and select its Parameters tab.

Option	**Default**
editable	true
password	false
text	*none*

Place a Static text field "Enter Name:" next to the TextInput Component as a label.

Repeat this process for a password field, except call the instance *passwd* and set the *password* option to *true*.

Retrieving TextInput data:

```
trace(username.text);
trace(passwd.text);
```

CheckBox

Drag three CheckBox Components onto the stage, giving them instance names *check1*, *check2*, and *check3*.

Open the Parameters tab of the Properties Panel.

Option	**Default**
label	CheckBox
labelPlacement	right
selected	false

Enter labels Apple, Banana, and Peach for the three CheckBox Components.

Retrieving the checked values:

```
fruit = new Array();
for (k=1; k<=3; k++)
{
    ch = eval("check"+k);
    if (ch.selected)
    {
        fruit.push(ch.label);
    }
}
trace(fruit);
```

RadioButton

Drag three RadioButton Components onto the stage, giving them instance names *red*, *green*, and *blue*.

Open the Parameters tab of the Properties Panel.

Option	**Default**
data	*none*
grouptName	radioGroup
label	RadioButton
labelPlacement	right
selected	false

Change the *groupName* for all three buttons to *color*.

Change the *label* values to Red, Green, and Blue.

Change the *data* values to 0xff0000, 0x00ff00, and 0x0000ff.

Retrieving the chosen RadioButton's data:

```
trace(color.selectedData);
if (red.selected)
{
    trace(red.data);
}
```

List

Drag a List Component onto the stage, giving it the instance name *city*.

Open the Parameters tab of the Properties Panel.

Option	**Default**
data	[]
labels	[]
multipleSelection	false
rowHeight	20

Change *multipleSelection* to *true*.

Double click on the *labels* value ([]) and enter these value (using the + button).

Berlin
Chicago
Dublin
London
New York
Paris
Rome
Vienna   Click OK.

Double click on the *data* value ([]) and enter these value (using the + button).

```
THF
ORD
DUB
LHR
NYC
PAR
ROM
VIE
```

Click OK.

Retrieving the selected information:

```
cities = city.selectedItems;          // an array
for (k=0; k<cities.length; k++)
{
      trace(cities[k].label+":"+cities[k].data);
}
```

TextArea

Drag a TextArea Component onto the stage, giving it the instance name *message*.

Change its width to 200 and its height to 80.

Open the Parameters tab of the Properties Panel.

Option	**Default**
editable	true
html	false
text	*none*
wordWrap	true

Change the value of *text* to "Message contents.".

Place a Static text field above the TextArea with the label "Please leave your message here.".

Retrieving TextArea data:

```
trace(message.text);
```

Button

Drag two Button Components onto the stage, giving them the instance names *submit* and *reset*.

Open the Parameters tab of the Properties Panel.

Option	**Default**
icon	*none*
label	Button
labelPlacement	right
selected	false
toggle	false

Change the *label* value for the two buttons to Submit Query and Reset.

Responding to the Buttons:

```
submit.onRelease = function()
{
    trace(username.text);
    trace(passwd.text);
    trace(color.selectedData);
    if (red.selected)
    {    trace(red.data); }
    fruit = new Array();
    for (k=1; k<=3; k++)
    {
        ch = eval("check"+k);
        if (ch.selected)
        {    fruit.push(ch.label);  }
    }
    trace(fruit);
    cities = city.selectedItems;
    for (k=0; k<cities.length; k++)
    {    trace(cities[k].label+":"+cities[k].data); }
    trace(area.text);
}

reset.onRelease = function()
{
    resetForm();
}

function resetForm()
{
    username.text = "Claude";
    passwd.text = "catcatcat";
    blue.selected = true;
    check1.selected = false;
    check2.selected = false;
    check3.selected = true;
    area.text = "        Message contents.";
```

Call the *resetForm* method when the Flash document is first loaded to initialize the Components on the stage.

Adding Components

Several web sites have Flash Components (and other things) that can be downloaded, sometimes at no cost.

Example

Open the web site called Flash Exchange: Select Help ➜ Flash Exchange.

Choose Flash under Exchange By Product.

Choose Freeware as the License type and click Filter.

Scroll down to an item called Dice and click Download.

Sign in to your Adobe account (or create one).

You will find a file *DiceComp4.mxp* that has been downloaded.

Install the new Component.

 Select Help ➜ Manage Extensions....

 Click the Install icon in the Extension Manager window.

 Navigate to the file *DiceConmp4.mxp* and click Install or Open.

Example: Components.fla

This Flash document illustrates using several more of the Flash Components and using the new Dice Component that we just installed.

ComboBox

Drag a ComboBox Component onto the stage, giving it the instance name *myCombo*.

Open the Parameters tab of the Properties Panel.

Option	**Default**
data	[]
editable	false
labels	[]
rowCount	5

The ComboBox may be populated the same way as the List Component.

As an alternative, both ComboBox and List can be populated dynamically.

```
    myCombo.removeAll();

    myCombo.addItem("Macromedia", "http://www.macromedia.com");
    myCombo.addItem("Adobe", "http://www.adobe.com");
    myCombo.addItem("Apple", "http://www.apple.com");

    myCombo.addEventListener("change", doChange);

    function doChange(evt)
    {       getURL(evt.target.selectedItem.data);
    }
```

DateField

Drag a DateField Component onto the stage, giving it the instance name *date*.

Open the Parameters tab of the Properties Panel.

Option	Default
dayNames	[S,M,T,W,T,F,S]
disabledDays	[]
firstDayOfWeek	0
monthNames	[January,February,...,December]
showToday	true

NumberStepper

Drag a NumberStepper Component onto the stage, giving it the instance name *num*.

Open the Parameters tab of the Properties Panel.

Option	Default
maximum	10
minimum	0
stepSize	1
value	0

Change *maximum* to 100, *minimum* to 2, *stepSize* to 2, and *value* to 50.

Dice

Drag a Dice Component onto the stage, giving it the instance name *die*.

Open the Parameters tab of the Properties Panel.

Option	Default
colArr	[]
cornerColor	0xA9A9A9
dotColor	0x333333
DiceNum	3
IsJumping	true

Drag a Button Component onto the stage, giving it the instance name *submit* and the label Submit. Enter the following ActionScript code to respond to the button.

```
    submit.onRelease = function()
    {
        trace("ComboBox:" + myCombo.value);
        trace("Date: " + date.selectedDate);
        trace("Number: " + num.value);
        trace("Die: " + die.DiceNum);
    }
```

Sample Output

```
ComboBox:http://www.apple.com
Date: Fri Dec 21 00:00:00 GMT-0600 2007
Number: 96
Die: 3
```

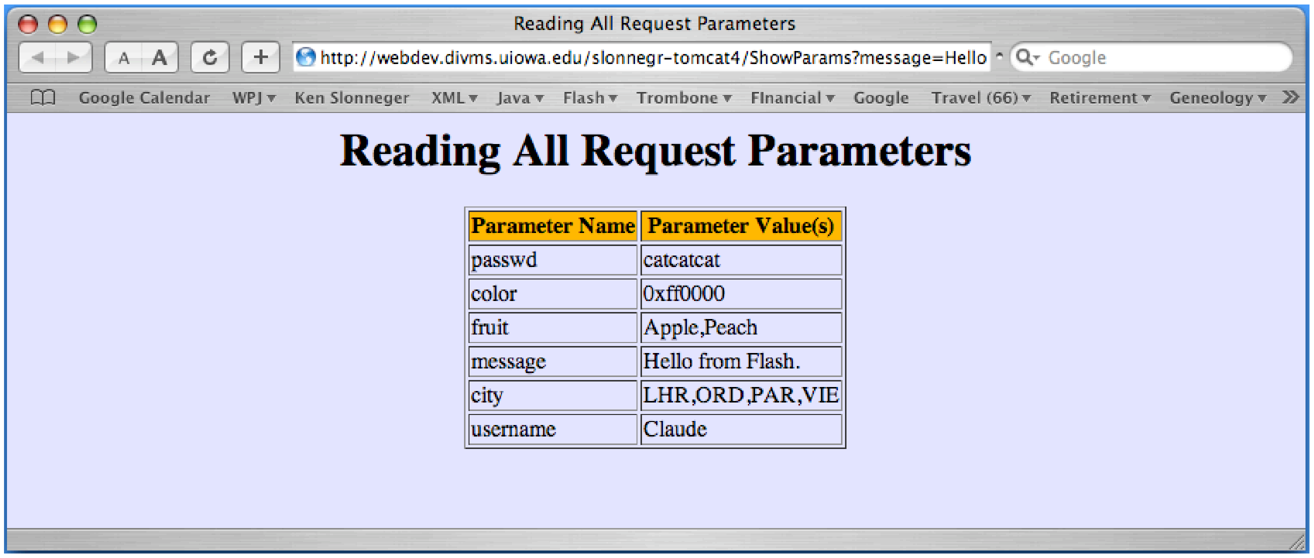
Example: SendForm.fla

This Flash document produces the same form used in Form.fla, but when the submit button is pressed, the data is collected into a LoadVars object and sent to a Java Servlet that displays all the parameters and values in a web page.

Enter the following ActionScript code at KeyFrame 1.

```
submit.onRelease = function()
{
    send = new LoadVars();

    send.username = username.text;
    send.passwd = passwd.text;
    send.color = color.selectedData;

    fruit = new Array();
    for (k=1; k<=3; k++)
    {
        if (eval("check"+k).selected)
        {
            fruit.push(eval("check"+k).label);
        }
    }
    send.fruit = fruit;

    cities = city.selectedItems;
    cityCodes = new Array();
    for (k=0; k<cities.length; k++)
    {
        cityCodes.push(cities[k].data);
    }
    send.city = cityCodes;

    send.message = area.text;

    send.send("http://webdev.divms.uiowa.edu/slonnegr-tomcat4/ShowParams",
                "_self", "GET");
}
```

The snapshot below shows the web created by submitting the form.


Reading All Request Parameters
http://webdev.divms.uiowa.edu/slonnegr-tomcat4/ShowParams?message=Hello
Google
Google Calendar WPJ XML Java Flash Trombone Financial Google Travel (66) Retirement Geneology

Reading All Request Parameters

Parameter Name	Parameter Value(s)
passwd	catcatcat
color	0xff0000
fruit	Apple,Peach
message	Hello from Flash.
city	LHR,ORD,PAR,VIE
username	Claude