ActionScript 3.0 for Designers

Wednesday, September 19
10:45 a.m. – 12:00 a.m.
Room 1

This is a skeletal outline, with select example code segments, to accompany the presentation notes for *ActionScript 3.0 for Designers*. To check for more up-to-date materials, visit: http://www.fmaonline.com/flashforward/

I hope to give you a fast-paced, information-packed look at Action Script 3.0 from a new user's perspective. If you are already familiar with AS3, you will likely be bored to slumber in this presentation. However, if you are just getting started—particularly within the Flash CS3 environment, in which this presentation is grounded—you may get more out of it.

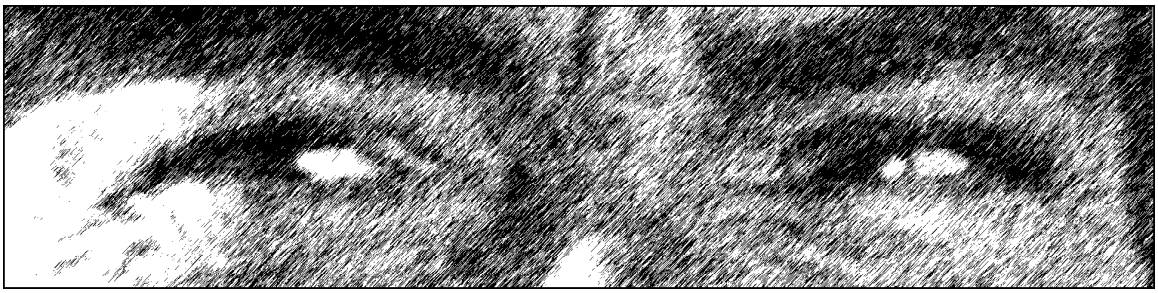
Please feel free to shout out questions as we go, but please also be prepared for me to defer them until the end of the presentation, if time is tight. I'll try to monitor our progress and adjust our pace if needed. To get the most out of our meeting, try to sit back and soak it in from a big picture perspective. I'll make these notes, and accompanying source files, available at the url above, so you don't have to worry about recording everything we discuss.

Finally, all of the material in this presentation is distilled from ***Learning ActionScript 3.0***, published by O'Reilly, and co-written with Zevan Rosser. It should be available, in one form or another, by the time of this session.

I. INTRODUCTION

The Good

- You're among friends.
- You can still code in the Timeline.
- AS3 is:
  - More Consistent
  - More Powerful
  - A Lot Faster
  - Closer to Other Languages
  - Stricter (Better Error Reporting)



The Bad

- AS3 is:
  - Stricter (Initially Frustrating)
  - More Verbose
- Some things have really changed or are just... gone
  - A *few* examples:
    - Initially a pain: `ReleaseOutside/MouseUpOutside, ....`
    - No problem: `eval(), _level, ....`
    - Adjustments needed, if you relied on: `_global, ....`
    - A bit more work: `Color` class, `Key` object, ...
- You're facing a bigger learning curve
- We only have 75 minutes...

The Ugly

- AS1/AS2 and AS3 are Not Compatible
  - Two separate virtual machines in Player
  - Communication only through indirect means (e.g. LocalConnections)
- You May Need to Straddle the AS2/AS3 Fence
  - To support legacy projects or ongoing existing development
  - If you can't afford to focus your learning on one version
- Flash Platform Diverging and Converging
  - Initial disparity between Flash and Flex still a problem, but improving
    - Example Issues:
      - Different component sets
      - Different authoring techniques (embedding and more)
    - Improvements:
      - Flex Component Kit for Flash CS3
  - Disparity to a lesser degree with AIR development, also improving
    - Can now author AIR apps in Flash CS3
- *We only have 75 minutes!!*

II. CODE PLACEMENT

NOTE: Some content herein will be explained in a little later on, as time allows.

Timeline

No More Stage Coding

Cannot apply scripts directly to MC or Button -- *on(), onClipEvent()*

(This is a *good* thing, so nut up.)

Frame Scripts

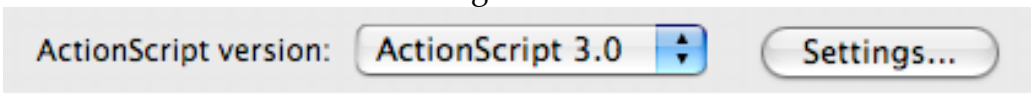
Business as usual (perhaps a few more imports?)

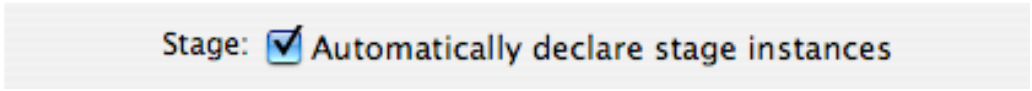
Stage Instances Auto-Declared

Handy for simple timeline scripts

Can get in the way when declaring/typing instance variables

Can be disabled in Publish Settings:





Can now adjust frame rate at runtime

```
this.stage.frameRate = 30;
```

Example (movie clip instantiated as *mc* on stage):

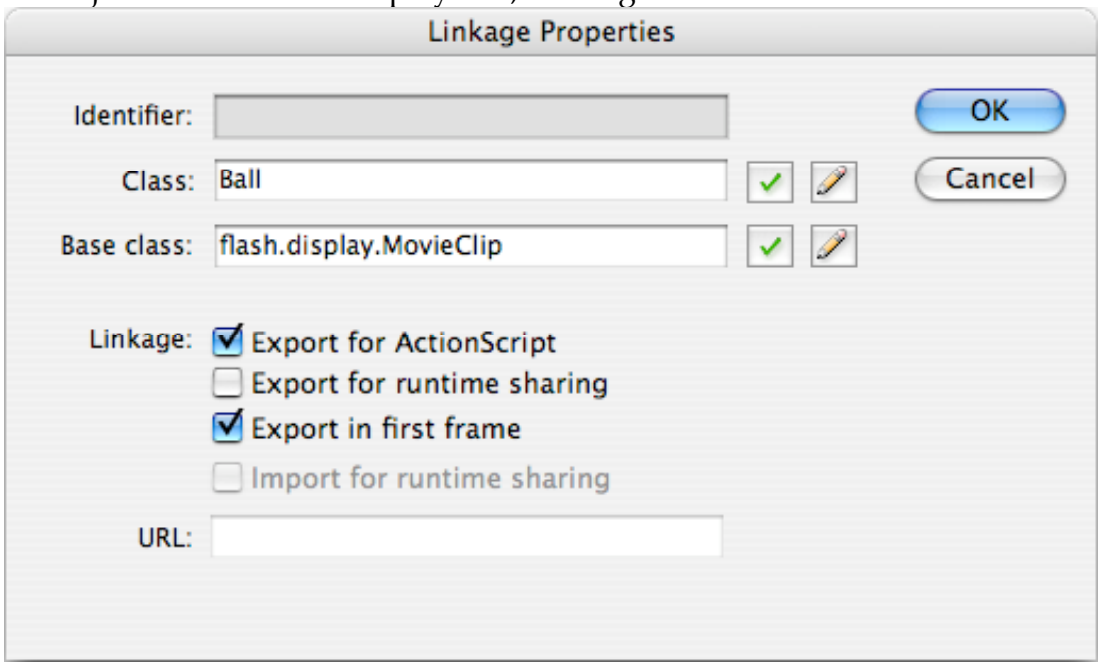
```
mc.scaleX = 2;
//note revised property name and 0-1 percentage scale
```

Symbol Instances

Simplified Linkage Process

Everything accomplished through Class and Base class

In conjunction with the Display List, this is great.



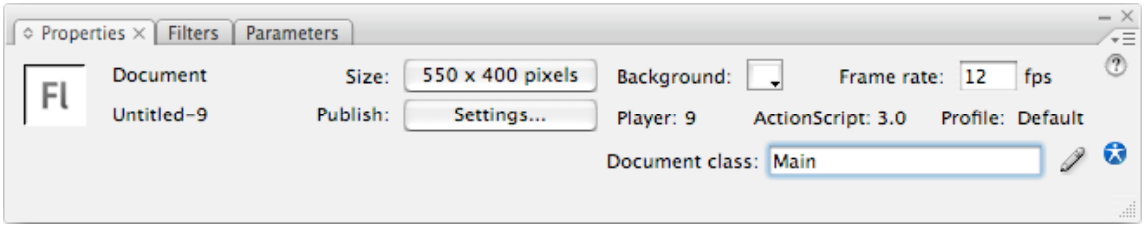
Ball class (in external class file called *Ball.as* in same directory):

```
package {

    import flash.display.Sprite;

    public class Ball extends Sprite {

        public function Ball() {
               this.scaleX = 2;
               //note use of this instead of instance name
        }
    }
}
```

Document Class

Easier Entrée into OOP

Timeline Class



Example (in external class file called *Main.as* in same directory):

```
package {

    import flash.display.MovieClip;

    public class Main extends MovieClip {

        public function Main() {
            trace("FlashForward Boston 2007");
        }
    }
}
```

Instantiation (and Static, etc.)

```
//in timeline, document class, or other class
var particle:Particle = new Particle();

//in external class
package {

    import flash.display.Sprite;

    public class Particle extends Sprite {

        public function Particle() {
            //particle physics here
        }
    }
}
```

III. DATA TYPING

Tell Flash compiler and Player what type of data will be in use at a given moment.
Mandatory
Better Error Reporting
    Compile-Time
    Runtime
Example:

```
var userName:String = "Rich";
trace(userName*3);
//error: 1067: Implicit coercion of a value of type String
  to an unrelated type Number.
```

IV. DISPLAY LIST

Easily create display objects. Everything is now accomplished with `new()`

```
var mc:MovieClip = new MovieClip();
```

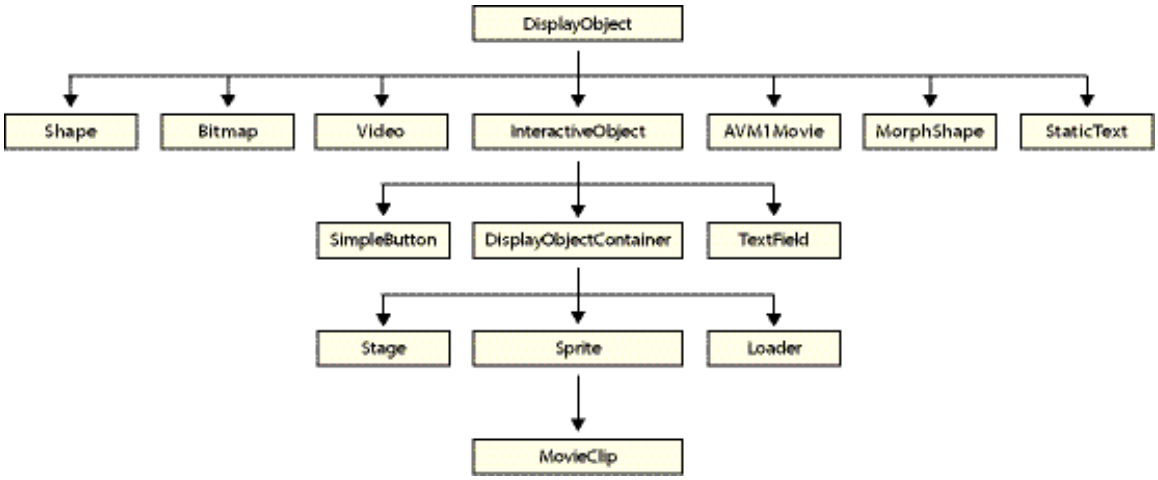
Easily add and remove display objects, including depth management

```
addChild(mc) //automatically add to top of visual stack
removeChildAt(0) //remove from specific depth (e.g. lowest)
```

Easily re-parent children CODE

```
mc1.removeChild(mc3)
mc2.addChild(mc3)
```

Take advantage of new display object types



A *couple* of examples:
    Dynamically create shapes
    A sprite is a one-frame movie clip
Object vs. Container
    Display object containers can have children
    All display object containers are display objects, but not vice versa
Cast display objects when necessary

```
MovieClip(obj).gotoAndStop(1);
```

V. Events

Listeners

No more onEventHandlers.

All events are now handled by the EventDispatcher, using event listeners

Their use is the same as in AS2, but they are not confined to custom objects (for use with components, EventDispatcher, etc.)

Flow

Events cascade through the display list, divided into two phases:

Capture and Target—which consists of event propagation from the senior-most object (the stage, for example) down to the target (such as a button clicked by the user).

Bubbling—when the event bubbles back to the top of the display list.

You can listen for events during either or both of these phases, and even cancel events during propagation.

Don't worry too much about the complexities of the multiple event phases until you get more comfortable with AS3. However, you can take full advantage of event propagation. Instead of assigning a listener to many buttons, you might assign a listener to a parent and allow it to work its way down to the buttons. You can then parse data from the event to determine its point of origin—the target of the original event.

Example:

The following will bring any movie clip in the same scope as the listener to the top of the visible stack.

```
this.addEventListener(MouseEvent.MOUSE_OVER, onBringToTop, false, 0, true);
function onBringToTop(evt:MouseEvent):void {
    var mc:MovieClip = evt.target;
    setChildIndex(mc, numChildren - 1);
}
```

Constants

Events now reside in their own classes, and are expressed as constants.

For example, mouse events are found in the MouseEvent class, and they include `MouseEvent.MOUSE_UP`, `MouseEvent.CLICK`, and so on.

VI. Loading

URLRequest

All URLs are now handled the same way

Data from URLs can now be accessed immediately, during streaming, including text, raw binary data, or URL-encoded variables

Data is then passed to loader objects, such as URLLoader, for loading external assets, and various other load commands. For example, the Sound class loads sound files with the .load() method, to which the URLRequest data is passed.

Example:

```
var snd:Sound = new Sound();
var snd_url:URLRequest = new URLRequest("song.mp3");
snd.load(snd_url);
```

VII. Sound

Now with greater granularity and control
With greater granularity comes greater verbosity
Briefly, sound works in this way:
Sounds are loaded and played via the *Sound* class
Optionally, loading can be buffered via the *SoundLoaderContext* class
Each sound is played in its own channel via the *SoundChannel* class
All sounds go through a single global mixer via the *SoundMixer* class
Raw sound data, in the form of frequency spectrum analysis and amplitude, can be accessed in real time using the computeSpectrum() method.
Said raw data is stored in a highly efficient array called the ByteArray, storing 256 amplitudes for each of the left and right channels, every time the data is sampled. For example, 20 frames per second means 20 * 512, or 10,240, samples per second.
Reading from the array is very easy, however, as a single method both reads the sequential data and advances the array index. Therefore, it's very easy to use an amplitude value to control anything you can manipulate with numerical data (such as x- and/or y- coordinates, scale, rotation, alpha, color, frame number, etc.

Project in Timeline

```
import flash.display.Sprite;
import flash.net.URLRequest;
import flash.media.*;
import flash.events.*;
import flash.utils.ByteArray;

var snd:Sound = new Sound();
var vis:Sprite;

function initSound() {
    snd.addEventListener(IOErrorEvent.IO_ERROR, onIOError, false, 0, true);
    snd.addEventListener(Event.COMPLETE, onLoadComplete, false, 0, true);

    snd.load(new URLRequest("song.mp3"));
}

function onLoadComplete(evt:Event):void {
    removeEventListener(IOErrorEvent.IO_ERROR, onIOError);

    vis = new Sprite();
    vis.x = vis.y = 20;
    addChild(vis);
    vis.stage.frameRate = 30;
    initVisualization();

    var channel:SoundChannel = new SoundChannel();
    channel = snd.play();
}

function onIOError(evt:IOErrorEvent):void {
    trace("An error occurred when loading the sound:", evt.text);
}
```

```
function initVisualization() {
    var ball:Sprite;
    for (var i:int=0; i<256; i++) {
        ball = new BallRed();
        ball.x = i*2;
        vis.addChild(ball);
    }
    for (i=0; i<256; i++) {
        ball = new BallYellow();
        ball.x = i*2;
        vis.addChild(ball);
    }

    addEventListener(Event.ENTER_FRAME, onVisualize, false, 0, true);
}

function onVisualize(evt:Event):void {
    var bytes:ByteArray = new ByteArray();
    SoundMixer.computeSpectrum(bytes, false, 0);

    for (var i:Number = 0; i < 256; i++) {
        vis.getChildAt(i).y = 100 - (bytes.readFloat() * 200);
    }
    for (i = 256; i < 512; i++) {
        vis.getChildAt(i).y = 200 - (bytes.readFloat() * 200);
    }
}

initSound();
```

VIII. Object-Oriented Programming (OOP) (10: 75)

This 5-minute *demo* can't even qualify as a crash course.

This is meant *only* as a demo of the logical extension in project development promoted by ActionScript 3.0.

For a good crash course, sit in on *OOP for the Noob – What's in the Box?* by the accomplished **Peter Elst**, immediately following lunch, in this same room.

Object-oriented programming improves development by breaking down a project into objects. Briefly, you can look at the evolution of programming in this way:

Sequential programming—a linear sequence of commands, as can sometimes be seen in basic individual scripts

Procedural programming—adds the use of procedures, also known as subroutines or functions, to partially compartmentalize code blocks for more efficient use

Object-oriented programming—encapsulates code into logical classes that are as reusable and multi-purposeable as possible, from which objects can be created. Classes can inherit from other classes to minimize redundancies and streamline development. As such, a Vehicle class can give rise to Car and Truck classes that share many attributes, but also contain elements private to each respective class.

OOP is not mandatory in AS3, but it can really help in the cases of large projects, when multiple programmers are involved, or when a project is well-suited to an object-oriented approach (such as certain kinds of games, for example).

Don't feel pressured into adopting OOP practices before you're ready, or make the mistake of inseparably linking OOP with AS3. AS3 is versatile enough to use for the kind of

As a very brief demonstration of AS3 OOP—again, merely as a demo of a possible progression of development you may follow in your learning of the language—the previous timeline example is provided in OOP form.

This demo takes advantage of the simplest possible OOP approaches made possible by Flash CS3 (outlined in a moment).

Project in OOP

Document Class **Main**, from *Main.as*

```
package {

    import flash.display.Sprite;
    import flash.net.URLRequest;
    import flash.media.*;
    import flash.events.*;
    import Visualization;

    public class Main extends Sprite {

        private var _snd:Sound = new Sound();

        public function Main() {
            _snd.addEventListener(IOErrorEvent.IO_ERROR, onIOError, false, 0, true);
            _snd.addEventListener(Event.COMPLETE, onLoadComplete, false, 0, true);

            _snd.load(new URLRequest("song.mp3"));
        }

        private function onLoadComplete(evt:Event):void {
            removeEventListener(IOErrorEvent.IO_ERROR, onIOError);

            var vis:Visualization = new Visualization();
            vis.x = vis.y = 20;
            addChild(vis);
            vis.stage.frameRate = 30;

            var channel:SoundChannel = new SoundChannel();
            channel = _snd.play();
        }

        private function onIOError(evt:IOErrorEvent):void {
            trace("An error occurred when loading the sound:", evt.text);
        }
    }
}
```

Class **Visualize**, from *Visualization.as*

```
package {

    import flash.display.Sprite;
    import flash.media.SoundMixer;
    import flash.utils.ByteArray;
    import flash.events.Event;

    public class Visualization extends Sprite {

        public function Visualization() {
            var ball:Sprite;
            for (var i:int=0; i<256; i++) {
                ball = new BallRed();
                ball.x = i*2;
                addChild(ball);
            }
            for (i=0; i<256; i++) {
                ball = new BallYellow();
                ball.x = i*2;
                addChild(ball);
            }

            addEventListener(Event.ENTER_FRAME, onVisualize, false, 0, true);
        }

        private function onVisualize(evt:Event):void {
            var bytes:ByteArray = new ByteArray();
            SoundMixer.computeSpectrum(bytes, false, 0);

            for (var i:Number = 0; i < 256; i++) {
                getChildAt(i).y = 100 - (bytes.readFloat() * 200);
            }
            for (i = 256; i < 512; i++) {
                getChildAt(i).y = 200 - (bytes.readFloat() * 200);
            }
        }
    }
}
```

IX. Conclusion

If you enjoyed listening to this presentation just half as much as I enjoyed giving it, well then, I enjoyed it twice as much as... oh, I can't even finish that. I hope you got something out of this. I realize that you have choices and that other damn-fine presentations were running alongside mine. If you see me in the halls, please let me know what you thought of it. If you like it, tell everyone. If you didn't, lie.

—Rich Shupe, FMA

X. Additional Resources

Additional info re: this presentation: http://www.fmaonline.com/flashforward/
Flash CS3 Training I contributed to the great Lynda.com library: http://www.lynda.com/
Adobe's ActionScript 3.0 Language Reference
http://livedocs.adobe.com/flash/9.0/ActionScriptLangRefV3/index.html
Peter Elst's OOP Primer: http://www.adobe.com/devnet/actionscript/articles/oop_as3.html