

# Object-Oriented Design: Extending Flash with ActionScript 2.0 Classes

**Dr. Scott D. Lipscomb**

*Northwestern University, School of Music*  
Music Education & Music Technology

<http://faculty-web.at.northwestern.edu/music/lipscomb/>

# Macromedia Flash allows the developer to ...

- create simple to complex interactive multimedia, greatly facilitated by an intuitive graphical user interface (GUI) & built-in “behaviors”
  - cross-platform, cross-browser compatibility
- wrap a wide variety of media types into a *web-ready* presentation format (SWF)
  - images, sounds, animations, etc.
- extend the program’s capabilities significantly by using ActionScript



# Creating Animation

A simple example

[01soundCrash\\_template.fla](#)

# Going to the “next level”

Danger, Danger ...  
Warning, Will Robinson!!

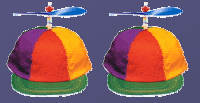
# Flash Comes of Age

- MX 2004 introduces ActionScript 2.0
  - Object-oriented programming (OOP) techniques become much more elegantly integrated
    - closer to industry standard programming languages like Java, C++, or C#
- It is now possible to create your own classes in Flash or to “easily” alter the behavior of classes distributed with the program

# ActionScript 1.0 → 2.0

(Moock, 2004)

- relatively little new runtime functionality
- radically improves OOP development
  - formalizes OOP syntax & method in the Flash environment
    - adds the “class” keyword for creating classes
    - adds an “extends” keyword to establish inheritance
    - “getter” and “setter” syntax for setting object properties
    - “public” and “private” keywords determine access to object properties
- *requires* that classes be defined in external “.as” files
  - these files can be created and edited in an external text editor or using the “Professional” version of Flash



# OOPs ... I did it again!\*

## What is object-oriented programming?

\*Perhaps the only Britney Spears quote you'll hear during the conference.



# OOP is ...

- a different approach to programming
- designed to make complex applications more manageable
  - breaking programs down into self-contained, interacting, and reusable modules
- adds a level of conceptual organization to a program
  - groups related functions & variables together into separate classes



# OOP Concepts

- *class* – a self-contained software module that contains related functions and variables
  - *methods* – a class's capabilities (functions)
  - *properties* – a class's data values (variables)
    - *public* – accessible from anywhere in the program
    - *private* – accessible only from within the class (or its subclasses)
- *object* – an instantiation of a class
  - *instance* – a specific incarnation of an object

# OOP Concepts

(continued)

- *encapsulation* – maintaining private properties and internal code isolated from the rest of the program
  - as long as method names, parameters, & return values remain the same, changes can be made to the class without necessitating changes to the program(s)
- *data types* – used to impose limits on what can be stored in a variable, used as a parameter, or passed as a return value
  - results in significantly improved debugging & error tracking within the Flash environment

# OOP Concepts

(continued)

- *inheritance* – allow one class to adopt the method & property definitions of another
  - can use Macromedia's classes as a source
- *packages* – directories that contain groups of classes
  - an easy way to organize your class files on your hard drive
- *compilation* – occurs when an OOP application is exported as a Flash movie (SWF)
  - potential errors are identified during this process

# Building an OOP from Scratch

- create one or more classes
- instantiate objects from those classes
- using ActionScript, tell the objects what to do

# Using Macromedia's Classes: Sound object

## ● properties

- duration
  - in milliseconds
- position
  - in milliseconds

## ● methods

- attachSound()
- getBytesLoaded()
- start() or start([offset])
  - offset in *seconds*
- stop()
- setVolume()
- setPan()

# Using Macromedia's Classes: MovieClip object

## ● properties

- `_alpha`
- `_currentframe`
- `enabled`
- `_height` & `_width`
- `_name`
- `_rotation`
- `_visible`
- `_x` & `_y`
- `_xscale` & `_yscale`

## ● methods

- `attachMovie()`
- `getBytesLoaded()`
- `getBytesTotal()`
- `getURL()`
- `gotoAndPlay()`
- `gotoAndStop()`
- `play()`
- `nextFrame()` & `prevFrame()`
- `stop()`





# Instantiating a Class

- Basic (AS 1.0)

```
mySound = new Sound();
```

- New & Improved (AS 2.0)

```
var mySound:Sound = new Sound();
```

• This is the ActionScript equivalent to dragging a symbol (class) onto the stage (creating an object instance).

It is possible to create multiple objects from a single class.

# Dot Syntax

Setting properties &  
calling methods

# Lingo Scripting (verbose)

```
on exitFrame
  if levelslider = TRUE then
    if the locV of sprite 116 > 362 then set the locV of sprite 116 to 362
    put the locV of sprite 116 into level
    set the soundlevel to (362-level) / 13
    set the locH of sprite 116 to 559
  end if
  go to the frame
end exitframe
```

from [Sinewave.dir](#)

# ActionScript – Dot Syntax

```
on clipEvent(exitFrame){  
    if levelslider == TRUE {  
        if sprite(116)._y > 362{  
            sprite(116)._y = 362;  
        }  
        level = sprite(116)._y;  
        set the soundlevel to (362-level) / 13  
        sprite(116)._x = 559;  
    }  
    gotoAndPlay(_currentframe);  
}
```

# Setting Properties & Calling Methods in ActionScript

```
// instantiate the Sound object  
var mySound:Sound = new Sound();  
  
// attach a sound - "linkage" necessary  
mySound.attachSound("cool_song");  
  
// set sound level between 0-100  
mySound.setVolume = 85;  
  
// initiate playback  
mySound.start();
```

# Macromedia's Inconsistency

yup ... they too are imperfect!



# The Problem

- Sound object (ms vs. sec)
  - [atmi2005\\_macromedia.fl](#)
- problem fixed (hardwired)
  - [atmi2005\\_macromedia\\_fixed.fl](#)
- creating a class
  - [sdlSound.as](#) ([expanded version](#))
  - [atmi2005\\_bare.fl](#)
- adding a package
  - [atmi2005\\_bare\\_package.fl](#)

# The Future

## ● Playback Control

- [atmi2005\\_playbackControl.fla](#)

## ● BubbleMachine (Lipscomb & Jacoby, 2004)

- [BubbleMachine.swf](#)
- version 2.0 in progress

# Contact Info

**Dr. Scott D. Lipscomb**

*Northwestern University*

Music Education & Music Technology

[lipscomb@northwestern.edu](mailto:lipscomb@northwestern.edu)

<http://faculty-web.at.northwestern.edu/music/lipscomb/>