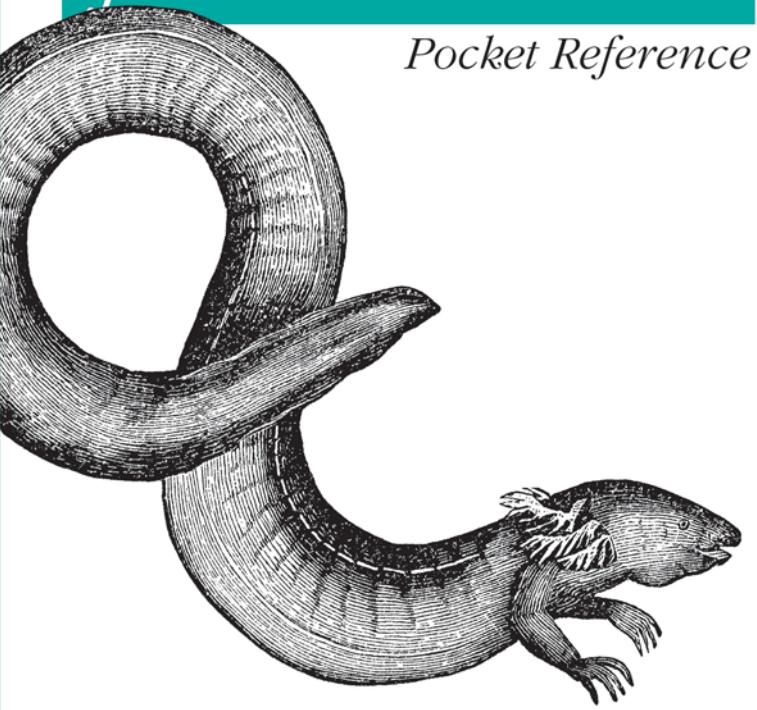
Quick Reference
for Flash MX Programmers

ActionScript *for* Flash MX

Pocket Reference



O'REILLY®

Colin Moock

Web Programming

O'REILLY®

ActionScript for Flash MX Pocket Reference

Macromedia Flash MX is the de facto standard for delivering web-based multimedia and rich Internet applications to over 500 million users worldwide. The *ActionScript for Flash MX Pocket Reference* provides a complete summary of ActionScript, Flash MX's object-oriented programming language, covering the methods and properties of core objects and classes. Also covered are ActionScript's global properties, global functions, operators, statements, keywords, and directives. This pocket reference includes an excellent summary of ActionScript syntax and best practices, covering datatypes, variables, loops, conditionals, identifiers, event handling, and object-oriented programming in short order. For programmers coming to ActionScript from other languages, it includes a quick orientation to the most common Flash elements and operations, including movie clips, loading and drawing graphics, text manipulation, data transfers, and XML parsing.

This pocket-sized book is easy to take anywhere and serves as the perfect companion to the best-selling *ActionScript for Flash MX: The Definitive Guide*. Completely up to date, including methods and properties added in the latest Flash Player, it is the indispensable quick reference for ActionScript programmers.

Visit O'Reilly on the Web at *www.oreilly.com*

ISBN 0-596-00514-8

US $9.95
CAN $14.95

90000

9 780596 005146

6 36920 00514 8

ActionScript for Flash MX
Pocket Reference

Colin Moock

O'REILLY®
Beijing · Cambridge · Farnham · Köln · Paris · Sebastopol · Taipei · Tokyo

ActionScript for Flash MX Pocket Reference
by Colin Moock

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly Media, Inc. books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*safari.oreilly.com*). For more information, contact our corporate/institutional sales department: (800) 998-9938 or *corporate@oreilly.com*.

Editor:	Bruce Epstein
Production Editor:	Emily Quill
Cover Designer:	Emma Colby
Interior Designer:	David Futato

Printing History:
March 2003: First Edition.

0-596-00514-8
[C] [10/04]

Contents

ActionScript for Flash MX Pocket Reference

Introduction

ActionScript is Macromedia Flash's scripting language, used to create everything from graphic user interfaces and games to sound sequencers and animated screensavers. Syntactically, ActionScript is nearly identical to JavaScript (both are based on the ECMA-262 specification), but it is tailored to Flash content rather than HTML content. ActionScript also resembles Java and C++, using many of the same statements, operators, and punctuation found in those languages. ActionScript supports both procedural and object-oriented programming or any mix of the two.

This book provides "just the facts" coverage of ActionScript for Flash MX (the sixth version of the software). Differences from ActionScript for Flash 5 (the first official version of ActionScript) are covered at *http://www.moock.org/webdesign/lectures/newInMX/*. For exhaustive coverage of ActionScript, consult O'Reilly's *ActionScript for Flash MX: The Definitive Guide*.

Authoring ActionScript Code

All code in a Flash document (a *.fla* file) must be attached to either a keyframe on the timeline, or a button or movie clip on the Stage.

To attach code to a keyframe:

1. Select the keyframe in the timeline (by clicking it)
2. Open the Actions panel (Window → Actions or F9)
3. Add the desired code to the right side of the panel (called the Script pane)

NOTE

The Actions panel has two different modes of operation, Normal Mode (menu-driven code creation) and Expert Mode (manual typing). To change the mode, use the pop-up Options menu in the upper-right corner of the panel, as shown in Figure 1.

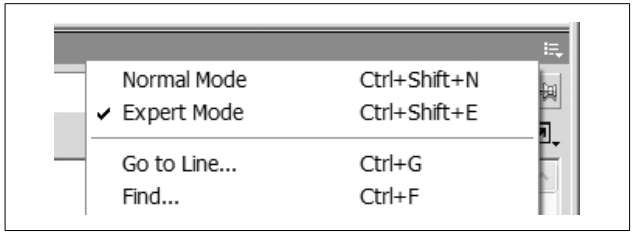


Figure 1. Setting the Actions panel to Expert Mode

During playback, Flash executes the code on a keyframe before displaying the contents of that frame (allowing the code to manipulate elements on the Stage before they are displayed). To add a new keyframe to a timeline, select a frame in the timeline and choose Insert → Keyframe.

To attach code to a button or movie clip:

1. Select the button or movie clip on the Stage
2. Open the Actions panel
3. Add the desired code to the right side of the panel

Code attached to buttons or movie clips must be contained within *event handlers*, which determine when the code

should execute at runtime. For example, to invoke the function *submitForm()* when a button is clicked, attach the following code to the button:

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

Similarly, to reposition a movie clip five pixels to the left as the playhead advances through the timeline (i.e., once for every tick of the frame rate), attach the following code to the clip:

```
onClipEvent (enterFrame) {
  this._x -= 5;
}
```

For information on button and movie clip events, see "Event Handling."

Outputting Debug Messages

The Flash authoring tool provides a "Test Movie" mode (Control → Test Movie), which is used to compile a Flash document (a *.fla* file) into a Flash movie (a *.swf* file) for testing in a debugging environment. To display text in the Test Movie mode's Output window, use the *trace()* function:

```
trace("Testing...testing...");
```

Text sent to the Output window appears in Test Movie mode only; to create a *.swf* file suitable for distribution (e.g., for display in a web browser), check the Omit Trace Actions option under File → Publish Settings → Flash, and click the Publish button in the Publish Settings dialog box. For details on embedding a *.swf* file in an HTML page, see "Working with Web Browsers." Use a dynamic text field to display text in the standalone or browser versions of the Flash Player.

Code Placement Best Practices

As a general rule, most code should reside on the main timeline of a project's main *.fla* file, on a frame labeled "main".

The "main" frame is not usually the first frame; instead, it typically follows any *preloader* code, which waits for the movie to load. Classes, long blocks of code, and large functions should be stored in external *.as* files (which are plain text files saved with the *.as* extension). Incorporate *.as* files into the document via the `#include` directive.

For example, a typical Flash application setup would be:

```
// PRELOADER ON FRAME 2 OF MAIN TIMELINE
if (this._framesloaded == this._totalframes) {
  this.gotoAndStop("main");
}

// CODE ON FRAME 3 OF MAIN TIMELINE
this.gotoAndPlay(2);

// CODE ON FRAME LABELED "main" OF THE MAIN TIMELINE
#include "SomeClass.as"
#include "SomeOtherClass.as"

init();

function init() {
  // ...start up application
}
```

For information on preloading code and content, see *http://design.oreilly.com/news/action_0501.html*. For more best practices, see *http://www.macromedia.com/desdev/mx/flash/whitepapers/actionscript_standards.pdf*.

Finding Code

If you're faced with a movie that seems to be missing important code, open the Actions panel and try these techniques for finding it:

- To search the current scene for ActionScript code, use the Movie Explorer search option (Window → Movie Explorer).
- With the Actions panel open, click on any frame in the timeline that contains a little "a" icon, which indicates

the presence of ActionScript code (the "a" may look like a tiny circle).

- Look for, and select, any white circle with a black outline on stage. Such circles indicate empty movie clips, which often contain code. If there's no code on the empty clip itself, double-click the clip to edit it, and investigate its frames.
- Select each button in the movie, one at a time. Some programmers place long, important scripts directly on buttons, instead of centralizing their code.
- Check the timeline for hidden or masked layers. A layer with a red X icon next to it is hidden during authoring, but may contain clips and buttons with code. Similarly, masked layers may also contain obscured objects that bear code. Unlock masked layers to reveal their contents.
- Unlock all layers. Empty movie clips (the little circles with black outlines) are hidden when the layer they're on is locked.
- In the Actions panel, use Ctrl+F (Windows) or Command+F (Macintosh) to open the Find dialog box. This is useful for searching large scripts for the specified text. (Use the Movie Explorer to search across all scripts.)

Using Movie Clips

Every Flash document contains a Stage, on which we place shapes, text, and other visual elements, and a main timeline, through which we define changes to the Stage's contents over time. The main timeline (i.e., the *main movie*) can contain independent submovies called *movie clips* (or *clips* for short). Each movie clip has its own independent timeline and *canvas* (the Stage is the canvas of the main movie) and can even contain other movie clips. A clip contained within another clip is called a *nested clip*. A clip that contains another clip is the nested clip's *parent clip*. The *playhead* represents the current slice in time (i.e., the active frame in a

timeline). The duration of a frame is called a *tick*. For example, if the frame rate is 30 frames per second, a tick is 1/30 of a second.

A single Flash document can contain a hierarchy of interrelated movie clips. For example, the main movie may contain a mountainous landscape. A separate movie clip containing an animated character can be moved across the landscape to give the illusion that the character is walking. Each movie clip maintains a numbered *content stack* that governs how individual elements are layered visually at runtime (see "Movie Clip Depths").

ActionScript offers detailed control over movie clips. Each movie clip in a movie is an instance of the *MovieClip* class, so we can programmatically play a clip, stop it, move its playhead within its timeline, set its properties (such as its size, rotation, transparency level, and position on the Stage), and manipulate it as a true programming object. For a complete list of *MovieClip* properties and methods, consult the reference section in the latter half of this book.

Movie clips can be thought of as the raw material used to produce programmatically generated content in Flash. For example, a movie clip can serve as the ball or paddle in a pong game, as a drop-down list in an order form, or as a container for background sounds in an animation. Even Flash's built-in GUI components are created with movie clips.

Just as all object instances are based on a class, all movie clip instances are based on a template movie clip, called a *symbol*. Movie clip symbols live in the Library of each Flash document (*.fla* file).

To make a new, blank symbol, follow these steps:

1. Select Insert → New Symbol. The Create New Symbol dialog box appears.
2. In the Name field, type an identifier (i.e., a name) for the symbol.

3. For Behavior, select the Movie Clip radio button.
4. Click OK.

To make a new instance of a movie clip symbol at authoring time, click the symbol in the Library and drag it to the Stage. An instance created in this way should be named manually via the Property inspector. A movie clip's *instance name* is the identifier used to refer to it from ActionScript. Clips without instance names are given an automatic name that is not easily accessible via ActionScript. A clip's instance name should not be confused with the name of the symbol from which the clip was created, nor confused with the symbol's *linkage identifier*.

To set a symbol's linkage identifier (a.k.a. exporting the symbol), follow these steps:

1. In the Library, select the desired symbol.
2. In the Library's pop-up Options menu, select Linkage. The Linkage Properties dialog box appears.
3. Select the Export For ActionScript checkbox.
4. In the Identifier field, supply a unique name for the clip symbol. The name can be any string—often simply the same name as the symbol itself—but should be different from all other exported clip symbols.

To make a new instance of a movie clip symbol at runtime, use *MovieClip.attachMovie()*. For example, the following code creates a new instance of the symbol whose linkage identifier is `boxSymbol` and names the instance `box_mc`, placing it on depth 0 of the main timeline (see the next section, "Movie Clip Depths," for details on depths).

```
_root.attachMovie("boxSymbol", "box_mc", 0);
```

Be sure to set the symbol's linkage identifier (here, `boxSymbol`) before creating instances of it with *attachMovie()*.

To make a new generic movie clip instance (i.e., one with no symbol) at runtime, use *MovieClip.createEmptyMovieClip()*, which has the following syntax:

```
theClip.createEmptyMovieClip(newName, depth);
```