- CREATE DYNAMIC, FUNCTIONAL WEBSITES AND GAMES.
- INTEGRATE FLASH WITH DATABASES, REMOTE SERVERS, AND PEERS.
- LEARN ACTIONSCRIPT, XML, PHP, SQL, ASP, RSS, HTTP, SOCKETS, PROXIES.

Flash and XML

A Developer's Guide

Dov Jacobson and Jesse Jacobson

Flash and XML
A Developer's Guide

Dov Jacobson
Jesse Jacobson

Addison-Wesley

Boston • San Francisco • New York • Toronto • Montreal
London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

Pearson Education Corporate Sales Division
201 W. 103rd Street
Indianapolis, IN 46290
(800) 428-5331
corpsales@pearsoned.com

Visit AW on the Web: www.aw.com/cseng/

Library of Congress Cataloging-in-Publication Data

Jacobson, Dov.
Flash and XML : a developer's guide / Dov Jacobson and Jesse Jacobson.
p. cm.
Includes index.
ISBN 0-201-72920-2 (alk. paper)
1. XML (Document markup language) 2. Web sites—Design. 3. Flash (Computer file)
I. Jacobson, Jesse. II. Title.
QA 76.76.H94 J334 2002
005.7'2—dc21 2001045738

ISBN: 0-201-72920-2
Text printed on recycled paper
1 2 3 4 5 6 7 8 9 10—DOC—0504030201
First printing, October 2001

Preface

Flash can make fabulous introductions to fancy web sites. High-grade wow factor fills the opening screens. Colors are delicious, typefaces are trendy, and movement is exquisite. Rave music is looping and attitude is everywhere. Each intro is a unique work of art. But they all have one thing in common.

Each has a button that says SKIP INTRO.

Rarely is gratuitous glitz so clearly labeled. When budgets tighten, webmasters will not skip shopping cart, or skip catalog. But they may skip intro.

What's Up?

The web is outgrowing its eye-candy phase, and so must Flash. A web site must be pretty to be viable, but it must be highly functional as well.

Flash pros can't get by on gee-whiz animation or cool interfaces. They'd better prepare to do some of the heavy lifting on the working web. Flash screens must interface with dynamic content, with back-end databases, with server-based applications, and even with other live users.

XML provides a path.

Don't Stop!

Once you connect your Flash code to the outside world, your scope is unbounded. And so is the list of things to learn.

In this book we learn a lot. We begin with Flash and XML. We study networking protocols and PHP server scripting. We learn sockets and SQL and a few fancy XML dialects. We go on to achieve competence in many other related technologies and put them together to build working web systems.

Who Are You?

If you are creative and technical, this book is for you. Maybe you have a design background. You learn whatever technology you need to realize your vision. This book offers you skills that open fresh new worlds. Let's hope your imagination can keep up.

Maybe you have computer science training. You're happy with simple gray buttons. Your art is an elegant code design. You are ready to put Flash to work with all the web technologies you already know.

This book was written by authors who approach Flash from both angles. We want it to speak to both engineers and artists, and we struggled (often with each other) to support both perspectives.

Why?

We wrote this book because it wasn't there when we needed it.

Typographic Conventions

We have adhered to several conventions in this book.

Case Conventions

lowerAndUpperCase is used to name variables and functions in ActionScript
UpperAndLowerCase labels frames and names symbols, objects and constructors.
ALL_UPPER_CASE is used in manifest constants, SQL keywords, PHP globals.
Separated_by_underscores is typically used for PHP names.

Typesetting Conventions

italics indicate names we gave variables, functions, elements, instances, and so on.
`codefont` shows language keywords and expressions.
`codefont` is also used to set off the text output of any program.
boldface introduces the first use of an important term.
SMALL CAPS are used for pull-down menu options. Slashes show hierarchy.
"quotes" are reserved for string literals.

These rules are breached occasionally in the book. Sometimes this is for historic reasons, sometimes due to ambiguity and sometimes because it just didn't look right.

Dov and Jesse Jacobson
Berkeley Lake, Georgia
dov@bigfun.net
jesse.jacobson@dartmouth.edu
FlashandXML.com

Acknowledgments

For her sure guidance, for her sturdy belief in this book and her faith in its authors: Mary T. O'Brien, Executive Editor at Addison-Wesley.

For their able support: her editorial assistants, Alicia Carey and Mariann Kourafas.

For his imaginative, clearheaded thinking on many aspects of this book, and especially for its handsome looks: Kevin Smith of stateless designs.

For her care in the tricky transformation of bytes into ink and paper: Jacquelyn Doucette.

For their patience and professionalism in the composition process: Stratford Publishing Services.

For his help in unraveling the toughest technical knots and his aggravating habit of improving our work: Nigel Head. He made his points with some very clever code sequences and, well, we included some verbatim.

For their careful, knowledgeable, and lively technical review: Christopher Ian Smith, John Paul Rawlins, Todd Williamsen, Scott Hamlin, Mike Callery, Jennifer Hall, and the accomplished Brendan Hall.

For their rapid responsiveness and for an enjoyable week of debugging Flash patches together: Zlavik Lozben and Eric J. Wittman of Macromedia.

For his clean direction of Flash technology: Peter Santangeli. (We can all thank him.)

For building Macromedia with his bare hands and his big mouth: our old friend Marc Cantor.

For their enthusiasm and their insights: Colin Moock and other FlashForward speakers, Urami and other macromedia.flash posters, and the lively community at FlashKit.com.

For their support—financial and creative—of the real-life version of the game described in this book: webmaster Garth Moore and producer Zoë Burkholder of the ASPCA (animaland.org).

For permission to reprint his haiku: Walter Vereertbrugghen.

And for their critical contribution to this project: the makers of fine caffeinated beverages all over the world.

Table of Contents

1 Flash *Basics*

CHAPTER ONE

In this chapter, we learn to build simple interface elements and interactive structures. Both the elementary structures we build and the techniques we learn are used and developed further throughout the book.

This chapter is intended for readers who are new to Flash. It brings them up to speed quickly on editing conventions and ActionScript fundamentals. It also presents an overview of the design of data and execution in the trivia game.

The Background of Flash

Flash is an interesting phenomenon. It began with technology intended for pen computing. This technology—which translated strokes into drawings—was commercialized as SmartSketch, a personal computer drawing package. SmartSketch made it easy for artists to create pictures that were defined as geometry, not as a screen of pixels. The inventors of the program, such as, Jonathan Gay, understood the advantages of this technology—vector drawing—over the more familiar technique of pixel painting. Geometric descriptions scale perfectly and they are extremely efficient. It is easy to move these pictures from one computer to another. High-level drawing descriptions also make it easy to organize the elements of a drawing—and to manipulate them mathematically.

Easy manipulation means powerful animation. SmartSketch evolved into CelAnimator—a professional cartooning tool. The cartooning world is fairly small—and was smaller still in the 1990s. This vector animation program might have disappeared like its predecessors except for the emergence of the Internet. Device independence and tiny bandwidth requirements were a great fit for the web—which was starved for animation.

Macromedia was introducing its venerable animation system, Director, to the Internet under the name Shockwave. But pixel-based technology such as Director is inflexible and slow to load. In 1996, Macromedia acquired the product, then called FutureSplash. They changed the name to Flash and merged it into the Shockwave line.

In succeeding years, more interactivity and much more scripting were added to Flash.

After a few awkward attempts to develop a proprietary coding syntax, Flash ActionScript was stabilized by adopting ECMA-262 as the basis of its language definition. ECMA-262 is the international standard for web scripting. (JavaScript is an ECMA-compliant language.)

This decision lowered the learning curve for developers and reduced a lot of confusion and ambiguity. It allowed Flash to develop from a simple animation system to a fairly robust environment for the development of interactive web applications.

A second wise decision also involved the adoption of web standards. Macromedia chose XML to be Flash's data-communication technology. Suddenly Flash developers were able to easily interact with an enormous number of web services. This changed the nature of Flash once more. Instead of a stand-alone web application, a Flash presentation can be integrated into a complex system of web-enabled software. Flash is no longer limited to entertainment. It is now ready to provide high-quality interfaces to all kinds of hard working web applications.

Practical Flash

In this chapter, we do not attempt to conscientiously teach Flash graphics creation. That is the subject of many other books. The chapter is not even an overview of the subject. In fact this chapter is more of an underview. We offer a mouse's-eye view of the construction of a few simple but sophisticated interface elements. We don't suggest that our methods are the best approach to each problem. Multifaceted software like Flash invites creative approaches. The approach we show here, based on much experience in team production of code and graphics, is one good approach. Consider it as you develop your own.

Project Design

The trivia quiz game that we will build throughout this book is complex and sophisticated. It starts, however, with some simple ideas that are recognizable in every later step of development.

Before we start any code, even demonstration code, we need to think a little about the design. In general, a detailed planning phase is always a good investment. But in this case, just a simple layout of hierarchy will suffice.

We organize a *Game* into a series of *Quizzes* (Figure 1.1). Each *Quiz* has several rounds. A *Round* is the presentation of a *Question,* the player's response, and the scoring. The *Question* is presented in Q&A format: a single *Q* and multiple-choice *A*'s.

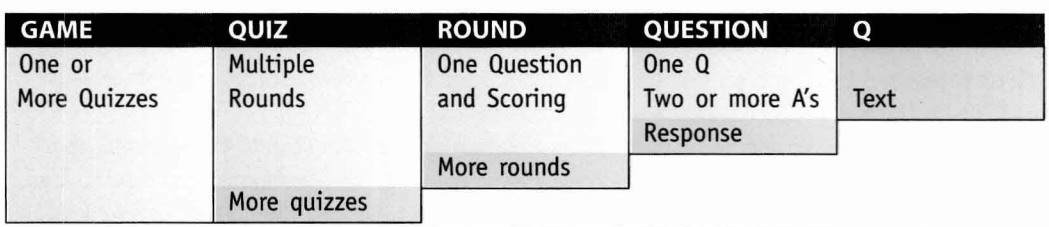


Figure 1.1 Object Hierarchy of the Trivia Game

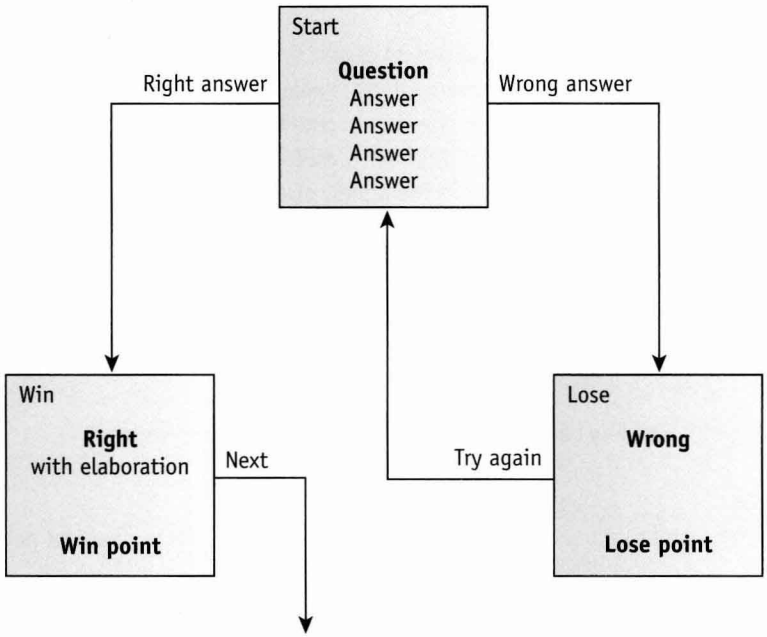


Figure 1.2 Logic at One Round

In this chapter we create a single functional *Round*. We present a *Question* and a series of *Answer* options and determine whether the player has chosen the correct one. That's all for now.

Our approach is to develop three states: *Start* state, *Win* state, and *Lose* state (Figure 1.2). The latter two states comprise the *Response* to the user. The *Start* state encapsulates the *Question* presentation.

Event-Driven Graphics

A great strength of ActionScript is that execution is event driven. Things that happen in the outside world cause things to happen in Flash.

Frame Events

The most common event is that time passes. Of course, time passes rather continuously, but in Flash (as in video or film) time is broken into a stream of regular intervals called frames. In a typical Flash application there are 12 frames per second. Twelve times every second, Flash is interrupted by a **frame** event (more precisely, a **frame start** event), which starts a cascade of calculations. Anything that is in the process of moving must determine its new position and prepare to redisplay. This is how animation happens.

Each movie has a timeline. When a frame event occurs, it advances a pointer along the timeline to execute any new script or effect any changes scheduled for this frame.

Mouse Events

Frame events give Flash animation. Mouse events give it interactivity. When a user clicks a button, that is an event. It is in fact a series of events: First the cursor enters the button's hot spot (the **Roll Over** event). Then the user depresses the button (the **Press** event). Then the user stops pressing (**Release**) and moves the cursor away (**Roll Out**). All four events happen often within a half second, but each has a different meaning in the rapidly stabilizing language of user interface:

Roll Over:	What is this?
Press:	I think I want to perform this action
Release:	Yes, I do! I am committed.
Roll Out:	Oh. Well, now I am bored.

A good designer considers each event and supplies appropriate responses.

Button State

A button has three preprogrammed responses to mouse events. Each is a special named frame.

The **Roll Out** event causes `gotoAndStop("Up")`.
Roll Over causes `gotoAndStop("Over")`.
The **Press** event causes `gotoAndStop("Down")`.

Button Event Handlers

There are many more mouse events besides these three. Flash allows us to attach activity to any combination of events.

An **event handler** connects an event to actions. Event handlers are expressed as code fragments, not as keyframes. They are explicitly packaged as the code called by (for example) the **Drag Out** event or the **<Escape> Key Press** event—or both.

There is a fundamental difference between button states and button events. Button state frames are formulated as part of the button object. This is, in effect, a prototype that we construct that lives in the library. Later, we place as many **instances** of this button in the application as we need.

An instance is not a copy. It is a new reference to the original unchanged prototype. This very powerful concept is at the heart of Flash architecture.

Since the button states are in the prototype, all instances we create (by dragging the button into the scene) share the same states. They behave identically, as long as their behavior is designed in the states.

But every button cannot do the same thing. Some buttons open and some buttons close. Some win and some lose.

To make this happen, Flash allows us to attach event handlers as **Object Actions** to the instance of the button. They can behave like the button states, and they can do more. They can

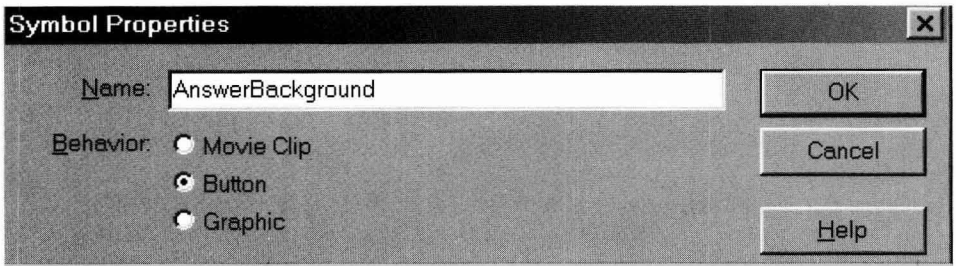


Figure 1.3 Starting a New Button Symbol

identify seven mouse events, not just three, and they can attach any ActionScript to those events on that button. Button state frames (in the button's prototype) usually define the button's appearance and its dynamics—how it looks, how it moves, and how it sounds. Event handlers (attached to the instance) usually specify what the button does, how it affects the rest of the world.

Building a Button

An answer option must be sensitive to the mouse. Players click on one of the options and actions are performed. It would also be good to have visible and audible reaction to the various steps in the mouse click and to the mouse rollover as well. Flash gives us all these abilities and more in the Button primitive. To clearly understand how buttons operate, it is best to take a step or two back.

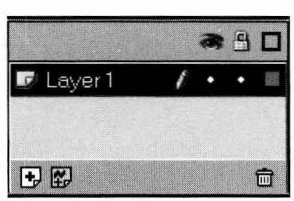
To create a button, invoke Flash's NEW SYMBOL function, tucked away in the INSERT pulldown (Figure 1.3).

Button Graphics

The button we want is a wide panel of color. One such button will sit behind the text of each answer option and perform all mouse interactions.

We create a filled rectangle—nice and wide and not much taller than a single line of text. It can easily be resized and reshaped later, if necessary. Our example is 400 pixels wide and 30 high.

We also included a simple beveled effect by making an outline rectangle of the same size. For clarity, it is given its own *Border* layer. New layers are created by clicking the dog-eared icon with a + sign.



Right-clicking on the outline brings us to the Stroke palette, where we set the thickness to 3. We duplicate the *Border* layer and create *Border-top* and *Border-bottom* using cut and paste and rename. We differentiate them by deletion so that *Border-top* has only the top and left line segments and *Border-bottom*