



A NEW CLASS OF APPLICATIONS: DESKTOP RIAs

FORMERLY
MX DEVELOPER'S JOURNAL

Web Developer's & Designer's Journal

WWW.WEBDDJ.COM

VOLUME 4 ISSUE 10 2006

BUILDING ENGAGING APPLICATIONS AND CONTENT WITH ADOBE TECHNOLOGIES

Video Conference with Flex & FMS

Creating a basic video
conference application

Plus:

Exclusive Preview of Apollo

by Kevin Lynch

Chief Software Architect, Adobe

Presorted
Standard
US Postage
PAID
St. Croix Press



©2006 Adobe Systems Incorporated. All rights reserved. Adobe, the Adobe logo and Better by Adobe are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries.



<cf_essentials>



Indispensable CFMX tools.



FusionReactor

CFMX / J2EE Server, database
and application monitoring

from **\$249**

FusionDebug 

Interactive Debugging
for ColdFusion MX

from **\$99***



www.fusion-reactor.com

* Using GOT2DEBUG discount coupon. Offer ends October 31st.

Trademarks and Registered Trademarks are the property of their respective owners.

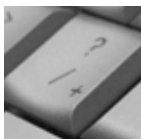
Web Developer's & Designer's Journal

October 2006

\$18 Billion Dollar Adobe Moves Center Stage

by Jeremy Geelan

12



Beyond the Browser

The next generation of Rich Internet Applications

by Kevin Lynch

14



Web and Mobile Convergence

Developing a blog aggregator

by Marco Casario

24



Video Conference with Flex & FMS

Learn how to use Flex 2 and FMS 2

by Renaun Erickson

32



Adding Flash Video to PowerPoint Presentations

It's worth the effort

by Kim Cavanaugh

36



Beyond 'Request-Response' Modes of Web Services

Using Flash and Flex

by Mansour Raad & Andy Gup

38

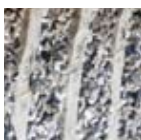


Back and Forth

Custom history management with Flex 2

by Rob Rusher

42

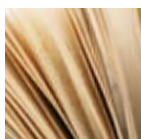


Flex On Ruby on Rails

Automating the communication between the client and server

by Harris Reynolds

46



Working with Large Applications

Static versus dynamic linking: development perspective

by Yakov Fain, Victor Rasputnis, & Anatole Tartakovsky



ColdFusion Hosting is our Complete Focus

➤ **POWERFUL HOSTING PLANS**

FREE SQL server access | FREE account setup | Unlimited email accounts | Generous disk space & data transfer
30 day money-back guarantee | Great value

➤ **RELIABLE NETWORK**

99.99% average uptime | State-of-the-art data center with complete redundancy in power, HVAC, fire suppression,
bandwidth and security | 24/7 network monitoring

➤ **FANTASTIC SUPPORT SERVICES**

24/7 support services | Knowledgeable phone support | We focus on your individual needs

CFDynamics

866.233.9626 ➤ CFDYNAMICS.COM

For years we have been involved in the Cold Fusion community and have come to know what developers and project managers look for in a web host. The combination of our powerful hosting plans, reliable network, and fantastic support sets us apart from other hosts.

Real service. Real satisfaction. Real value. Real support. Real Freedom.





**For the greatest hits
of the 70's, 80's and 90's
call your web host's
tech support.**

For answers call us at 1-866-EDGEWEB
3 3 4 3 9 3 2

When calling your web host for support you want answers, not an annoying song stuck in your head from spending all day on hold. At Edgewebhosting.net, we'll answer your call in two rings or less. There's no annoying on-hold music, no recorded messages or confusing menu merry-go-rounds. And when you call, one of our qualified experts will have the answers you're looking for. Not that you'll need to call us often since our self-healing servers virtually eliminate the potential for problems and automatically resolve most CF, ASP, .NET, SQL, IIS and Linux problems in 60 seconds or less with no human interaction.

Sleep soundly, take a vacation, and be confident knowing your server will be housed in one of the most redundant self-owned datacenters in the world alongside some of the largest sites on the Internet today and kept online and operational by one of the most advanced teams of skilled Gurus, DBAs, Network and Systems Engineers.

By the Numbers:

- 2 Rings or less, live support
- 100% Guarantee
- 99.999% Uptime
- 2.6 GBPS Redundant Internet Fiber Connectivity
- 1st Tier Carrier Neutral Facility
- 24 x 7 Emergency support
- 24 Hour Backup
- Type IV Redundant Datacenter



**For a new kind of easy listening,
talk to EdgeWebHosting.net**

<http://edgewebhosting.net>



2003 - 2006

- Shared Hosting
- ColdFusion
- Managed Dedicated Servers
- BlueDragon
- Managed Colocation
- ASP
- .NET
- .Linux
- .Java
- SQL Server
- .MySQL
- Self-Healing Servers
- Semi-Private Servers

**Web Developer's &
Designer's Journal**

Group Publisher Jeremy Geelan
Art Director Louis F. Cuffari

Editorial Board

Aral Balkan
Erik Bianchi
Craig Goodman
Jim Phelan
Andrew Phelps
Darron J. Schall
Stephanie Sullivan
Jeff Tapper
Jesse Randall Warden

Editorial

Editor
Nancy Valentine, 201 802-3044
nancy@sys-con.com

Associate Editor

Lauren Genovesi, 201 802-3041
laureng@sys-con.com

To submit a proposal for an article, go to
<http://gndis.sys-con.com/proposal>.

Subscriptions

E-mail: subscribe@sys-con.com
U.S. Toll Free: 888 303-5282
International: 201 802-3012
Fax: 201 782-9600
Cover Price U.S. \$5.99
U.S. \$29.99 (12 issues/1 year)
Canada/Mexico: \$49.99/year
International: \$59.99/year
Credit Card, U.S. Banks or Money Orders
Back Issues: \$12/each

Editorial and Advertising Offices

Postmaster: Send all address changes to:
SYS-CON Media
577 Chestnut Ridge Rd.
Woodcliff Lake, NJ 07677

Worldwide Newsstand Distribution

Curtis Circulation Company, New Milford, NJ

List Rental Information

Kevin Collopy: 845 731-2684,
kevin.collopy@editthroman.com,
Frank Cipolla: 845 731-3832,
frank.cipolla@epostdirect.com

Promotional Reprints

Megan Mussa, 201 802-3024
megan@sys-con.com

Copyright © 2006

by SYS-CON Publications, Inc. All rights reserved. No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopy or any information storage and retrieval system, without written permission.

Web Developer's & Designer's Journal (ISSN#1546-2242)

is published monthly (12 times a year) by

SYS-CON Publications, Inc., 577 Chestnut Ridge Road,
Woodcliff Lake, NJ 07677.

SYS-CON Media and SYS-CON Publications, Inc., reserve the right to revise, republish, and authorize its readers to use the articles submitted for publication. Adobe and Adobe products are either registered trademark or trademarks of Adobe Systems Incorporated in the United States and/or other countries. SYS-CON Publications, Inc., is independent of Adobe. All brand and product names used on these pages are trade names, service marks or trademarks of their respective companies.

\$18 Billion Dollar Adobe Moves Center Stage

by Jeremy Geelan

Once upon a time, Sun and Apple used to have about the same market cap. Today Apple has a market cap of \$63BN while Sun's is just \$17BN. Ahead of Sun, unbeknownst to many, is Adobe. Its market cap today is \$18BN. For a company that in 1998 was worth "only" \$1.7BN it not been a bad 8 years!

My point is: Everything Changes. And in the world of Web development and design, and of the Internet technologies undergirding it all, it changes faster than most everywhere else.

Just 18 months ago no one had heard of "AJAX" or "Ajax," let alone of the OpenAjax Alliance, ajaxCFC, or LAJAX. No one had heard of Atlas and no one had heard of Apollo. Yet right now as you read this you are fully aware of all these A-words. Everything changes.

As you read this issue at MAX 2006, enjoying the late October sun in Las Vegas, stop and think a moment of how Web savvy Adobe has become in that same short 18-month period, and how mobile savvy it is about to prove itself in the next 18 months. Think about how much Google's role in the i-Technology Landscape has changed in the same period. Think about your own career goals, your own skill set. Has it changed as fast?

It is my firm belief that for developers and designers alike, from now on, keeping up with Adobe is going to be tough. If you are not yet familiar with Flash Lite, you ought to start tomorrow. If you didn't yet deploy Flash video on your clients' websites, ditto. If you are not yet experimenting with the Flash-PDF combo, why not?

With a greater reach than even Microsoft has, in terms of the number of devices -- more than 115 million Flash technology-enabled devices have now shipped worldwide -- Adobe is very much at what I call the "i-Technology sweet spot." In this issue the company's Chief Software Architect, Kevin

Lynch, unveils how the Apollo runtime will give developers a new paradigm that dramatically changes how applications can be created, deployed, and experienced.


Whether you are using Dreamweaver to build HTML or Ajax applications for the browser, you will be able to take that Web application to the desktop with Apollo. While everyone else is wrestling with how to get Rich Internet Applications working within the browser, Adobe is already taking RIAs outside the browser. The possibilities, once that happens, are virtually limitless.

As Chief Operating Officer Shantanu Narayan said in a phone interview last month, "We're seeing an explosion of digital content and we're poised to capitalize on that." The next version of the PDF reader, Adobe Acrobat, is due out in November. The third edition of Creative Suite, which packages Photoshop, Illustrator and other programs (and will be the first to run native on the new Intel-based Macs), is expected to ship in the first half of 2007.

The i-Technology stage, in short, is set. And Adobe is fully kitted out to stride out right into the very center of it. And stay there.

Don't take my word for it. Take instead the word of the high-caliber speakers at MAX, or of the top-notch writer-developers in this issue, including Kim Cavanaugh, Mansour Raad, Rob Rusher, Marco Casario, Rebaun Erickson, and Harris Reynolds.

Or take the word of Kevin Lynch, leader of Adobe's Apollo mission to create a new cross-OS, cross-device application runtime to extend the reach of Rich Internet Applications to the desktop. Because of Adobe, users will soon get a more compelling experience offline as well as online, with a rich media application that supports video, audio, HTML, Flash, and PDF. What's not to like?

Everything changes. With Adobe, it seems always to be for the best. Seize the day! 

Jeremy Geelan is Sr. Vice-President, Editorial & Events, of SYS-CON Media. He is Conference Chair of the AJAXWorld Conference & Expo series and of the "Real-World Flex" One-Day Seminar series. From 2000-6, as first editorial director and then group publisher of SYS-CON Media, he was responsible for the development of all new titles and i-Technology portals for the firm, and regularly represents SYS-CON at conferences and trade shows, speaking to technology audiences both in North America and overseas.

Meet Robert

A business executive at a popular social media site

Challenges

- Encountering poor video quality due to exponential growth in traffic.
- Increasing bandwidth costs due to growing audience size.
- Experiencing compatibility issues due to user-generated video arriving in multiple formats.

Solutions

- VitalStream Streaming Services – Improved quality of end-user video experience using a scalable and global content delivery network.
- VitalStream Advertising Services – Transformed the delivery of digital media from a cost center into a profit center.
- VitalStream Transcoding Services – Automatically converted all user-generated content into the leading streaming media format.

Providing End-to-End Solutions for Your Business

VitalStream is more than a rich media delivery company. We are your professional partner providing solutions that meet your unique business challenges. To learn more about VitalStream solutions, call 800-254-7554 or visit www.vitalstream.com/go/solutions/







Aral Balkan

Aral Balkan is founder and managing director of Ariaware, a London-based company offering products like Ariaware Optimizer and the open-source Ariaware RIA Platform (ARP 2.0) for Flash developers. Ariaware also offers RIA development process and usability consulting and development services. Aral holds an MA in Film and Electronic Media, is a Macromedia Certified Instructor and is celebrating his 20th year as a programmer (he's only 27!). His passions include software architecture and Human-Computer Interaction - in other words, building solid, usable applications. He's co-author of "Flash MX Most Wanted Components" and "Flash 3D Cheats Most Wanted," as well as author and editor of numerous articles for Adobe Developer Center and Ultrashock.com.



Erik Bianchi

Erik Bianchi is a software engineer with more than five years of experience developing Flash-based RIAs and enterprise-wide applications for Fortune 50 and 500 companies. In his spare time he enjoys building Flash-based games, writing or tech editing Flash-related books, and when burned out on code, playing video games on his PC/console systems. You can get more info about Erik and his latest projects on his blog at www.erikbianchi.com.



Craig Goodman

Craig Goodman is the managing editor of Adobe's Developer Center. He and his team publish the tutorials and articles in the area. Craig joined Macromedia in 1995 and his past roles include managing web support and supervising product technical support for Macromedia Flash.



Jim Phelan

Jim Phelan is vice president of development for Stream57, a New York City based firm specializing in communication solution development for the enterprise. Jim's expertise in creating solutions for consolidation and collateralization of business communications has allowed his team to create applications for the management and delivery of live and on demand rich media content. Jim is a strong proponent of the Adobe Flash Platform and is a member of the editorial board of MX Developer's Journal.



Andrew Phelps

Andrew M. Phelps is in the Information Technology Department at the Rochester Institute of Technology in Rochester, NY (<http://andysgi.rit.edu/>).



Darron J. Schall

Darron J. Schall has been programming long before he could drive. In school he studied programming languages, ranging from Basic to Pascal to C++ and eventually moving into Java and C# throughout college. Somewhere in the middle he got hooked on Flash 5 and it's been a crazy love affair ever since. Darron is an independent consultant specializing in RIA development. He maintains a Flash Platform related weblog (www.darronschall.com) and is an active voice in the Flash and Flex communities.



Stephanie Sullivan

Stephanie Sullivan is a Web developer, partner at CommunityMX (www.communitymx.com), owner of VioletSky Design (www.violetsky.net), and contributing author of Dreamweaver MX 2004 Magic.




Jeff Tapper

Jeff Tapper, co-founder of Tapper, Nimer and Associates, has been developing Internet-based applications since 1995, for a myriad of clients including Toys R Us, IBM, Allaire, Dow Jones, American Express, M&T Bank, Verizon, Allied Office Supplies, and many others. As an Instructor, he is certified to teach all of Adobe's courses on Flex, ColdFusion and Flash development. He has worked as author and technical editor for several books on technologies including Flex, Flash and ColdFusion, such as "Object Oriented Programming with ActionScript 2.0," and "Flex 2 Training from the Source."



Jesse Randall Warden

Jesse R. Warden is a senior Flash developer at Surgical Information Systems, an operating room software company, where he currently uses Flash MX, Flash Remoting, .NET, and Oracle to create next-generation rich Internet applications for the OR. He contributed four chapters to the Flash Communication Server MX Bible and has written articles for various publications, including one for Macromedia for a DRK. 

```
function optimizeRIA() {  
    if (omniture.actionsource == true) {  
        businessSuccess();  
    } else {  
        if (javascript.futile == true) {  
            businessFail();  
        }  
    }  
}
```

businessSuccess();



OMNITURE®
ActionSource™

A new way to measure the impact of your **Rich Internet Applications**.
No JavaScript required. No hassles. **More** accurate. **More** success.

WANT TO LEARN MORE ABOUT OMNITURE ACTIONSOURCE™?

→ www.omniture.com/actionsource

1.877.722.7088

© SEPTEMBER 2006 Omniture, Inc. Omniture, and the Omniture logo are trademarks of Omniture. All other trademarks and logos are the property of their respective owners. All rights reserved.

OMNITURE®
— — —

Beyond the Browser

The next generation of Rich
Internet Applications
by Kevin Lynch

The growth of Flash and AJAX in Web applications is driven by real market needs – applications that are visually compelling and simple to use gain faster adoption and can be a competitive differentiator, enabling customers, employees, and partners to interact effectively with information and other people. There has been tremendous innovation in applications delivered via the Web; however, browser limitations such as the lack of access to local files, the inability to leverage desktop functionality, and reliance on continuous connectivity ultimately limit the functionality of a browser-based application. In addition, creating these applications is not always a simple process and browser compatibility issues continue to plague front-end developers.

While there are many Web applications that will continue to thrive in the browser, the need for better access to local audio and video assets, and seamless integration between desktop and Web services is driving the next generation of Rich Internet Applications (RIAs). The new RIAs will bring added flexibility and control to developers and engaging application experiences to users by combining the capabilities of desktop applications with the reach of the Web.

A New Class of Applications: RIAs on the Desktop

Adobe is currently working on a project code-named Apollo, which is a new cross-OS, cross-device application run-

time to extend the reach of RIAs to the desktop. With Apollo, Web developers will be able to leverage their existing skills in HTML, XML, JavaScript, AJAX, Flash or Flex to build RIAs that break free of browser and platform constraints, allowing them to run on the desktop.

Since Apollo is a cross-platform application runtime, it has little or no visible UI itself, so developers have complete creative control over the application experience they provide to end users. Apollo eliminates cross-browser testing and the need for constant page refreshes, ensuring consistent functionality and interactions across desktops. In addition, Apollo supports seamless online and offline usage, and rich media plays smoothly and reliably. The result is a new class of applications that enhance end-user application experiences without the complexity of traditional desktop development.

End users can interact with applications running on Apollo in the same way that they interact with native desktop applications. Users will go through a familiar install process when placing Apollo applications on their desktops, launch the applications the same way they launch other desktop applications, and uninstall applications in the same way they do today. The end result is that users engage with Apollo applications in the same way as they do with traditional desktop applications, but now they will get a more compelling experience offline as well as online with a rich media appli-

cation that supports video, audio, HTML, Flash, and PDF.

Of course, not all RIAs need to run outside of a browser on the desktop or integrate tightly with the OS. There will be classes of RIAs that work sufficiently within the browser. Some RIAs may have elements in the browser and on the desktop, and it may make sense for other RIAs to live only outside the browser on a desktop. Going beyond the browser is sensible when developers need to:

- Support users in offline as well as online environments
- Break free of the browser and its chrome
- Establish a closer relationship with the user through often-used applications
- Integrate directly with the desktop to enable capabilities such as drag and drop, clipboard, system tray icons, and file extension registration
- Run applications in the background to bring information to the user

Build Applications Using Familiar Web Technologies

Apollo is an extension of existing workflows, and enables developers to work in familiar environments, leveraging the skills, tools, and approaches that are most comfortable. By supporting HTML, XML, JavaScript, AJAX, Flash, Flex and incorporating PDF, Apollo allows developers to build the best possible experience that meets their needs. Whether using IDEs such as Flex Builder, which is built on Eclipse, or Web

As senior vice president and chief software architect, Kevin Lynch leads Adobe's Platform Business Unit, which is focused on advancing the company's software platform for the creation and delivery of engaging applications and content to any desktop or device. He joined Adobe through the company's 2005 acquisition of Macromedia, Inc., where he served as chief software architect and president of product development. Kevin holds three patents with others currently pending, and he is actively involved in Adobe's international standards efforts with organizations such as the W3C, ECMA and ISO.



Kevin Lynch

Chief Software Architect,
Adobe

design tools such as Flash or Dreamweaver, or design tools for PDF, developers can quickly create and deploy applications.

Apollo will provide a set of APIs that can facilitate different capabilities, such as system file access, windowing, background processing, and system tray/toast notifications. These APIs will be accessible within JavaScript and ActionScript (both ECMAScript standard languages) for use by applications running on Apollo. Their capabilities are abstracted so that developers need not worry about the underlying implementation for each operating system. The Apollo runtime will take care of integration with the desktop across all supported operating systems.


By supporting existing Web content and design patterns, developers can reuse many of the same assets and code from their Web-based applications to build applications that run on the desktop. In addition, developers can easily integrate services, such as Flex Data Services for data model synchronization and push and pull notifications, as well as the Flash Media server for bidirectional audio/video.

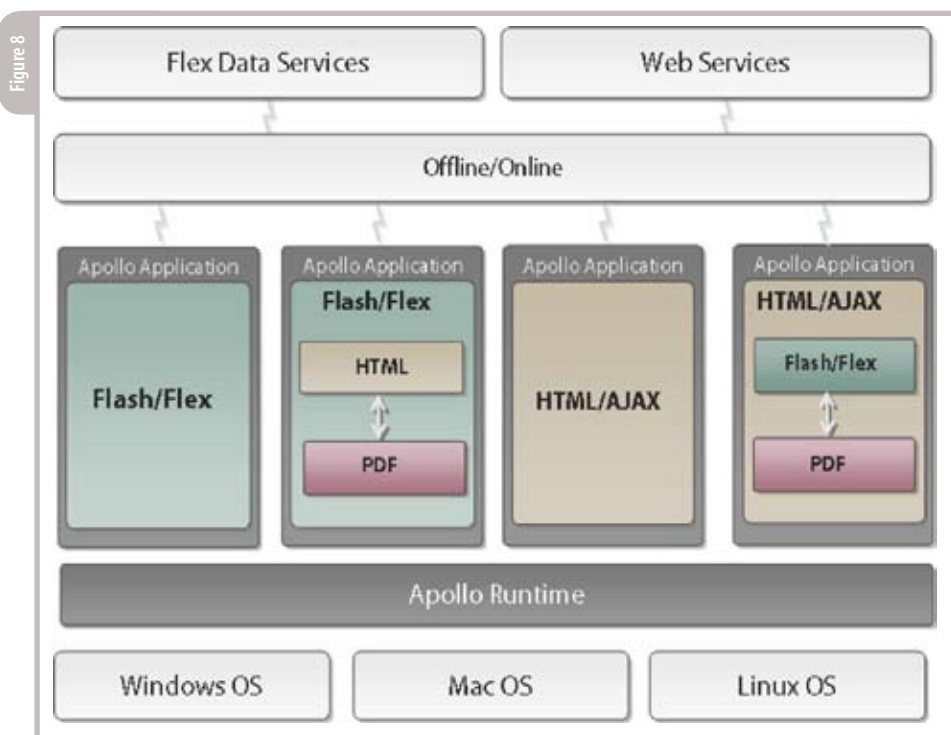
Adobe authoring tools will be optimized for building Apollo applications. If you are using Dreamweaver to build HTML or AJAX applications for the browser, you will be able to take that Web application to the desktop with Apollo. If you are currently developing browser-based RIAs using Flex Builder and the Flex declarative markup language – MXML

– and ActionScript, you can easily extend your Flex applications to the desktop with Apollo. In the future, the same will also be true when using Flash to build applications using ActionScript. If you choose a different tool, you'll be able to leverage the free Apollo SDK, which provides a command-line utility for packaging Apollo applications.

Apollo gives developers a new paradigm that dramatically changes how applications can be created, deployed, and experienced. Developers gain

more creative control and can extend their work to the desktop, without needing to learn complex desktop development technologies. Given the tremendous creativity displayed by Web developers worldwide, the opportunities for taking RIAs outside the browser are virtually limitless.

For more information on Apollo and building Apollo applications, visit <http://www.adobe.com/go/apollowiki>. To learn more about Flex, visit <http://www.adobe.com/products/flex/>. 



Web and Mobile Convergence

Sharing Data Between Flex 2 and Flash Lite

Developing a blog aggregator
by Marco Casario

I love the definition in Wikipedia of Media convergence: “Media convergence is a theory in communications where every mass medium eventually merges to the point where they are indistinguishable to each other due to the advent of new communication technologies.”

According to the theory of media convergence, very soon, there will be no more need to have a television and a computer separate from each other, since both would be able to do the job of the other, ultimately making both extinct and creating a new medium from the synthesis.”

Even though the prospect of a complete merger among “traditional” media is still far away, the fact that we have access to content on multiple devices (internet, cellular phone, PDA, television) already enhances the usefulness that such content provides to users.

Adding to Wikipedia’s definition, convergence also involves the possibility of deploying content through different channels from a unique source in a simple, quick, and cost-effective way, such as the one allowed by the use of Flex, which I attempt to describe in this article.

Nowadays, we have all the technologies that permit developers to create cross-device applications. By sharing and accessing the same remote data (via a database connection, using simple text

file or feeding XML), it’s possible to have interchangeable content and use them via a mobile phone rather than via a Web application.

Flex 2 and Flash Lite Together

Soon the next generation of Adobe’s technology will allow developers to deploy their Flex (under SWF format) applications out of the browser, directly through the desktop.

For those who have never heard about Apollo, this is the definition given by Adobe: “Apollo is the code name for a cross-operating system run-time being developed by Adobe that allows developers to leverage their existing Web development skills (Flash, Flex, HTML, JavaScript, Ajax) to build and deploy Rich Internet Applications (RIAs) to the desktop” (<http://labs.adobe.com/wiki/index.php/Apollo:developerfaq>).

While waiting for this killer application, we can now use Flex 2 to create a rich interactive Web application that will add and edit remote data. The same remote data is then served and consumed by a mobile application developed in Flash Lite. In this article, we’ll see how to create a Flex 2 blog aggregator where users will be able to add, view, and save their favorite RSS feeds.

Each blog post can be bookmarked and stored by users and then viewed under a Flash Lite application. Let’s see how it works.

The Database and the PHP Remote Services

To start, create the database that you’ll use to store all the blog feeds data saved by the user. I chose MySQL database, but you can use whatever you like. I called my db “rssimport”. Next, create three tables that will hold the user data. Here is the SQL structure for our tables:

```
CREATE TABLE `bookmarks` (
  `id` int(11) NOT NULL auto_increment,
  `id_news` int(11) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1
AUTO_INCREMENT=6 ;
```

This table contains a reference to the news table:

```
CREATE TABLE `rss_news` (
  `id` bigint(20) NOT NULL auto_increment,
  `id_sito` int(11) NOT NULL default '0',
  `category` varchar(20) default '0',
  `title` varchar(255) NOT NULL default '',
  `summary` text,
  `pub_date` int(11) NOT NULL default '0',
  `link` varchar(255) NOT NULL default '',
  `creator` varchar(100) NOT NULL
```

Marco Casario is CEO of Comtaste, a company devoted to developing Rich Internet Applications on the Web and for mobile devices. He collaborates intensively with Adobe Italy as a speaker at conferences and as a consultant for Flash, Flex, and Flash Lite. Learn more about Marco Casario at his blog <http://casario.blogs.com>.

```
default '',
    PRIMARY KEY ('id'),
    KEY 'link' ('link')
) ENGINE=MyISAM DEFAULT CHARSET=latin1 AUTO_
INCREMENT=36;
```

This table is populated by a Cron service. Cron is the name of a program that enables UNIX users to execute commands or scripts (groups of commands) automatically at a specified time/date. It's normally used for sysadmin commands. In our example, we use a Cron service to retrieve blog posts from the different blogs found on "testate" table:

```
CREATE TABLE `testate` (
  `id` int(11) NOT NULL auto_increment,
  `nome_testata` varchar(50) NOT NULL default '',
  `preview` varchar(255) default NULL,
  `url` varchar(100) NOT NULL default '',
  PRIMARY KEY (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1 AUTO_
INCREMENT=5 ;
```

The database is done. We've got our tables, so now create the PHP scripts that will add blogs to the database, retrieve information over blogs, save posts, and export all the data in the XML that the Flex application will consume. The PHP services are pretty simple, but they're large. You can download all the files here:

<http://88.149.156.198/develop/rssToDb/phpservices.zip>

This package contains four PHP files. These are the files called by the Flex's RPC service. All PHP files return values in a proper XML format. In fact, data must be encoded in XML in order to pass them to Flex.

Commenting the PHP file is out of the scope of this article, but if you develop in PHP, the code should be self explanatory. If you're not a PHP developer, don't worry – you'll be able to finish the application and make all the code work anyhow.

Developing the Flex 2 Blog Aggregator

It's now time to start building the Web interface that will allow users to add and save their favorite blogs. Create a new Flex project in Flex Builder 2 (File >

New > Flex Project) and give it a name. I've called mine "BlogRoll."

Automatically, some files and folders have been created by Flex Builder 2. The most important ones are the bin folder, where Flex Builder puts all the compiled code and the main MXML application file.

We need to create another folder and name it "components." This is the folder where external MXML Components will be stored. To create a new folder in the context of the project, go to File > New > Folder. Our main application is made up of a TabNavigator container that houses two states for adding and viewing blogs. The TabNavigator is a navigation container that creates and manages the set of tabs used to navigate among its children. The children of a TabNavigator container are other containers.

The container automatically creates a tab for each of its children and determines the tab text from the label property of the child, and the tab icon from the icon property of the child.

When a TabNavigator gets the focus, the Flash Player processes keystrokes as described in Table 1 (see the Flex 2 LiveDocs to learn more):

Let's write some MXML code to create our interface. Use the BlogRoll.mxml file that Flex Builder has previously created when you defined your project. Let's examine in detail the code for the main application file:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.
com/2006/mxml"
  layout="vertical"
  xmlns:cust="components.*"
  creationComplete="hs.send()"
  themeColor="haloSilver"
  pageTitle="Blog Aggregator - Developed by
Marco Casario for WebDDJ" >
```

We start creating the application tag where we defined a customized namespace:

```
xmlns:cust="components.*"
```

This is the package where we'll insert our MXML components (it points to the folder we previously created). Then we call the send() method of the HTTPService to populate the Datagrid. The method triggers after all

SYS-CON Media

CEO

Fuat Kircaali, 201 802-3001
fuat@sys-con.com

President & COO

Carmen Gonzalez, 201 802-3021
carmen@sys-con.com

Sr. Vice-President, Editorial & Events

Jeremy Geelan, 201 802-3051
jeremy@sys-con.com

Advertising

Vice President, Sales & Marketing

Miles Silverman, 201 802-3029
miles@sys-con.com
Robyn Forma, 201 802-3022
robyn@sys-con.com

Advertising Sales Manager

Megan Mussa, 201 802-3023
megan@sys-con.com

Associate Sales Managers

Kerry Mealia, 201 802-3026
kerry@sys-con.com
Lauren Orsi, 201 802-3024
lauren@sys-con.com

Production

Lead Designer

Louis F. Cuffari, 201 802-3035
louis@sys-con.com

Art Director

Alex Botero, 201 802-3031
alex@sys-con.com

Associate Art Directors

Abraham Addo, 201 802-3037
abraham@sys-con.com
Tami Beatty, 201 802-3038
tami@sys-con.com

Assistant Art Director

Mandy Eckman, 201 802-3032
mandy@sys-con.com

SYS-CON.COM

Consultant, Information Systems

Robert Diamond, 201 802-3051
robert@sys-con.com

Web Designers

Stephen Kilmurray, 201 802-3053
stephen@sys-con.com
Paula Zagari, 201 802-3042
paula@sys-con.com

Accounting

Financial Analyst

Joan LaRose, 201 802-3081
joan@sys-con.com

Accounts Payable

Betty White, 201 802-3002
betty@sys-con.com

Accounts Receivable

Gail Naples, 201 802-3062
gailn@sys-con.com

Customer Relations

Circulation Service Coordinator

Edna Earle Russell, 201 802-3081
edna@sys-con.com

the user interface elements are loaded on the creationComplete event, as shown in Listing 1.

This is where we set up the HTTPService and call two PHP remote services: rss.php and add.php. The former returns all the posts that users saved. It accepts one parameter defined inside the <mx:request xmlns=""> tag to indicate the quantity of posts returned by the function: 10, 20 or 30 posts. Then on the result event, it populates the "listAC" ArrayCollection with data returned by the response.

The second PHP service - add.php - receives the name and the URL of the blog inserted by the user. The two variables - blog_name and url_blog - are set to the text attribute of the two TextInput with id "frm_name" and "frm_url". The HTTPService handles the result and the fault events. For both services, we use the POST method to send data, as shown in Listing 2.

I strongly commented the Actionscript code, as shown in Listing 3, so we could go ahead.

We put a <mx:TabNavigator> inside a panel. The TabNavigator has two children. On the first VBox child, we need to create a simple form that contains the two Text Input elements: frm_name and frm_url.

At the end of the form, we placed a Submit button that calls the send() method of the HTTPService with the addBlog id. This remote service stores the name and the URL of the blog added by the users and puts it into the MySQL database (see Figure 1).

The second child of the TabNavigator is made up of a simple Label, a Combo Box, and an MXML Component. The ComboBox is populated by the ArrayCollection "myArray" defined as a nested tag:

```
<mx:ComboBox id="myCombo"
change="changeHandler()" creationComplete="init();"myCombo.selectedIndex=0" >
<mx:ArrayCollection id="myArray">
    <mx:Object label="20"
posts" data="20"/>
    <mx:Object label="30"
posts" data="30"/>
</mx:ArrayCollection>
</mx:ComboBox>
```

When the user selects an option from the combo box, the changeHandler() function is called. The creationComplete event triggers the init() function and the index of the combobox is set up to the first element.

The MXML Component's invocation uses the "cust" namespace defined on the application tag. The "lista" property is passed to the component, and it contains the data returned by the HTTPService request:

```
<cust:custBlogData width="100%"
lista="{hs.lastResult.rss.channel.
item}" />
```

The code of the custBlogData.mxml file saved in the "components" folder is shown in Listing 4. The file's role is to display the titles and links of all blogs added by the users. The DataGrid populates itself with the lista property (data typed as ArrayCollection) that we get from the HTTPService. It shows two columns that display the title and link property contained into the complex lista object:

```
<mx:DataGridColumn headerText="Posts"
dataField="title" width="150" />
<mx:DataGridColumn headerText="Link"
dataField="link" />
```

At the end, a Hbox container surrounds a label and a Button that, if clicked, calls the send() method of the HTTPService:

```
<mx:HTTPService
id="saveHS"
url="http://88.149.156.198/develop/rssToDb/savenews.php"
useProxy="false"
method="POST"
result="linkHandler(event)"
fault="faultHandler(event)">

<mx:request xmlns="">
<url>{myDG.selectedItem.link}</url>
</mx:request>
</mx:HTTPService>
```

The HTTPService request waits for a parameter that represents the URL of the post selected by the user in the datagrid control. These saved values are stored

into the database and will be requested by the Flash Lite application. So that all the news bookmarked will be available via a Flash Lite enabled mobile phone.

For our purposes, this is enough, but you could go further in extending the Flex application.

You should use the StringValidator tag to validate the data inserted by the user into the Text Inputs so that you avoid errors on the business tier. Or, you could add some styles or customizations to the user interface using Cascading Style Sheets (CSS).

The Mobile Application: Creating a Flash Lite XML Parser

All the data is stored in the MySQL database. Unfortunately, Flash Lite is not able to access a database directly. Therefore, in order to access data, we need to develop a PHP function that makes the connection and returns the values to our Flash Lite interface.

You can try it by launching the URL, <http://88.149.156.198/develop/rssToDb/rssmobile.php>. However, you will see that another limitation occurs because the Flash Lite 1.1 player cannot parse XML directly. It can load dynamic data via the loadVariables function, using the name/value pair data structure. Using this method, you can load almost any data within Flash Lite 1.1 content.

We were inspired by a ready-to-use PHP RSS parser written by Alessandro Pace who explained in an article for Adobe Developer Center (www.adobe.com/devnet/devices/articles/flash-litell_rss.html) how to create a Flash Lite 1.1 RSS (Really Simple Syndication) reader using a server-side PHP RSS parser.

With the following PHP 5 XML parser, a RSS 2.0 feed will be read to extract the <title> and <description> tags contents for each <item> block, and then the content will be customized so it can be loaded into the Flash Lite 1.1 RSS reader on the mobile phone, as shown in Listing 5.

The values returned by the echo will produce the following output:

```
&title1=Tips for Developing
Flash Games for the iRiver
U10&description1=New article on DevNet
at Adobe: Tips for Developing Flash
```


BALANCE

Designer/developer, front-end/back-end, clients/sanity. . . web development is a balance and we can help you maintain it. Join now and experience a wealth of training resources tailored to the tools you use every day.

www.communitymx.com



Visit www.communitymx.com/trial/ for your free 10 day trial.



Games for the iRiver U10. It includes two samples available for download. The article is written by Sung-Hee Park of MiniGate. Alessandro

The code is pretty simple and well commented but if you want further explanation, you can read the "Analyze the Code" section of the tutorial (http://www.adobe.com/devnet/devices/articles/flashlite11_rss_02.html).

It's now the turn of the Flash Lite user interface. You can download the complete application on the Adobe Devnet: Flashlite11_rss_samples.zip (http://download.macromedia.com/pub/developer/flashlite11_rss_samples.zip). Opening the Flash file, you'll see the application is made up of 5 frames and 3 layers. The most important part of code is contained in frame two, where the call to the server is made using the loadVariable function:

```
loadVariable (http, "");
```

The "http" variable points to the server-side PHP file's URL, and it's defined on frame one:

```
http = "http://88.149.156.198/develop/rssToDb/mobileparser.php";
```

The loadVariable function reads data from an external file, such as a text file or text generated by server-side script (Coldfusion, PHP ...), and sets the values for variables in a target movie clip. The text at the specified URL must be in the standard MIME format application/x-www-form-urlencoded (a standard format used by CGI scripts):


```
&title1=more on  
apollo&description1=news  
from apollo&title2=flex url  
kit&description2=a new kit for han-  
dling with URL has been released
```

All the navigation features are hidden on frame five inside the hidden button used to select the variables title and description and to scroll the dynamic text contained in the description variable, as shown in Listing 6.

Conclusion

The game is over. Thanks to the Flash

Platform, in just a few simple steps, we built a simple Flex 2 application that manages data with PHP from a MySQL database. Data is then exposed and presented through a mobile application that uses Flash Lite as development technology. Users can reach through their cellular phones blog posts they previously selected and stored via a Flex 2 Web application.

The source code for this article can be downloaded from the online version of this article at webddj.sys-con.com. 

Listing 1

```
<mx:HTTPService  
    id="hs"  
    url="http://88.149.156.198/develop/rssToDb/rss.php"  
    useProxy="false"  
    result = "listAC = hs.lastResult.  
    rss.channel.item as ArrayCollection" >  
  
    <mx:request xmlns="">  
        <qty>{myCombo.value}</qty>  
    </mx:request>  
    </mx:HTTPService>  
  
    <mx:HTTPService id="addBlog"  
        url="http://88.149.156.198/develop/rssToDb/add.php"  
        useProxy="false"  
        method="POST"  
        result="blogAddedHandler(event)"  
        fault="faultHandler(event)">  
        <mx:request xmlns="">  
  
            <blog_name>{frm_name.text}</blog_  
name>  
            <url_blog>{frm_url.text}</url_blog>  
  
        </mx:request>  
        </mx:HTTPService>
```

Listing 2

```
<mx:Script>  
    <![CDATA[  
        import mx.collections.  
        ArrayCollection;  
        import mx.rpc.events.  
        ResultEvent;  
  
        // This is the  
        ArrayCollection variable that store  
        the  
  
        // values returned by  
        HTTPService's request
```

```
[Bindable]  
private var listAC:ArrayCollection;  
  
// The init function is called by  
the combobox element.  
// It adds at the first item of  
the ArrayCollection named  
// myArray the value of 10  
  
private function init():void  
{  
    myArray.addItemAt({label:"10  
posts", data:"10"}, 0);  
  
}  
  
// This function is called when  
the user selects  
// an item from the combo box.  
// It simply calls the send()  
method of the HTTPService  
// with an id "hs"  
  
private function changeHandler():  
void  
{  
    hs.send();  
}  
  
// The event handler of the  
HTTPService's result event  
// It makes  
// the text input empty. Then it  
calls again the  
// send() method of the  
HTTPService  
  
private function  
blogAddedHandler(event:ResultEvent):  
void  
{  
  
    frm_name.text = "";  
    frm_url.text = "";  
    hs.send();  
    mx.controls.Alert.show ("Blog  
added", "HTTPService's Result Event");  
  
}  
  
// The event handler of the  
HTTPService's fault event  
// This is a function to handle  
potential errors that  
// occur during a HTTPService
```

```

request

    private function
faultHandler(event:mx.rpc.events.
FaultEvent):void
    {
        var faultInfo:String="fault
description: "+event.fault.faultDe-
tail+"\n\n";
        faultInfo+="fault faultstring:
"+event.fault.faultString+"\n\n";
        mx.controls.Alert.
show(faultInfo,"Fault Information");

    }

    ]]>
</mx:Script>

```

Listing 3

```

<mx:WipeLeft id="WL" />

<mx:Panel title="Flex 2 Blog
Aggregator" >

<mx:Image source="http://www.com-
taste.com/images/logo.gif" />

<mx:Text text="Insert the blog you
want to add to your preferite list."
height="28"/>

<mx:TabNavigator width="350"
height="350">
    <mx:VBox label="Add Blog"
width="100%" height="100%"
showEffect="WL">
        <mx:Form width="100%">
            <mx:FormHeading label="Insert a
new blog"/>
            <mx:FormItem label="Name">
                <mx:TextInput id="frm_name" />
            </mx:FormItem>
            <mx:FormItem label="URL's
address">
                <mx:TextInput id="frm_url" />
            </mx:FormItem>
            <mx:HBox>
                <mx:Spacer width="50" />
                <mx:Button label="Add" id="btn_
add" click="addBlog.send()" />
            </mx:HBox>

        </mx:Form>
    </mx:VBox>
</mx:TabNavigator>

```

```

</mx:VBox>
    <mx:VBox label="View Blogs"
width="90%" height="100%"
showEffect="WL">

        <mx:Label text="Show the latest : "
id="myLbl" fontWeight="bold"/>

        <mx:ComboBox id="myCombo"
change="changeHandler()" creationCompl
ete="init();myCombo.selectedIndex=0" >
            <mx:ArrayCollection id="myArray">
                <mx:Object label="20
posts" data="20"/>
                <mx:Object label="30
posts" data="30"/>
            </mx:ArrayCollection>
        </mx:ComboBox>

        <cust:custBlogData width="100%"
lista="{hs.lastResult.rss.channel.
item}" />

    </mx:VBox>

</mx:TabNavigator>

<mx:ControlBar>
    <mx:Label text="Developed by Marco
Casario" />
    <mx:LinkButton label="http://casario.
blogs.com" click="navigateToURL(new
URLRequest('http://casario.blogs.
com'),'_blank');"/>
</mx:ControlBar>

</mx:Panel>

</mx:Application>

```

Listing 4

```

<?xml version="1.0" encoding="utf-8"?>
<mx:VBox xmlns:mx="http://www.adobe.
com/2006/mxml">

    <mx:HTTPService
        id="saveHS"
        url="http://88.149.156.198/devel-
op/rssToDb/savenews.php"
        useProxy="false"
        method="POST"
        result="linkHandler(event)"
        fault="faultHandler(event)">

```

```

<mx:request xmlns="">
    <url>{myDG.selectedItem.link}</
url>
</mx:request>
</mx:HTTPService>

<mx:Script>
<![CDATA[
    import mx.collections.
ArrayCollection;
    import mx.rpc.events.ResultEvent;

    [Bindable]
    public var lista:ArrayCollection;
    [Bindable]
    private var counter:Number;

    private function linkHandler(event:
ResultEvent):void
    {
        counter = event.result.list.
total;
        counterLbl.text = "Post saved so
far : " + counter;
        mx.controls.Alert.show ("Post has
been stored", "HTTPService's Result
Event");

    }
    private function
faultHandler(event:mx.rpc.events.
FaultEvent):void
    {
        var faultInfo:String="fault
description: "+event.fault.faultDe-
tail+"\n\n";
        faultInfo+="fault faultstring:
"+event.fault.faultString+"\n\n";
        mx.controls.Alert.
show(faultInfo,"Fault Information");

    }

    ]]>
</mx:Script>

<mx:Style>
    DataGrid {
        depthColors: #EAEAEA, #FF22CC,
#FFFFFF;
        alternatingItemColors: white,
gray;
    }
</mx:Style>

```

“While waiting
for this killer
application, we
can now
use Flex 2
to create a rich
interactive Web
application that
will add and edit
remote data”

```
<mx:DataGrid id="myDG"
dataProvider="{lista}" width="100%" >
    <mx:columns>
        <mx:DataGridColumn
headerText="Posts" dataField="title"
width="150" />
        <mx:DataGridColumn
headerText="Link" dataField="link" />
    </mx:columns>
</mx:DataGrid>

<mx:HBox>
    <mx:Label text="Add this news to
your bookmark : " />
    <mx:Button label="Save"
click="saveHS.send();" />
</mx:HBox>

<mx:Label id="counterLbl" />

</mx:VBox>
```

Listing 5

```
//Link to RSS 2.0 feed
$link_to_rss_feed
="http://88.149.156.198/develop/rss-
ToDb/rssmobile.php";

//Counter to append to &variables to
return to the client
$i=1;

// This PHP 5 function loads the
entire XML feed
$xml = simplexml_load_file($link_to_
rss_feed);

// Parse each item in the RSS feed

foreach ($xml->channel[0]->item as
$item)
{
    if($item) {
        // Need to strip html tags
since Flash Lite does not support
        // html in dynamic text
fields
        $title = strip_tags($item-
>title);
        $description = strip_
tags($item->description);

// Replace special symbols &
        $title = str_replace("&", "",
```

```
$title);
        $description = str_
replace("&", "", $description);

// Return the content with the appro-
priate index
        echo "&title$i=".$title;
        echo "&description$i=".$desc
ription;
        //echo $i;
        $i++;
    } else {
        // Send and Error message if
content is not found
        echo "&msg=Content not
found";
        echo "&flag=error";
        echo "&end=end";
        exit;
    }
}

$i=$i-1;
echo "&flag=ok";
echo "&totalitems=".$i;
echo "&end=end";
```

Listing 6

```
// Shifting variables
on (keyPress "<Right>") {
    if (n ne totalitems) {
        n++;
    } else {
        n = 1;
    }
    title = eval("title" add n);
    description = eval("description"
add n);
}

on (keyPress "<Left>") {
    if (n ne 1) {
        n- -;
    } else {
        n = totalitems;
    }
    title = eval("title" add n);
    description = eval("description"
add n);
}

//Scrolling
on (keyPress "<Up>") {
    description.scroll--;
}

on (keyPress "<Down>") {
    description.scroll++;
```


CF_UNDERGROUND VIII PRE-MAX EVENT



DATE: SUNDAY, OCTOBER 22, 2006

LOCATION: THE BEACH, LAS VEGAS

Speakers include:
Simon Horwith
Michael Smith
Glenda Vigoreaux
and more...

365 Convention Center Drive, Las Vegas, Nevada

Meet with old friends and new before the
MAX event. A full day of networking,
ColdFusion, great talks, drinks, food, and fun.

WWW.CF-UNDERGROUND.COM



TeraTech
www.teratech.com

COLD FUSION Developer's Journal

Scary Question.

Exactly who is developing your next app?



Contact Us



Address:

555 Not My Home St.
Big City, CO 12345



Your App Starts Here.

We are the leaders in RIA development services.

INCREDIBLE APPLICATIONS

PASSIONATE USERS

PROVEN SUCCESS



Unlock your potential

with the help of industry leaders in
Rich Internet Application development.

10 Years. 1400+ Customers.

Your app starts here.

CYNERGYSYSTEMS.COM



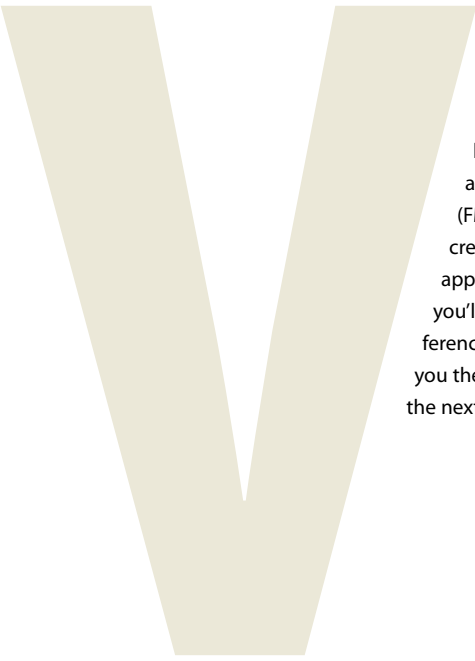
Solution
PARTNER



Video Conference with Flex & FMS

Learn how to use Flex 2 and FMS 2 by creating a
basic video conference application

by Renaun Erickson



video over the Internet
and Rich Internet
Applications (RIA) are the
latest craze. With Flex 2
and Flash Media Server 2
(FMS), developers can easily
create interactive media Web
applications. In this article
you'll create a basic video con-
ference application that will give
you the foundation to take you to
the next step.



Flash Media Server 2 software has been in the wild since November 2005. In the last year, the use of media on the Web - especially video - has taken off. From streaming video to real-time multi-user interactive applications, FMS is the tool to use. For example, chat, video, and audio conferencing, as well as whiteboard sharing applications can be built upon the FMS technology. The user clients are typically built with the Flash IDE, but this article will show you how to put it all together with Flex 2 and FMS. Flex 2 has made it much easier to create high, impactful RIAs. With each new wave of technology, features are built, and interconnectivity standards have to be explicitly stated. You will learn about a subtle change in the Flash Player 9 that Flex 2 uses. The subtle change will need to be addressed to create our video conference application in Flex and FMS.

The Flash Player 9 introduced a newer Action Message Format version called AMF3. Flex 2 creates ActionScript 3.0 SWF's, but Flash Media Server is still based on ActionScript 2.0 and the earlier AMF0. By default, the Flex 2 NetConnection class uses AMF3, therefore you must tell the NetConnection what object encoding to use. There are two ways to do this:

```
var nc:NetConnection = new
NetConnection();
nc.objectEncoding = flash.net.
ObjectEncoding.AMF0;
```

or

```
NetConnection.defaultObjectEncoding =
flash.net.ObjectEncoding.AMF0;
```

The first method changes the object encoding for the instantiated object only, whereas the second method changes the object encoding for all NetConnections globally. With the NetConnection set to use AMF0, the Flex application will know how to communicate with the FMS application properly.

Another common pitfall has to do with understanding when to connect Flex components to the FMS application. You need to wait for the NetConnection to return a status of NetConnection.Connect.Success before connecting any Flex component

to the FMS server-side component. This is done by adding a status event listener on the NetConnection with a function that will consume the response and check on connection success.

```
import flash.events.NetStatusEvent;
.....
nc.addEventListener( NetStatusEvent.
NET_STATUS, netStatusHandler );
.....
public function netStatusHandler(
event:NetStatusEvent ):void
{
    switch( event.info.code ) {
        case "NetConnection.Connect.
Success":
            // Now connect Flex components
            ;
            break;
        }
    }
```

Now it's time to create the FMS back end. I assume that you understand the basic layout and application structure of the FMS server. You can visit the FMS documentation to gain a basic tutorial on how to install, configure, and deploy FMS applications. For this article, you will want to create a flex_videoconference folder under the applications folder in the default install of FMS. The default FMS installation folder on a Windows machine is C:\Program Files\Macromedia\Flash Media Server 2\applications; for Linux, it's /opt/macromedia/fms/applications. FMS uses service-side ActionScript (ASC) files. The ASC files are used to create the server-side component and FMS applications. All service-side applications require a main.asc file. For the example video conference application, you will need to create main.asc and add the following ActionScript code (see Listing 1 for full source). The application startup code will initialize and retrieve a shared object users_so that contains the list of users.

```
application.onAppStart = function()
{
    application.users_so = SharedObject.
get("users_so", false);
}
```

When a client connects to the application, the server will accept the connec-

tion, and the user will be added to the users_so shared object.

```
application.onConnect =
function(client, name, identifier)
{
    application.users_so.setProperty(ide
ntifier, name);
    application.acceptConnection(client)
;
}
```

The last part of the main.asc file contains the code to handle client disconnections and remove them from the users_so shared object.

```
application.onDisconnect =
function(client)
{
    application.users_
so.setProperty(client.identifier,
null);
}
```

I want to point out that this application implementation of handling the list of current users as a single remote shared object will not allow different simultaneous video conferences. This can be achieved a couple of different ways which are out of the scope of this article, mainly through the use of FMS application instances or more sophisticated FMS server-side application implementation.

With the back end FMS application ready to go, the next step is to create the Flex client. The basic layout of the sample application uses a ViewStack with two children UI controls, a form to log in, and a video conference viewing control.

```
<mx:ViewStack id="vsMain"
width="100%" height="100%">
    <!-- Login Panel -->
    ...
    <!-- Video Panel -->
    ...
</mx:ViewStack>
```

A simple form with a text box for the name of the user and a submit button that calls createConnection will do just fine. In the full source listing, you will find added code to disable the login form if the user does not have a camera available (see Listing 2).

```
<mx:Panel id="pnlLogin">
<mx:Form>
  <mx:FormItem label="Name:">
    <mx:TextInput
      id="txtName" />
  </mx:FormItem>
  <mx:Button label="Submit"
    click="createConnection()" />
</mx:Form>
</mx:Panel>
```

After the submit button is clicked and the `createConnection` method is called, the Video panel that contains all the current video conference users will be displayed upon a successful connection. To do this, a connection to the FMS server is made. As defined in the main, `asc`, the Flex client needs to send a name and identifier to the server-side application when connecting to the server application.

```
public function createConnection():
void
{
  // Set AMF0 NetConnection objectEncoding
  ...
  var identifier:String = txtName.text;
  while( identifier.search( " " ) > 0 )
    identifier = identifier.replace( "
", "_" );
  nc.connect( "rtmp://flex_videoconference/", txtName.text, identifier );
}
```

The identifier algorithm used is not terribly important, but the code above goes through and swaps all the space characters with underscores. These identifiers are used to publish and play the streaming video. One of the features of FMS server technology is the ability to record the published video on the FMS server. The FMS server uses the published name to create the name of the record FLV file, and depending on the underlying OS, you might want to pay attention to the identifier value.

The `NetConnection` object connects to FMS through a protocol called Real-Time Messaging Protocol (RTMP). The `NetConnection`'s `connect` method requires URL parameters. All other parameters after the URL parameters are sent to the server as arguments to

the `connect` method call. The URL value `"rtmp://flex_videoconference/"` tells the NC to connect to the server running the script's `flex_videoconference` application. You can learn more about RTMP and FMS connection URLs in the FMS live documentation. The server-side application creates a `SharedObject` by the name of `users_so`. The shared object stores the list of users connecting to the application, and provides a list of all connected users to each specific client.

Remember that the connecting of components to the server needs to be done after the `NetConnection` has successfully connected. You can now add the `connectComponents` and the code to switch to video conference view into the `netStatusHandler` function.

```
public function netStatusHandler(
event:NetStatusEvent ):void
{
  switch( event.info.code ) {
    case "NetConnection.Connect.
Success":
      connectComponents();
      vsMain.selectedChild = pnlVideo;
      break;
  }
}
```

In `connectComponents` a client-side `SharedObject` object is created and connected. Flex is asynchronous, and you will need to set an event to listen for any changes to the remote shared object.

```
public function connectComponents():
void
{
  SharedObject.defaultObjectEncoding =
flash.net.ObjectEncoding.AMF0;
  users_so = SharedObject.
getRemote("users_so", nc.uri, false);
  users_so.addEventListener( SyncEvent.
SYNC, usersSyncHandler );
  users_so.connect( nc );
}
```

The sync event handler will monitor the remote `users_so` and retrieve the latest value if anything changes. After receiving a sync event message, the function turns the remote object into an array of users. The array of users then gets created into our `dgUsers`'s `ArrayCollection`.

The `dgUsers` is used as the data provider in video conference view control.

```
public function usersSyncHandler(
event:SyncEvent ):void
{
  var results:Object = event.target.
data;
  var usersArray:Array = new Array();
  for( var a:String in results ) {
    var obj:Object = new Object();
    obj.name = results[ a ];
    obj.identifier = a;
    usersArray.push( obj );
  }
  dpUsers = new ArrayCollection( user-
sArray );
}
```

The video conference view is now ready to be created. The video conference view consists of a panel, tile, and repeater control. The repeater, with `dgUsers` as the data provider, will create a custom `VideoPod` control for each user. Although the video component is custom, the guts are quite simple. You will find the full source of the `VideoPod` in Listing 3. The idea is to first check if the user is a sender or receiver. If it's a sender, you attach the camera and microphone to the `NetStream` object and publish it to the server.

```
public function init():void
{
  ns = new NetStream( nc );
  ...
  camera = Camera.getCamera();
  ...
  ns.attachCamera( camera );
  microphone = Microphone.getMicro-
phone();
  ns.attachAudio( microphone );
  ns.publish( userItem.identifier )
}
```


When the custom `VideoPod` is to display the receiving video feeds, the component creates a stream and calls the `play` method.

```
video.attachNetStream( ns );
ns.play( userItem.identifier );
```

The video class in Flex 2 is used to display video streams. A simple

VideoContainer class is found in Listing 4 and is used in the VideoPod component to make it easier to add any video object as a UIComponent. This is needed because the video class is not a UIComponent, so it can't be added as a child to the panel class directly. The VideoPod now is ready to be put into the video conference view control using the repeater inside a tile control.

```
<!-- Video Panel -->
<mx:Panel id="pnlVideo"
width="100%" height="100%"
title="Welcome { txtName.text }!"
layout="vertical">
<mx:Tile height="100%" width="100%">
<mx:Repeater id="rpUsers"
dataProvider="{ dpUsers }">
<VideoPod
nc="{ nc }"
isSender="{ rpUsers.currentItem.name ==
txtName.text }"
userItem="{ rpUsers.currentItem }" />
</mx:Repeater>
</mx:Tile>
</mx:Panel>
```

Now, you can put it all together, compile it, and try it out for yourself. The application's structure consists of the main mxml application called FlexVideoMain.mxml (see Listing 2), VideoPod.mxml (see Listing 3), and VideoContainer.as (see Listing 4). Complementing the client application the FMS server application requires main.asc (see Listing 1) to be placed in flex_videoconference folder in the applications area. The application provides a simple one room multi-user video conference solution. Of course, the maximum number of users possible in each video conference will depend on bandwidth. There are other considerations in creating live, media-rich applications like latency, quality of the video, and frame rates (which affects perceived latency). 

Listing 1

```
application.onAppStart = function()
{
trace("Begin sharing text");

// Get the server shared object 'users_so'
application.users_so = SharedObject.get("users_
so", false);
}

application.onConnect = function(client, name,
identifier)
{
client.identifier = identifier;
```

```
application.users_so.setProperty(identifier,
name);
application.acceptConnection(client);
}
```

```
application.onDisconnect = function(client)
{
trace("disconnect: " + client.identifier);
application.users_so.setProperty(client.identi-
fier, null);
}
```

Listing 2

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.
com/2006/mxml"
xmlns="*"
initialize="haveCamera = ( Camera.getCamera()
!= null )"
layout="vertical">
<mx:Script>
<![CDATA[
import mx.collections.ArrayCollection;
import mx.controls.Alert;

[Bindable]
private var haveCamera:Boolean;

[Bindable]
public var nc:NetConnection;

public var users_so:SharedObject;
[Bindable]
public var dpUsers:ArrayCollection;

public function createConnection():void
{
if( txtName.text.length > 0 ) {
nc = new NetConnection();
nc.objectEncoding = ObjectEncoding.AMFO;
nc.addEventListener( NetStatusEvent.NET_
STATUS, netStatusHandler );
var identifier:String = txtName.text;
while( identifier.search( " " ) > 0 )
identifier = identifier.replace( " ",
"_" );
nc.connect( "rtmp://flex_videoconfer-
ence/", txtName.text, identifier );
} else {
Alert.show( "Please provide a name for
the video chat connection!" );
}
}

public function netStatusHandler( event:
NetStatusEvent ):void
```



The new MX CSS Menus2 for Dreamweaver

Create and customize CSS menus in Dreamweaver !

The new MX CSS Menus 2 from InterAKT is a Dreamweaver extension that offers you the choice to create your own CSS menu, implement your own ideas in a customized skin and perfectly blend your menu into your site's design. With visual step by step implementation and no CSS knowledge required this new version includes:

- CSS Skin Editor to create your own design
- Support for HTML and database driven sites
- Faster loading in the browser
- Vertical, tabbed, horizontal or expandable menu types
- 23 pre-built CSS skins
- PNG and PSD files included
- Animation effects on sub-levels
- New wizards and interfaces



Buy at only \$99

Free trial available at: **<http://www.interaktonline.com/mxcss>**

“Another
common pitfall
has to do with
understanding
when to
connect Flex
components to the
FMS application”

```

        {
            switch( event.info.code ) {
                case "NetConnection.Connect.
Success":
                    connectComponents();
                    vsMain.selectedChild =
pnlVideo;
                    break;
            }

            public function connectCompo-
nents():void
            {
                SharedObject.defaultObjectEncod-
ing = flash.net.ObjectEncoding.AMFO;
                users_so = SharedObject.
getRemote("users_so", nc.uri, false);
                users_so.addEventListener(
SyncEvent.SYNC, usersSyncHandler );
                users_so.connect( nc );
            }

            public function usersSyncHandler(
event:SyncEvent ):void
            {
                var results:Object = event.tar-
get.data;
                var usersArray:Array = new
Array();
                for( var a:String in results ) {
                    var obj:Object = new Object();
                    obj.name = results[ a ];
                    obj.identifier = a;
                    usersArray.push( obj );
                }
                dpUsers = new ArrayCollection(
usersArray );
            }

            public function logout():void
            {
                users_so.close();
                nc.close();
                dpUsers = null;
                users_so = null;
                nc = null;
                vsMain.selectedChild = pnlLogin;
            }

        ]]>
</mx:Script>

```

```

<mx:ViewStack
    id="vsMain"
    width="100%" height="100%">
    <!-- Login Panel -->
    <mx:Panel id="pnlLogin">
        <mx:Form>
            <mx:FormItem label="Name:">
                <mx:TextInput
                    id="txtName"
                    enabled="{ haveCamera }" />
            </mx:FormItem>
            <mx:Button label="Submit"
                click="createConnection()"
                enabled="{ haveCamera }" />
            <mx:Label text="{ ( haveCamera )
? '':'Now camera is found!' }" />
        </mx:Form>
    </mx:Panel>
    <!-- Video Panel -->
    <mx:Panel id="pnlVideo"
        width="100%" height="100%"
        title="Welcome { txtName.text }!"
        layout="vertical">

        <mx:Tile height="100%"
            width="100%">
            <mx:Repeater id="rpUsers"
                dataProvider="{ dpUsers }">
                <VideoPod
                    nc="{ nc }"
                    isSender="{ rpUsers.curren-
tItem.name == txtName.text }"
                    userItem="{ rpUsers.curren-
tItem }" />
            </mx:Repeater>
        </mx:Tile>
        <mx:ControlBar>
            <mx:Button label="Logout"
                click="logout()" />

        </mx:ControlBar>
    </mx:Panel>

</mx:ViewStack>

</mx:Application>

```

Listing 3

```

<?xml version="1.0" encoding="utf-8"?>
<mx:Panel xmlns:mx="http://www.adobe.
com/2006/mxml"
    xmlns=""
    title="{ userItem.name } Pod"
    creationComplete="init()"

```

```

        layout="vertical" width="200"
        height="220">
<mx:Script>
    <![CDATA[
        import flash.media.Video;

        private var microphone:Microphone;
        private var camera:Camera;
        private var video:Video;

        [Bindable]
        public var userItem:Object;
        public var nc:NetConnection;
        private var ns:NetStream;

        [Bindable]
        public var isSender:Boolean;

        private function init():void
        {
            ns = new NetStream( nc );
            video = new Video( 180, 180
        );

            if( isSender ) {
                camera = Camera.getCamera();
                //Number(objConfig.quality)
                camera.setQuality( 16384, 0 );
                camera.setMode( 180, 180, 15,
false );
                // Set loopback as compressed
                camera.setLoopback( true );

                video.attachCamera( camera );
                ns.attachCamera( camera );
                microphone = Microphone.getMi-
crophone();
                ns.attachAudio( microphone );
                ns.publish( userItem.identifier
);
            } else {
                video.attachNetStream( ns
);
                ns.play( userItem.identi-
fier );
            }
            videoDisplay.video = video;

        }
    ]]>
</mx:Script>
<VideoContainer
    id="videoDisplay"
    width="180" height="180" />

```

```
</mx:Panel>
```

Listing 4

```

package
{
    import mx.core.UIComponent;
    import flash.events.Event;
    import flash.media.Video;

    public class VideoContainer extends
    UIComponent
    {
        private var _video:Video;

        public function VideoContainer()
        {
            super();
            addEventListener(Event.RESIZE,
resizeHandler);
        }

        public function set
video(video:Video):void
        {
            if (_video != null)
            {
                removeChild(_video);
            }

            _video = video;

            if (_video != null)
            {
                _video.width = width;
                _video.height =
height;
                addChild(_video);
            }
        }

        private function
resizeHandler(event:Event):void
        {
            if (_video != null)
            {
                _video.width = width;
                _video.height =
height;
            }
        }
    }
}

```

Renaun Erickson is a RIA developer specializing in Flex, ColdFusion, and PHP, and he is a Flex Adobe Community Expert. He is active in the community through <http://renaun.com/blog/>, as well as the local Las Vegas Adobe User Group <http://vegasaug.org>.

Adding Flash Video to PowerPoint Presentations

It's worth the effort
by Kim Cavanaugh

PowerPoint has been capable of accepting and playing back videos for quite some time, at least as far back as Office 97.

In most cases the process is quite simple. Go to the Insert menu, click on Insert Movies, and away you go. At least in theory.

The reality of the situation is that your success with video in PowerPoint may vary wildly. Sometimes things will play back absolutely perfectly, especially if you use the same computer with the same operating system to create and encode the video, create the PowerPoint file, and play back the slide show. When a single machine is used, your results are generally pretty good, although there may be times when the audio and video lose their synchronization. For the most part, as long as you stick with a single machine, video in PowerPoint works just fine.

But what happens when you develop your video and PowerPoint show on different machines? Perhaps you're collaborating with someone else who is producing the video while you create the slide show. It's possible that they may use a video codec that is incompatible with your machine, such as when a newer version of QuickTime is used for the video, but an older version of PowerPoint is the only one you have available. Or perhaps your presentation looks great on your computer, but when you attempt to take the show on the road and use a different

machine, you find that your video fails to play back correctly. Maybe you need to share your completed presentation with a co-worker so they can also use the PowerPoint for presentations. In many of those types of situations, video in PowerPoint can be problematic.

Luckily, there is a solution that works between versions of PowerPoint that eliminates the unknowns and ensures that your video will play back exactly the way you want. By converting your video to Flash Video and inserting the completed file into your PowerPoint show, you can be certain that the video will play back regardless of the computer and even the operating system that is being used.

In this tutorial you will learn:

- How to set up your project for the best results
- Options for encoding the video for playback
- How to insert Flash files into PowerPoint

Preparing Your Project

When you are working with a project such as this, there is a very simple and time-honored principle to keep in mind – K.I.S.S. or Keep It Simple, Stupid. (At least that's what I was taught the acronym meant.)

When you are working with multiple files, it is almost always best to work entirely within one folder. Your PowerPoint file as well as the Flash video

and Flash files you need for playback should all be saved into a single folder. There are several advantages to this method:

- By keeping your files in one location, the task of writing the path to the Flash files is greatly simplified. As you'll see, you need to know the path and file name to the Flash file if you want your movie to play back correctly. With all of the files in one location, you eliminate the issue of having to remember how to write the path. Instead, you'll only need to write the file name when the time comes to insert your Flash movie into PowerPoint.
- Moving the project to another computer or to removable media becomes much simpler when all the files are in one location. If you need to show your presentation at a location other than your workplace, you can simply copy the entire folder onto a flash drive or CD and take the show with you. Or, if you need to share the presentation with someone else, you can zip the file up and send it to them by e-mail. Either way, having all your files in one location makes them much more portable and easy to work with.
- If you are using Flash 8 and the video components for producing your video playback movie, you'll need to move the playback SWF that is created along with the movie that holds your video. Once again, having all these files in a single folder greatly simplifies this

Article courtesy of
Community MX. For
more quality articles such
as this one, go to
www.communitymx.com

process and eliminates errors when playing back your file.

Cross-Platform Compatibility

Sadly, the latest Macintosh version of PowerPoint does not play well with Flash files that contain videos in the FLV format as of this writing. However, Flash Video is still a great way to incorporate your QuickTime files that may not play properly in the Windows environment. You can do your editing in iMovie and share the files at the size of your choosing, convert the files to FLV, then incorporate them easily for playback in Windows. For video files that need to play on a Macintosh computer your best bet will be to stay with the QuickTime format and insert the movies directly into your presentation.

Options for FLV Conversion

If you have Flash 8 Professional, you already own the best tool for converting video to the FLV format. Using the Flash Video Encoder that ships with Flash 8 Pro you can convert your video easily, then use the excellent FLV Playback component to build your SWF file for insertion into PowerPoint. This is the most full-featured option as you can choose from the different playback controllers that ship with Flash 8 Pro to add a sweet little controller that will allow you to start, stop, and rewind your video directly from within PowerPoint.

If you don't own Flash 8 Pro, you'll need another method for converting your videos to the FLV format. My personal recommendation is Sorenson Squeeze (http://www.sorensonmedia.com/solutions/prod/comp_win.php), which allows you to create either an FLV file for use in building a Flash project, or, for even faster conversion, you can create the complete SWF directly in Squeeze with the FLV file embedded in the completed file. The only drawback to this method is that there will be no controller available when the video plays within your PowerPoint presentation. You will still be able to use simple controls for playing the video, but you will need to do that by right-clicking on the video and choosing the options presented in the context menu that appears.

Preparing Your Video in Flash 8 Professional

For the sake of the remainder of this tutorial, let's assume that you will be using Flash 8 Professional to prepare your video files for playback and that you have already converted your movie into the FLV format. For more on creating video playback files in Flash 8, see Tom Green's Creating a Video Player in Flash Professional 8 (<http://www.communitymx.com/abstract.cfm?cid=12EBE>).

There are a few things to keep in mind when creating your movie file in Flash if you plan to incorporate them into a PowerPoint presentation.

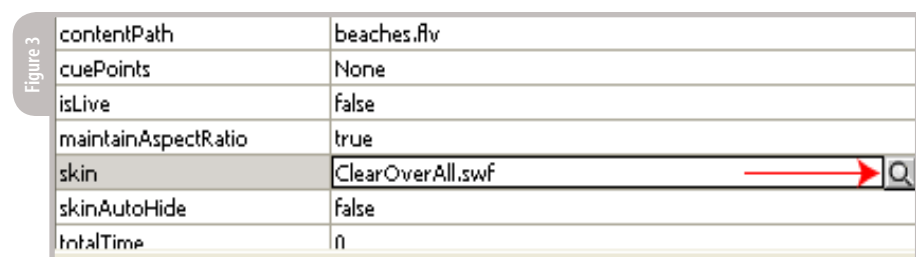
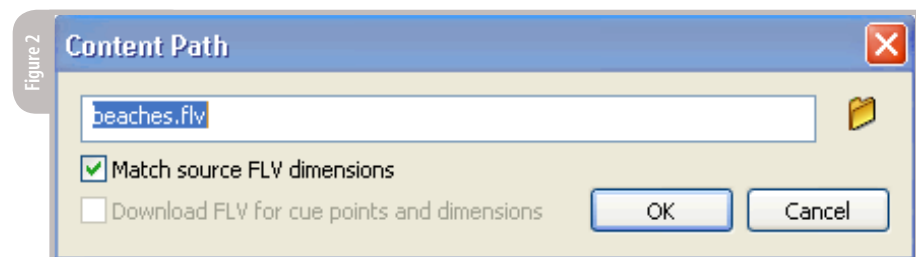
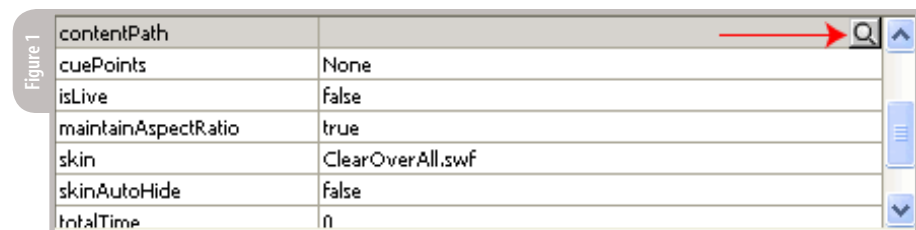
First, when choosing the playback controller you should select one of the "Over" style skins rather than an external controller. While the controller will work just fine inside the presentation, the difficulty comes when you need to scale the movie to the proper size. PowerPoint can present a challenge when it comes to scaling your movie to the correct size as it seems to have a mind of its own for creating and maintaining an aspect to the resized movie. In practical terms, this can lead to a gray box appearing around the movie when you attempt to resize the SWF file. Just as both-
ersome, the controller itself can be cut off

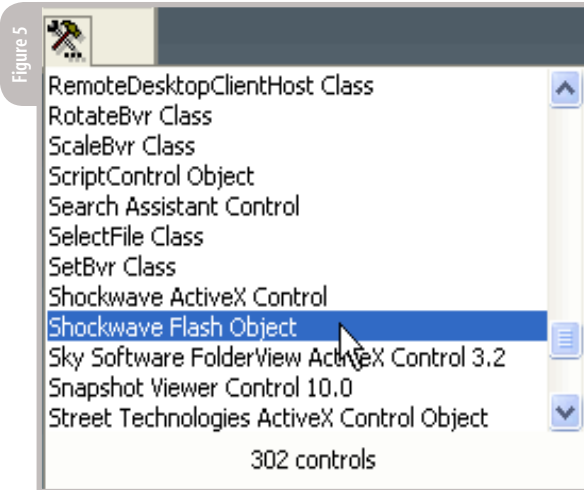
when the movie is resized as well, leading to lots of wasted time when you want to get the movie object sized to one that gives the best quality during playback. The best method to avoid this is to simply put the controller on top of the video.

Second, the computer that you will be presenting on must have Flash Player 8 installed in order for the video to play back correctly. This is not a big problem in most cases. If you simply go to the Flash player download page (http://www.macromedia.com/shockwave/download/download.cgi?P1_Prod_Version=ShockwaveFlash) using Internet Explorer, the latest version of the Flash Player will be downloaded to the machine in, well, a flash. Just be certain that you allow yourself sufficient time if your presentation will be given away from your normal workplace to check for Flash Player 8 and install the player if needed.

With all of that said, the actual creation of a video playback file in Flash 8 couldn't be easier. Let's run briefly through the steps.

1. Create a new Flash document. Save the file in the same folder as your PowerPoint presentation.
2. Open the Components panel by choosing Window > Components.
3. Open the FLV Playback – Player 8 folder in the Components panel and

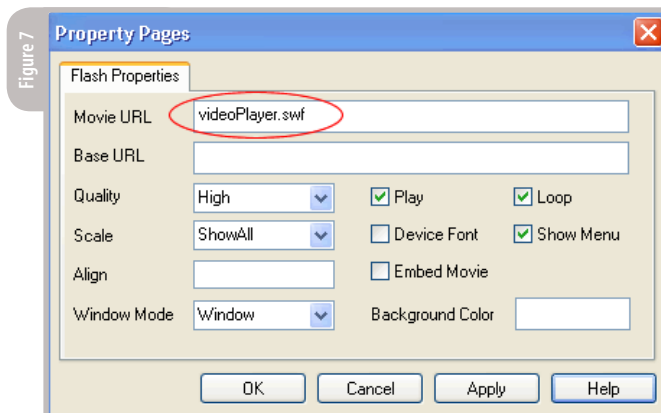
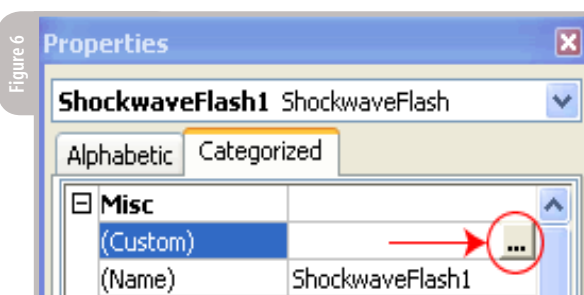




drag an instance of the FLVPlayback component onto the stage.

4. Name the component MyPlayer in the Properties inspector.
5. Click on the Parameters tab in the Properties inspector.
6. Click on the Browse button in the contentPath category of the parameters listing as you see in Figure 1.
7. In the Content Path dialog box, enter the name of your FLV file. You can also click the Browse button to browse to your file location. (Remember that it is recommended that the FLV file be located in the
8. Be sure to check the Match source FLV dimensions checkbox as you see in Figure 2.
9. Click on the Browse button in the Skin category of the parameters listing as you see in Figure 3.
10. Use the dropdown menu in the Select Skin dialog box to choose one of the available playback controller skins. Remember that due to the particular way that PowerPoint sizes and resizes Flash movies, one of the "Over" styles is recommended.

same folder as the PowerPoint file for ease of use.)



11. Click Modify > Document.
12. Click on the Match Contents radio button to have Flash automatically size the movie correctly.
13. Click File > Publish. Your movie is now ready to insert into your PowerPoint presentation.

When you publish your movie in Flash 8 you will create two files that are needed for the playback of the completed movie. The SWF file that will have the same name as the source FLA file and a SWF file that matches the name of the playback skin. Both of these files are necessary and should be located in the folder containing your project files.

Inserting a Flash File into PowerPoint

Now that your Flash movie is ready, it's time to insert the Flash file into your PowerPoint presentation. This involves a toolbar that most people don't use very often, if at all, but other than turning that toolbar on, there isn't anything in this process that is difficult. Let's see how it's done.

1. Save your PowerPoint presentation in the same folder where your Flash files are located.
2. Go to the slide where you want the video to appear.
3. Click on View > Toolbars > Control Toolbox.
4. Click on the More controls button that you see circled in Figure 4.
5. From the menu that appears, select Flash Shockwave Object as you see in Figure 5.
6. Your cursor will now change to a crosshair symbol. Drag our rectangle onto the slide that is approximately the size of your Flash movie. You'll resize the object once the movie is loaded.
7. Right-click the white box that represents your Flash object on the slide and select Properties from the context menu. The Properties dialog box will open.
8. Click on the button next to the Custom field at the top of the dialog box as you see in Figure 6.
9. In the dialog box that appears, you will insert the name of the SWF file that contains your movie. Once again, this is another place where the wis-

dom of keeping all the files in one location is seen as there is no Browse button or automatic method of setting the path to the file. In order to insert the Flash movie file, simply place the name of the SWF file that you published from Flash in the Movie URL field as you see in Figure 7.

Figure 7 shows the default settings that appear when a Flash movie is inserted into PowerPoint. Notice the Play and Loop checkboxes are selected, which will autoplay the movie when the slide is shown, and will loop the movie back to the beginning when it is finished – but only if no controller is present. You will probably want to experiment with these settings to see which works best for you. Note that you can choose to embed the movie into the PowerPoint file, but if you are using a playback controller with Flash 8 video, then the controller SWF file must still be present in the same folder as the presentation.

10. Click OK.
11. Test your slide show by choosing Slide Show > View Show. Your movie will load and play when the slide opens. When you end the show you will see the last frame where the video stopped inside the PowerPoint slide.
12. It's likely that you'll need to resize the Flash object on the stage to get the best quality from your video. You can do that by clicking once on the Flash object on the slide and using the resize handles to change the size of your video as you see in Figure 8. Your presentation with a slick little video included is now ready to go.

One thing to keep in mind is that PowerPoint will remember the movie's position every time you play your slide-show. To ensure the video plays from the beginning, be sure to rewind and save your file before you make that big presentation.

Conclusion

As you have seen here, inserting Flash files containing Flash video is fairly easy, and definitely worth the effort if you want to ensure consistent playback of your video, no matter where you show it. By inserting a Flash movie with


an embedded video file inside, you can be certain that whether you need to move the presentation onto a Flash drive or CD to show at another location, your movie will open with its audio intact and play back perfectly. Not only that, but you'll also have the nice controls that are available in Flash 8 just in case the people watching your show are so impressed that they want to see your video over and over again. 

Figure 8



“Inserting Flash files containing Flash video is fairly easy, and definitely worth the effort if you want to ensure consistent playback of your video”

Beyond 'Request-Response' Modes of Web Services

Using Flash and Flex
by Mansour Raad & Andy Gup

geographic Information Systems (GIS) services today are going mainstream. Corporate executives and individuals at home are demanding access to many of the same GIS capabilities previously used only by engineers, geologists, and government officials.

ESRI's ArcWeb Services are at the forefront of expanding demand for interactive GIS applications. Using subscription-based Web mapping APIs, developers can integrate mapping, GIS functionality, and content into applications delivered online or on desktops, handheld computers, and other devices. Every day, ArcWeb Services serves more than 6 million mapping requests, helping companies manage commercial fleets, analyze demographic data to spot trends, map locations, and execute other requests.

As the reach of GIS expands, new approaches are needed to develop Web services that are more intuitive, interactive, and visually engaging. Typically, ArcWeb Services developers program in variety of programming languages including Java, .NET, and PHP to build request-response infrastructures for presenting increasingly complex data sets that mix archived information with real-time data feeds from ArcWeb Services. The applications generally wait for some input from the end user, and, once a request is completed, the application waits for the next input.

To address new demands for GIS

capabilities, the ArcWeb Services team took a closer look at Adobe's Macromedia Flash and then evaluated Adobe Flex. Flex addressed several needs. First, it lets ArcWeb Services deliver dynamic content as SWF files, providing flexibility in the type and richness of geographic information available to customers. This content is delivered through ArcWeb Services Explorer, which is currently based on Flex 1.5 and will be migrated to Flex 2 in the next release.

With ArcWeb Services Explorer, developers can now build Rich Internet Applications (RIAs) that provide geographic functionality and content in a desktop-like environment. The capabilities are encapsulated within the ArcWeb Services JavaScript API, which allows developers to embed ArcWeb Services Explorer into a Web application with just a few lines of code.

In addition, Flex provides the ability for applications to take advantage of vector graphics, which are mathematical representations of images that contain points, lines, curves, and polygons. Vector maps are vector graphics that contain information related to individual geographic features within the map image, such as roads, lakes, national parks, overpasses, and elevation data. The applications can now interact with the individual features. For example, features can be highlighted or turned on and off, objects can be dynamically moved around, and even sound and movies can be embed-

ded. The vector mapping capabilities are made possible by Flex, and they are the key to providing ArcWeb Services customers with desktop-like functionality.

In comparison, traditional raster map images, such as bitmaps (.bmp) and GIFs, are static and are designed to provide only a description of the colors within the image. Raster images were designed to be rendered, rather than to contain or enable functionality, and are the format of choice for delivering mapping applications that require visual precision, as in satellite imagery.

With vector-based mapping functionality, ArcWeb Services developers can now rapidly build and deploy Flash-based client applications that directly interact with individual features, as well as multiple layers of data. For example, a street map can be used as the base layer and a zip code boundary shape file can be overlaid on top of that. Then, demographic information can be overlaid as different colored, semi-transparent shadings. This application can be built using just the ArcWeb Services JavaScript API.

The rich Internet mapping applications our customers are developing today have dramatically changed how people access and interact with content. For instance, ArcWeb Services can be used to enhance a real-estate Web site. Rather than limiting users to searching by zip codes or street addresses, clients can draw their own boundaries on screen. The real-estate application then automat-

Mansour Raad is senior software architect, ArcWeb Services, ESRI.

Andy Gup works in product marketing, ArcWeb Services, ESRI.


ically searches for properties that meet their criteria – price, size, specific features – within their drawn boundaries. Search results are presented as rich content, including video and interactive maps.

Other clients are finding new ways to engage with once-static online maps. Mapping services that simply showed people how to get from point A to point B are being replaced by more dynamic services that combine directions and images with real-time traffic and weather information. For instance, if the directions drivers received an hour ago are no longer optimal because of changing

traffic conditions, the interactive system automatically pushes out new information, alerting drivers that route changes are needed.

A primary reason ESRI can deliver these enhanced services is that Flex gives ArcWeb Services more control on both the server and client sides. The ArcWeb Services Flash client provides a rich component library that doesn't require developers to spend a lot of time creating trees, tables, and canvases. With Flex, we have skinning, piling, and the ability to easily integrate other SWF components into ArcWeb Services' client interfaces.

ArcWeb Services customers can now develop rich applications in hours that previously could have taken weeks to develop.

Based on our development experience and new client expectations, it is clear that the nature of Web services is changing. For many of ESRI's customers, simply receiving and viewing content is not enough. Instead, end users want real-time access to dynamic mapping content, trusting that the data and services they require will be available even before they ask for them. 

Efficient Web Content Management

CommonSpot™ Content Server is the ColdFusion developer's leading choice for efficient Web content management.

Our rich Web publishing framework empowers developers with out-of-the-box features such as template-driven pages, custom content objects, granular security, and powerful ColdFusion integration hooks (just to name a few), allowing you to rapidly build and efficiently deploy dynamic, high performance Web sites.

CommonSpot's open architecture and extensive APIs enable easy customization and integration of external applications. With CommonSpot, you can design and build exactly what you need. Best of all, CommonSpot puts content management in the hands of content owners. With non-technical users responsible for creating and managing Web content, developers are freed to focus on strategic application development.

Evaluate CommonSpot today.

To see your site *running* under CommonSpot, call us at 1.800.940.3087.



fast
easy
affordable

features.

- 100 % browser-based
- Content object architecture
- Template driven pages
- 50+ standard elements
- Content reuse
- Content scheduling
- Personalization
- Flexible workflow
- Granular security
- Mac-based authoring
- 508 compliance
- Full CSS support
- Custom metadata
- Taxonomy module
- Extensible via ColdFusion
- Web Services content API
- Custom authentication
- Replication
- Static site generation
- Multilanguage support

1.800.940.3087
www.paperthin.com

Paper | Thin

© Copyright 2005 PaperThin, Inc. All rights reserved.



Back and Forth

Custom history management with Flex 2

by Rob Rusher

For over a decade now, we have been trained to use the “back” and “forward” buttons in our Web browser to review or back-track previously viewed content. We are trained to the point that there are even keyboard shortcuts. Unfortunately, this can be a problem when browsing Flex applications. Being the well-trained users that we are, we often forget that we could cause the Flex application to reload unwittingly.

Fortunately, there is a solution that is built into the Flex SDK and HTML templates that are generated in a Flex Builder project. By default, Flex enables history management for only a couple of navigator containers, without using any additional ActionScript or MXML tags. Although this is a good start, we often need more options to build truly custom applications.

In order to implement your own custom history management, you will need to use the HistoryManager class that is part of the Flex 2 SDK. By implementing the API, you can create custom history management for your custom objects in an application. Note: History management is not supported on Netscape 4.x and Opera 6.0 web browsers.

History Enabled Components

History management is available by default for the Accordion and TabNavigator navigator containers. It is disabled by default for the ViewStack

navigator container. When history management is enabled, as the user navigates different navigator containers within an application, each navigation state is saved. Selecting the Web browser’s “back” or “forward” browser button displays the previous or next navigation state that was saved. History management keeps track of where you are in an application, but it is not an undo and redo feature that remembers what you have done. Note: When history management is enabled for a particular component, such as a navigator container, only the state of the navigator container is saved. The state of any of the navigator container’s child components is not saved unless history management is specifically enabled for that component.

Custom History Management

The mx.managers.HistoryManager package is pretty useful for building generic undo mechanisms. It must be imported before using it. The HistoryManager provides a register () method that allows you to register any object with it, even if it does not have a visual element. To register a component with the HistoryManager class, that component must implement the IHistoryManagerClient interface, and then it must be registered with the HistoryManager class. You will then pass along a reference to a component instance to manage. Each registered component receives a

unique state ID based on its full path-name.

In the following example, the Application component (this) is registered with the HistoryManager class when the Application is initialized:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" initialize="mx.managers.HistoryManager.register(this);" implements="mx.managers.IHistoryManagerClient">
```

By implementing the IHistoryManagerClient interface, you “agree” to implement the following two methods into your object’s code:

- function saveState(): Object
- function loadState(state : Object): void

The HistoryManager calls the saveState () method on all registered objects every time the state of the application changes. It’s up to you to return any object that describes the current state of your object (like a snapshot). If this state is going to be restored, then all registered objects receive their state back via the loadState () method. Again, it’s up to you to rebuild the state by investigating the snapshot.

You can also inform the HistoryManager to save the current state of the whole application by calling its static save() method. This has to be done by hand if a performed action has


Rob Rusher is a solutions architect with Effective UI, a technology design and development agency that excels in the strategy, design, and development of interactive Rich Internet Applications and Web 2.0 solutions. He has been developing internet-based applications since 1995. rob.rusher@effectiveui.com

to result in building a “snapshot” of your application. I’ve illustrated (see Listings 1 and 2) two ways to implement the HistoryManager:

How the HistoryManager Class Saves and Loads State

The history management feature uses the Macromedia Flash `navigateToURL()` function to load an invisible HTML frame in the current Web browser window. It then encodes a Flex application’s navigation states into the invisible frame’s URL query parameters (using the standard `prop1=value1&prop2=value2` format). A SWF file, called `history.swf`, in the invisible frame decodes the query parameters and sends the navigation state back to the HistoryManager class. You can investigate this by viewing the HTML source that is generated along with a Flex application.

Conclusion

Once again, the Flex SDK has made it possible for extending components to be history management enabled very easy. By simply implementing an interface and utilizing the HistoryManager class, you are able to quickly avoid many Pavlovian nightmares from the back button. Keep in mind that this feature should be used primarily for managing navigation and not for undo/redo. Although you could accomplish some level of undo/redo if you are creative, there are definitely better ways of doing it, so it’s best to stick to navigation with the HistoryManager. I hope you found these examples easy to follow and helpful in your learning process. 

Listing 1

```
HistoryWindow.mxml
<?xml version="1.0" encoding="utf-8"?>
<mx:TitleWindow xmlns:mx="http://www.adobe.com/2006/mxml"
    implements="mx.managers.
IHistoryManagerClient"
    creationComplete="initPanel()"
    move="doMove()"
    mouseUp="doMouseUp()"
    width="200" height="150"
    backgroundColor="#FFFFFF"
    cornerRadius="8"
    horizontalAlign="center">

<mx:Script>
<![CDATA[
    import mx.effects.Move;
    import mx.managers.HistoryManager;

    private var changed : Boolean;

    public function initPanel() : void
    {
        x = 0;
        y = 0;
        HistoryManager.register( this );
        storeInternalState();
    }

    public function saveState() :
Object
    {
        // called by HistoryManager,
        tells the component
        // to create a “state” object
        and to return it
```

```
        return { xPos:x, yPos:y };
    }

    public function loadState( state :
Object ) : void
    {
        // called by HistoryManager,
        passes in a state
        // object so the component can
        rebuild it’s state
        if( state != null )
        {
            restoreInternalState( state );
        }
    }

    private function doMove() : void
    {
        changed = true;
    }

    private function doMouseUp() :
void
    {
        if ( changed ) storeInternal-
State();
    }

    private function storeInternal-
State() : void
    {
        HistoryManager.save();
        changed = false;
    }

    private function restoreInternal-
State( state : Object ) : void
```


“Once again, the Flex SDK has made it possible for extending components to be history management enabled very easy”

```
{
    var mover : Move = new Move( this );
    mover.xTo = state.xPos;
    mover.yTo = state.yPos;
    mover.play();
}

]]>
</mx:Script>

<mx:Text text="Just move me around, use the
browser's buttons to restore state."
selectable="false"
width="100%"
fontSize="12"/>

</mx:TitleWindow>
Main.mxml
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.
com/2006/mxml"
    initialize="initApp()">

<mx:Script>
<![CDATA[
    import mx.managers.PopUpManager;

    public function initApp() : void
    {
        PopUpManager.createPopUp( this,
HistoryPanel, false );
        PopUpManager.createPopUp( this,
HistoryPanel, false );
    }
]]>
</mx:Script>

</mx:Application>
```

Listing 2

```
StockQuote.mxml
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.
com/2006/mxml"
    implements="mx.managers.
IHistoryManagerClient"
    initialize="mx.managers.HistoryManager.
register( this );">
```

```
<mx:Script>
<![CDATA[
    import mx.managers.HistoryManager;

    public function saveState() : Object
    {
        var state : Object = new Object();
        state.symbol = symbol.text;
        return state;
    }

    public function loadState( state :
Object ) : void
    {
        if (state != null && state.symbol !=
null ){
            symbol.text = state.symbol;
            ws.getQuote.send();
        }
    }

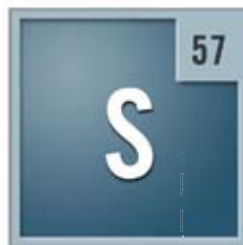
    public function getQuote() : void
    {
        ws.getQuote.send();
        HistoryManager.save();
    }
]]>
</mx:Script>

<mx:WebService id="ws"
    wsdl="http://services.xmethods.net/soap/
urn:xmethods-delayed-quotes.wsdl"
    useProxy="false">
    <mx:operation name="getQuote">
        <mx:request>
            <symbol>{ symbol.text }</symbol>
        </mx:request>
    </mx:operation>
</mx:WebService>

<mx:Label text="Enter a symbol:"/>

<mx:HBox>
    <mx:TextInput id="symbol" width="60"/>
    <mx:Button label="Get Quote"
click="getQuote()" />
</mx:HBox>

<mx:Label text="{ ws.getQuote.lastResult }"
fontSize="24"
fontWeight="bold"/>
```



STREAM57



Customized Flash-based media solutions

Software and services for collaboration, video conferencing, and e-learning

visit us
stream57.com

e-mail us
streamline@stream57.com

call us
212.909.2550 x1012

Flex On Ruby on Rails

Automating the communication between the client and server

by Harris Reynolds

With the release of Flex 2, Adobe has introduced a tool that makes building rich user interfaces for the Web easier than ever; of course, as anyone remotely plugged in to the Web development community knows, Ruby on Rails has made creating database-driven Web applications brain dead simple. This article discusses a technology that marries Flex with RoR applications by providing a means of automating the communication between the client and server. The technology is "WebORB for Rails," a free and open source (GPL) server made available by Midnight Coders (www.themidnightcoders.com).

Harris Reynolds has been working in software development and systems integration for the past eight years. Most recently he was an enterprise architect at Fannie Mae where he was responsible for the development of messaging architecture supporting internal integration. Prior to consulting for Fannie Mae he was a lead engineer for webMethods working on SOA infrastructure and their underlying service registry. He has also completed projects for several large companies including Amkor Technology, BASF, PricewaterhouseCoopers, and Bank of America.

Many of the tutorials on the Web that talk about integrating Flex clients with Rails discuss using raw XML over HTTP. While this mechanism is nice for the standard "mom and apple pie" tutorial that return a simple list of records, this level of integration quickly becomes unwieldy as the complexity of the application grows. With each new operation on the server, a developer must spend valuable cycles serializing and deserializing requests and responses before they can focus on the actual business logic.

WebORB relieves the burden of this "serialization tax" by supporting the concept of remote objects whereby Flex clients can natively invoke methods on the server and retrieve the responses all within ActionScript, the object-oriented programming language used by Flex.

This article will discuss installing WebORB to "Flex-enable" a Ruby on Rails application, the steps required to create an application using Flex + WebORB + RoR, and some advanced topics like dealing with Rails models, etc.

Installing WebORB for Rails

WebORB runs as a plug-in for Rails and is installed by simply running the following command within the root directory of a Rails application. On *nix systems:

```
./script/plugin install http://themidnightcoders.net:8089/svn/weborb/
```

or on Windows:

```
ruby script/plugin install
http://themidnightcoders.net:8089/svn/
weborb/
```

Writing an Application Using WebORB

There are three simple steps that are required to build a Flex-based RIA utilizing WebORB and Rails.

1. Build the service class and drop into RAILS_APP/app/services
2. Add an entry to RAILS_APP/config/WEB-INF/flex/remoting-config.xml for the remote service
3. Set up a RemoteObject on the client-side using Flex that will communicate directly with the back-end service.

As a simple example consider the following service written in Ruby:

```
require 'weborb/context'
require 'rbconfig'
class InfoService
  def getComputerInfo( requestId )
    computer_info = Hash.new
    request = RequestContext.get_request
    computer_info['serverName'] = request.
server_software
    computer_info['requestId'] = requestId
    computer_info['os'] = Config::
CONFIG["arch"].to_s
    computer_info['currentTime'] = Time.
now
    computer_info
  end
end
```

This example is bundled within the WebORB on Rails product and simply returns a bag of properties related to the server running the service. Generally, the classes that are surfaced as remote objects are added to a /services directory under the RAILS_APP/app directory.

After creating this class, you must also tell Flex that this class needs to be available as a remote object. This is configured by adding the following snippet of XML to the remoting-config.xml file located within the RAILS_APP/config/WEB-INF/flex directory:

```
<destination id="InfoService">
<properties>
<source>InfoService</source>
</properties>
</destination>
```

Last, this service is invoked by a client by using the RemoteObject class within ActionScript. Below is a small chunk of code that demonstrates how to use this class:

```
remoteObject = new RemoteObject();
remoteObject.destination =
    "InfoService";
remoteObject.getComputerInfo.addEventL
istener("result", onResult);
remoteObject.addEventListener("fault",
onFault);
```

Assuming that the remote object variable was declared at the top of your MXML application, this code illustrates creating and initializing a remote object for use. Note in particular that the destination property is set to "InfoService", which maps directly to the ID attribute of the destination we just configured within remotings-config.xml. Note that this is how Flex (and WebORB) determines where to send this request. The final two lines above demonstrate adding callback methods to the remote object; the first line tells Flex to pass the result of a call to the getComputerInfo method (this maps to the Rails service method on the server) to the onResult method (which is on the client). The last line tells the remoteObject instance to pass any faults to the onFault method (also implemented in ActionScript). The example below shows the two event listeners that were registered above:

```
public function onFault(event:
FaultEvent):void
{
    Alert.show(event.fault.faultString,
    'Error');
}
public function onResult(event:
ResultEvent):void
{
    var computerInfo:Object = event.
```

```
result;
serverInfoText.text = computerInfo.
serverName;
requestIdText.text = computerInfo.
requestId;
osText.text = computerInfo.os;
timeText.text = computerInfo.current-
Time.toString();
invokeButton.enabled = true;
}
```

Now the remote object can invoke the method on the server written in Ruby by simply calling:

```
remoteObject.getComputerInfo("ABC123
");
```

and the results will be available in the event.result object that is passed to the onResult method as demonstrated above. In this example each property that is stored on the server is available as a property of the event.result object that we store above in the computerInfo instance. The source code for this example is available as part of the WebORB distribution for those interested in looking at the complete picture; after installing WebORB the server code is available in /app/services/InfoService.rb and the client code is available in /public/examples/example.mxml.

Advanced Features of WebORB Active Record Integration

The previous example is a simple tutorial on how integrating Flex and Rails works using WebORB and the RemoteObject class. However, when building Rails applications, most of the interesting data for your application will be available via the ActiveRecord API in the data model layer. This is actually where WebORB provides a great deal of value: it natively serializes Rails model objects to Flex clients including handling any associations that are requested as part of an active record query.

Consider the following Ruby class that can be dropped into a ProductService.rb file within /app/services:

```
class ProductService
  def getProducts
    Product.find(:all, :include => [ :
    images ])
  end
end
```

This service will return all the products in the database and include associated image records; this type of service would be great for displaying a product list within a DataGrid that also could display product images upon selecting a row in the grid. On the client side the results of this method invocation could be made available in an ActionScript method such as:

```
public function onGetProducts(event:
ResultEvent):void
{
    var products:Array = event.result as
    Array;
    dataGrid.dataProvider = products;
}
```

Also note that within each product, another array is available that is accessed through the images property (the Image model in rails would likely include a URL or path attribute that Flex could use to load pictures). As an example, the following ActionScript method could be attached to the "change" event of the DataGrid:

```
public function onProductSelected():
void
{
    var product:Object = dataGrid.select-
    edItem;
    for( var i:Number = 0; i < product.
    images.length; i++ )
    {
        var imageObject:Object = product.
```


**“Another
useful feature
of WebORB
is the ability
to access
the HTTP
request or
session objects
within remote
Rails services”**

```
images[i];  
// do something useful with imageOb-  
ject.path attribute  
}  
}
```

Last, remember that if you do not want to include associated data (has_many/belongs_to relationships within Rails models), then simply do not pass the :include option to your query.

Access to the HTTP Request and Session

As a final note, another useful feature of WebORB is the ability to access the HTTP request or session objects within remote Rails services. These objects are available to standard Rails controllers that operate at the HTTP level; however, when using WebORB, developers simply develop “services” that are server methods that generally act as a layer between clients and Active Record models without inherent knowledge of the HTTP transport protocol. There will be times, however, when a service will need access to either the HTTP request or session. This is made available through a RequestContext class that can be included in any remote service. This technique was actually used in our first example. Note the “require ‘weborb/context’” line within InfoService.rb listed above; this line imports the RequestContext class so that developers can access the HTTP request and session via the respective RequestContext.get_request and RequestContext.get_session class-level methods. Any data that is stored in the session will be available to subsequent requests using the standards Rails session API after retrieving the session object similar to:

```
session = RequestContext.get_session  
#do something interesting with the  
session object
```

Basic Authentication Support

WebORB provides support for basic authentication of client applications. This allows credentials to be set on the RemoteObject instance, and services

to access the associated user name and password on the server. On the client side this is done with ActionScript using the setCredentials method on a RemoteObject instance.


```
remoteObject.setCredentials("UserNameH  
ere", "PasswordHere");
```

This information is available using the same RequestContext API discussed above. Within your service you can get the credentials using the API demonstrated below:

```
require 'weborb/context'  
class SecureService  
  def doSomethingSecretive  
    user = RequestContext.get_user_name  
    password = RequestContext.get_password  
    //Authenticate credentials here, if  
    unauthorized raise error  
  end  
end
```

The one problem with this approach is that now the service code is contaminated with authentication code. Ideally this would happen before ever reaching the method (like using a “before filter” in Rails controllers. WebORB will eventually provide an interface that will authenticate users in a similar way before ever reaching secure methods. In the meantime, it is important to make this feature available for developers who need this capability today (WebORB on Rails will continue to support this API even after a cleaner approach is provided).

Conclusion

This article provided an initial glimpse into how Flex clients can be integrated with Ruby on Rails using WebORB’s Flex RPC implementation and the RemoteObject API. While some applications with very simple data requirements can happily integrate Flex and Rails using model.to_xml and standard HTTP requests, WebORB on Rails simplifies the development process as the amount of client/server interaction scales to a large number of request/response types. 

Rich Internet Applications: AJAX, Flash, Web 2.0 and Beyond...

REGISTER TODAY AND SAVE!

www.AjaxWorldExpo.com

AJAXWORLD™ EAST

CONFERENCE & EXPO



NEW YORK CITY

THE ROOSEVELT HOTEL LOCATED AT MADISON & 45th

**SYS-CON Events is proud to announce the
AjaxWorld East Conference 2007!**

The world-beating Conference program will provide developers and IT managers alike with comprehensive information and insight into the biggest paradigm shift in website design, development, and deployment since the invention of the World Wide Web itself a decade ago.

The terms on everyone's lips this year include "AJAX," "Web 2.0" and "Rich Internet Applications." All of these themes play an integral role at AjaxWorld. So, anyone involved with business-critical web applications that recognize the importance of the user experience needs to attend this unique, timely conference – especially the web designers and developers building those experiences, and those who manage them.

BEING HELD APRIL 2-3 2007!

We are interested in receiving original speaking proposals for this event from i-Technology professionals. Speakers will be chosen from the co-existing worlds of both commercial software and open source. Delegates will be interested in learning about a wide range of RIA topics that can help them achieve business value.

Working with Large Applications

Static versus dynamic linking: development perspective

Part 1

by Yakov Fain, Victor Rasputnis,
& Anatole Tartakovsky

In this excerpt from our book, *Rich Internet Applications*, we'll cover how to set up large applications intended for Web or, more broadly speaking, distributed deployment. As an example let's consider an enterprise application that consists of hundreds of screens, reports, forms, and dashboards. Accordingly, about a dozen engineers specializing in GUIs, frameworks, data layers, and business domains are working on this application in parallel.

Every application "run" in Flex Builder as well as the invocation of the application's MXML file processed by the Web-tier Flex compiler requires an application build. Needless to say, this takes time. The bigger the application, the more time it takes.

Developers need a fast process of building and deploying their applications.

The application also has to be partitioned for team development both vertically (application screens) and horizontally (UI, skins, reusable components, and back-end code). Imagine working on two "portlets," one of them showing a DataGrid and the other a TreeView. You have a choice: either package a 1MB module with both portlets or download separate ones (500K each) on demand. The latter way has the additional benefit of isolating the work between the team members.

Finally, the model should allow the extensibility of the product, meaning the integration of patches and portlet-style

add-ons as well as external Flex subsystems shouldn't impact the main build.

We'll try to accommodate these requirements emphasizing the productivity of team development and deployment flexibility. But first let's review the deployment scenarios from the business point-of-view in detail.

Deployment Scenarios

Throughout this excerpt we'll use the term patches, which are the fixes and additions made to an application between releases. Add-ons are the parts of the application that are typically added over time. Similar to patches add-ons blend seamlessly into the hosting application, both visually and programmatically.

In some cases the application build may not even "know" about specific add-ons since they're different for each user type. For example, an enterprise application can reveal more screens or more functionality within these screens to internal users. In this case we talk about portlet-style add-ons.

Plug-ins are independent applications that don't share the look-and-feel of the main application. No intensive interaction between the main application and plug-ins is expected.

Application Domains 101

As much as we don't like to duplicate the work done by the Adobe Flex documentation team, we have to cover the

subject of Application Domains since it's essential to this chapter. So, here are the facts.

All code loaded from an SWF file lands in one or another application domain. Conversely, instances of the `flash.system.ApplicationDomain` class stores the tables of the ActionScript 3.0 definitions and class definitions in particular.

A system domain contains all the application domains and there's a current domain where the main application runs. There's also a parent domain for each domain except the system one. System domains, quite naturally, happen to be the parent of the main application's domain.

The definition of the loaded classes as long as they remain loaded can't be overridden down the parental chain. Attempts to reload a class that's already been loaded by its parent will fail. If you're coming from the Java side, you may have seen a similar mechanism called Java `ClassLoader`.

We also have to mention the security domains of the Flash Player, since partitioning the classes (visibility) via application domains is within the confines of the security domains. The choice of a security domain is relevant for use cases when we need to load an SWF file coming from a different server, which is outside the scope of this chapter.

So, an application domain is the mechanism that (a) supports multiple definitions of the same class where chil-

dren can access parent definitions seamlessly or (b) lets child definitions be tentatively merged with parent ones so that accessing the other party's definitions is seamless for both the child and the parent.

This mechanism is enacted by `flash.display.Loader` and `mx.controls.SWFLoader` controls.

The first choice is represented by the syntax `new ApplicationDomain(ApplicationDomain.currentDomain)` while the second one is `ApplicationDomain.currentDomain`.

A specific sub-case of (a) is the use of a system domain as a parent: `new ApplicationDomain(null)`, which results in the ultimate separation of the (class) definitions, eliminating any overshadowing.

Either way, the required application domain is getting assigned the `applicationDomain` property of a `Flash.system.LoaderContext` instance, which, in turn is used as an argument to construct a `flash.system.Loader`, or acts as a property of an `mx.controls.SWFLoader`.

There are nuances in accessing child definitions from the parent as well as in loading/accessing possibly overshadowing the class definitions.

When you bring existing Flex subsystems (perhaps even written in a different version of Flex) under a common application umbrella, it makes sense to resort to a separate application domain. At the same time, if you need to dynamically load `DataGrid` definitions, it makes sense to load them in the same application domain where the main application is running.

Runtime Shared Libraries 101

Flex documentation defines Runtime Shared Libraries (RSL) as "a library of components." We would like to start with the clarification that RSL is not a file but a pattern of using an SWF file from within another SWF file.

Specifically, SWFs marked as RSLs are automatically pre-loaded during the application's bootstrap as opposed to being explicitly loaded by the code you write. To be exact, definitions contained in the SWF are loaded into the `applicationDomain` of the hosting application.

Now how does the application's bootstrap know which SWF files are to be pre-loaded?

Here is an answer. Let's assume that:

- You made the file `FlexLibrary.SWC` (using the `compc` compiler explicitly or from within the Flex Builder's Library Project);
- You've created the file `FlexApplication.mxml`, which refers to components from `FlexLibrary.SWC`;
- While compiling `FlexApplication.mxml` you instructed the `mxmmlc` compiler that `FlexLibrary.SWC` contains an image of an SWF to be pre-loaded during the bootstrap (this will be explained later in this chapter).

Then, the corresponding ActionScript file generated by the `mxmmlc` compiler will have the code fragment shown below. You'll find this and other files in the generated folder of your application project once you set the compiler's option to `keep-generated-actionscript=true`:

```
public class _FlexApplication_
mx_managers_SystemManager extends
mx.managers.SystemManager implements
IFlexModuleFactory {
    public function _FlexApplication_mx_
managers_SystemManager() {
        super();
    }
    override public function info():
Object {
        return {
            "currentDomain": ApplicationDomain.
currentDomain,
            "layout" : "absolute",
            "mainClassName" : "FlexApplication",
            "mixins" : [_FlexApplication_
FlexInit, .....]
        },
        "rsls" : [{url: "FlexLibrary.swf",
size: -1}]
    };
}
```

As a reminder, the `SystemManager` is a parent of all the displayable objects within the application, such as the main window (an instance of `mx.core.Application`), pop-ups, cursors, etc. `SystemManager` also creates the

`mx.preloaders.Preloader` that loads SWF files.

Please note that `FlexLibrary.swf` is not an RSL. As we said above, RSL is a usage pattern rather than a file. What makes `FlexLibrary.swf` part of this pattern is the intent to pre-load it during the application startup communicated by us to the `mxmmlc` compiler.

```
"currentDomain": ApplicationDomain.
currentDomain,
```

This illustrates that the RSL approach results in class definitions from the library are loaded into the same domain where the definition of the application classes belong. That's why, in particular, we find the RSL technique especially useful for delivering various patches, which should be loaded prior to any other class definitions.

SWFs and SWCs: What's Under the Hood

How do our SWC files relate to SWFs? Like every Flex SWC, `FlexLibrary.SWC` contains the `library.swf` and `catalog.xml` files. The latter describes the hierarchy of dependencies found in `library.swf`, which can potentially become `FlexLibrary.swf` (depending on the selected link type described below).

When we compile `FlexApplication.mxml` containing references to `FlexLibrary.SWC` in the library search path, there are three link types to choose from:

- **External** – The `catalog.xml` in the `FlexLibrary.swc` will be used to resolve references; however the code contained in `library.swf` won't be included in the body of the `FlexApplication.swf`. The External link type assumes that by the time `FlexApplication` needs to create instances of classes from the `library.swf` part the definitions for these classes will somehow get loaded in the relevant `applicationDomain`.
- **RSL** – The `catalog.xml` in the `FlexLibrary.swc` will be used to resolve references; code contained in `library.swf` won't be included in the body of the `FlexApplication.swf`. So far sounds like External, right? Here's the difference: all definitions originally contained in the `library.swf` part will be

upfront-loaded into the main applicationDomain during application startup.

- **Merge-in** – Classes that are explicitly referenced by the code (and their dependencies) explicitly get included in the FlexApplication.swf. This is a default option for statically linked applications and guarantees that the definitions of all referenced classes as well as the classes they depend on are loaded into the main applicationDomain outright.

A Merge-in scenario is often called static linking, while External and RSL are cases of dynamic linking.

Suppose we went with dynamic linking via RSL. As illustrated in the previous code fragment, this means pre-loading the FlexLibrary.swf. Here's the question: where do we get this FlexLibrary.swf from? Under one scenario we can let Flex Builder extract and rename the library.swf from the FlexLibrary.swc. In Flex Builder (project Properties > Flex Build Path > Library Path) this option is called Auto extract swf.

Alternatively, we could have declined auto-extracting and unzipped the SWF from the SWC ourselves. As we'll show later, there's yet another way of explicitly controlling the upfront build of FlexLibrary.swf.

We'll illustrate these cases in the next section.

Making the FlexLibrary.swc

Let's make an SWC in Flex Builder by creating a new Flex Library Project. The only component we're going to add to this SWC is the CustomPanel from the following code which enumerates the instances of itself and imprints the number of the instance as part of its title, using the variable instanceNumber that we've declared bindable:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- CustomPanel.mxml -->
<mx:Panel xmlns:mx="http://www.adobe.com/2006/mxml" title="'Custom' Panel
#{instanceNumber}" width="300" height="150"
creationComplete="instanceNumber=++count;" >
<mx:Script>
    public static var count:int;
    [Bindable]
    private var instanceNumber:int;
</mx:Script>
</mx:Panel>
```

To ensure that our CustomPanel is accounted for (in both library.swf and catalog.xml) we have to verify that it's included in the Flex Library

Build Path. Please be aware that every time you add or rename files in your Library Project the corresponding checkbox in Flex Builder gets cleared.

After we click OK, Flex Builder will invoke the compc compiler to create the FlexLibrary.swc in the output bin folder.

Making a FlexApplication Application

Now let's make the application in a separate Flex Builder project. Nothing fancy, we'll just make a static reference to the CustomPanel:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- FlexApplication.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute" xmlns="*">
    <CustomPanel />
</mx:Application>
```

Link our library, compile, and run the application.

To resolve the reference during compile time we had to add our recently created library FlexLibrary.swc to the project's Library Build Path. The default link type is to merge-in the SWC's content.

Merging the contents of SWC results in an optimized, non-overlapping size of the monolithic application. The size of such a self-sufficient FlexApplication.swf is 123KB.

Now let's alter the setting and turn static linking dynamic and setting the Link type to RSL. We'll accept the (default) value of Auto extract.

As a result of the Auto extract the file FlexLibrary.swf will appear adjacent to the FlexApplication.swf. The size of the FlexLibrary.swf will be 236K, nearly as much as the entire FlexLibrary.swc, but the size of the FlexApplication.swf itself will decrease to 40K.

As you can see, Flex attempts static linking of RSL content (Merged into code) by default. This ensures that the smallest size of the resulting monolithic SWF, because it carries only the code that was deemed relevant during the compilation process.

Naturally, the total of 236K + 40K takes two times longer to download than downloading of the statically linked 123K. However, once you have a family of three applications to offer the same user, all approximately the same size and all reusing the same FlexLibrary.swf, it becomes (236 + 3*40) versus 3*123 and you break even.

The download considerations are less relevant on the fast connections and, perhaps, are not relevant at all in scenarios where you can

count on the browser cache to keep SWF files loaded for the next run of the application. A typical example of the latter would be a corporate environment, but then again, some administrators set policies to wipe out the browser's cache on user logoff.

Static versus Dynamic Linking: Development Perspective

While the time of the initial download is an important factor in favor of static linking, we have to consider development productivity as well.

Let's look at enterprise applications. Driven by user requirements, they tend to change frequently. Many enterprise applications tend to be on the larger side, growing in functionality and, accordingly, size, with phased delivery to the users. As an application gets to tens of megabytes, the time required to build the monolithic application becomes a noticeable setback to developer productivity. The problem escalates in a team development where it translates into many hours wasted every day.

And, regardless of size, let's look at the use case of portlet-style add-ons. These are modules that are simply impossible to reference statically in advance. In fact, they are not even supposed to be pre-loaded at all: unlike RSLs they get loaded on demand.

All in all, we need to be able break the application into a set of modules that can be built independently and linked dynamically at runtime.

So, You Say Dynamic Linking?

By now the reader may say, "OK, I got the message, I'll go with RSLs to modularize my development with dynamic linking." Not so fast. Like ancient Achilles, these mighty RSLs have a small weak spot: their well-being depends on static linkage from the main application.

Oh, but doesn't that ruin the hope of dynamic linking? The answer is no, and the explanation is just around the corner in the next section.

First, however, let's create an application to expose the problem. We'll build a FlexApplication2.mxml application. Unlike our previous example, it won't contain static references to CustomPanel. Instead, FlexApplication2 will create instances of a CustomPanel (or any other object for that matter) dynamically, given a name of the class definition as it's done in the function createComponent():

```
<?xml version="1.0" encoding="utf-8"?>
<!-- FlexApplication2 -->
```




Visit the *New*
www.SYS-CON.com
 Website Today!

The World's Leading i-Technology
 News and Information Source

24/7

FREE NEWSLETTERS

Stay ahead of the i-Technology curve with E-mail updates on what's happening in your industry

SYS-CON.TV

Watch video of breaking news, interviews with industry leaders, and how-to tutorials

BLOG-N-PLAY!

Read web logs from the movers and shakers or create your own blog to be read by millions

WEBCAST

Streaming video on today's i-Technology news, events, and webinars

EDUCATION

The world's leading online i-Technology university

RESEARCH

i-Technology data "and" analysis for business decision-makers

MAGAZINES

View the current issue and past archives of your favorite i-Technology journal

INTERNATIONAL SITES

Get all the news and information happening in other countries worldwide

JUMP TO THE LEADING i-TECHNOLOGY WEBSITES:

IT Solutions Guide

Information Storage+Security Journal

JDJ

Web Services Journal

.NET Developer's Journal

LinuxWorld Magazine

Linux Business News

Eclipse Developer's Journal

MX Developer's Journal

ColdFusion Developer's Journal

XML Journal

Wireless Business & Technology

Symbian Developer's Journal

WebSphere Journal

WLDJ

PowerBuilder Developer's Journal

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute" xmlns="*">
<!--CustomPanel /-->
    <mx:Button label="CreatePanel" click="createComponent('CustomPanel')"/>
<mx:Script>
    <![CDATA[
        private var displayObject:DisplayObject;
        private function createComponent(componentName:String) : void {
            var clazz : Class = getDefinitionByName(componentName) as Class;
            displayObject = DisplayObject(new clazz() );
            this.addChild(displayObject);
        }
    ]]>
</mx:Script>
</mx:Application>
```

If you run the application and click the “Create Panel” button, the code terminates abnormally.

The reason for this error is that many times mxmmlc complements classes with additional initialization code at the SystemManager level. But if the relevant classes are completely shielded from mxmmlc it’s absolved from taking care of them. Let’s explain this in detail. Please have another look at the generated SystemManager of the FlexApplication that we had in the previous example:

```
package {
import mx.managers.SystemManager;
import flash.utils.*;
import flash.system.ApplicationDomain;
```

```
import mx.core.IFlexModuleFactory;
public class _FlexApplication_mx_managers_SystemManager extends mx.managers.SystemManager implements IFlexModuleFactory {
    public function _FlexApplication_mx_managers_SystemManager() {
        super();
    }
    override public function info():Object {
        return {
            “currentDomain”: ApplicationDomain.currentDomain,
            “layout” : “vertical”,
            “mainClassName” : “FlexApplication”,
            “mixins” : [“_FlexApplication_FlexInit”, “_activeTabStyleStyle”, ... “_ControlBarStyle”, “_PanelStyle”, “_CustomPanelWatcherSetupUtil”]
        }
    },
    “rsls” : [{url: “FlexLibrary.swf”, size: 1}]
};
}
//_FlexApplication_mx_managers_SystemManager
}
```

If we compare this SystemManager with the one generated for FlexApplication2 we’ll see that in the latter case the mixins array is short of the three values: “_ControlBarStyle,” “_PanelStyle,” and “_CustomPanelWatcherSetupUtil.”

The classes referenced in the mixins array take part in the initialization sequence of the application upon the initial load. In particular, CustomPanelWatcherSetupUtil is the class that facilitates the binding for the variable instanceNumber of CustomPanel. In the case of FlexApplication2, this part of the initialization “gets forgotten.”

Now that you have seen the differences between the applications manifested in <ApplicationName>_mx_managers_SystemManager files, you’re likely to notice the others, such as files <ApplicationName>_FlexInit.as. For example, in Chapter 8 we annotated the

EmployeeDTO class with the metadata keyword RemoteClass:

```
[RemoteClass(alias="com.theriabook.composition.dto.EmployeeDTO")]
```

In the case of SimpleAutoCompleteWithDynamicDataDemo, this metadata annotation results in the following:

```
package {
[Mixin]
public class _SimpleAutoCompleteWithDynamicDataDemo_FlexInit
{
    . . . .
    public static function init(fbs:IFlexModuleFactory):void
    {
        . . . .
        flash.net.registerClassAlias(
            “com.theriabook.composition.dto.EmployeeDTO”,
            com.theriabook.composition.dto.EmployeeDTO
        );
        . . . .
    }
} // FlexInit
} // package
```

And again, the registration snippet


```
flash.net.registerClassAlias(
    “com.theriabook.composition.dto.EmployeeDTO”,
    com.theriabook.composition.dto.EmployeeDTO
);
```

wouldn’t have materialized if the compiler didn’t know about the EmployeeDTO.

All in all, MXML files as well as metadata-annotated ActionScript classes might not get the expected initialization support if you put them in RSL and don’t explicitly reference them in the calling application.

Conclusion

The second part of this article will discuss dynamic linking, self-initialized libraries, RSL vs. custom loading of the dynamic libraries, embedded applications and SWFLoader, and optimization of application size.

To order a copy of *Rich Internet Applications with Adobe Flex & Java*, please go to www.riabook.com. 

Advertising Index

Advertiser	URL	Phone	Page
Adobe	adobe.com/go/designer		2, 52
CFDynamics	www.cfdynamics.com	866-233-9626	5
CFUnited	www.cfunderground.com		19
Community MX	www.communitymx.com		15
Cynergy			20, 21
EdgeWeb	http://edgewebhosting.net	1-866-334-3932	6
HostMySite	www.hostmysite.com		51
InterAKT	www.interAKTonline.com/macromedia	4031 401.68.19	27
Integral	www.fusiondebug.com		3
Omniture			11
PaperThin	www.paperthin.com		37
Stream57	www.stream57.com	212-909-2550x1012	41
Vitalstream	www.vitalstream.com	800-254-7554	8, 9

Other companies in this magazine spent a lot of time on pretty ads. As you can see, we did not. We spent our time hiring the best people and training them to deliver outstanding support for your website. We spent our time building a state of the art datacenter and staffing it with people who care about your website like it's their own. Compassion, respect, credibility, ownership, reliability, "never say no," and exceed expectations are words that describe our service philosophy. From the first time you interact with us, you'll see what a difference it really makes. And you'll also forgive us for not having a pretty ad.





Beyond boundaries.

In the right environment, imagination has no limits.

MAX 2006, the annual Adobe user conference, offers a unique opportunity to learn about Adobe software, interact with industry experts, connect with the Adobe community, and explore ideas yet to be imagined.

Choose from over 100 different workshops and hands-on sessions presented by Adobe experts and industry leaders. Exchange ideas with other community members at a variety of networking events. Experience current and emerging Adobe technology.

Join us for MAX 2006 this **October 23-26** in Las Vegas to learn, network, and move beyond the boundaries of what you believe is possible.

Online registration ends Oct 16.
Register now: www.adobe.com/go/wddj



Beyond boundaries. The 2006 Adobe conference.

Adobe and the Adobe logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries.